44 Schriften aus der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg

Designing Complex Software Architecture

Guidelines on Pattern Selection and Microservice Identification Methods

Vincent Georg Lauber

University of Bamberg Press

44 Schriften aus der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg

Contributions of the Faculty Information Systems and Applied Computer Sciences of the Otto-Friedrich-University Bamberg

Schriften aus der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg

Contributions of the Faculty Information Systems and Applied Computer Sciences of the Otto-Friedrich-University Bamberg

Band 44

University of Bamberg Press
2024

Designing Complex Software Architecture

Guidelines on Pattern Selection
and Microservice Identification Methods

Vincent Georg Lauber

University
of Bamberg
Press
2024

Bibliografische Informationen der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de/ abrufbar.

Herstellung und Druck: docupoint, Magdeburg
Umschlaggestaltung: University of Bamberg Press

ISSN: 1867-6197 (Print)
ISBN: 978-3-98989-038-1(Print)
eISSN: 2750-8560 (Online)
eISBN: 978-3-98989-039-8 (Online)

URN: urn:nbn:de:bvb:473-irb-1045284
DOI: https://doi.org/10.20378/irb-104528

Dedicated to Annegret and Cosima.

Why is composing symphonies tough? I don't know.
It's just very few people in the world can do it well.
And I think that's the case with upfront design.
It is very hard to do well.
Martin Fowler

Table of Contents

List of Abbreviations

MS	Microservice
MSA	Microservice Architecture
SOA	Service-Oriented Architecture
ESB	Enterprise Service Bus
SOAP	Simple Object Access Protocol
AISeL	Association for Information Systems eLibrary
CSV	Comma Separated Values
MSADF	Microservice Architecture Design Framework

List of Figures

List of Tables

1 Introduction

1.1 Strategic Architectural Design

The concept of unmanaged mutation leading to chaos is a fundamental principle observed not only in biological evolution—where sexual reproduction introduces new gene combinations that increase genetic entropy—but also in the evolution of software architecture. In the former, natural selection guides this evolution by optimizing adaptation to environmental needs, selecting only the fittest individuals for reproduction, and introducing order amidst chaos, whereas in the latter, the need to adapt and optimize in response to environmental demands drives continuous change. Historically, straightforward monolithic architectures were standard in enterprise application development. Over time, driven by the need for better separation of concerns, they have been contrasted with more complex architectural patterns such as service-oriented architectures (SOA) and microservices (MS). Today, monolithic and service-oriented architecture, specifically microservices, are the dominant paradigms of modern enterprise application development (Blinowski et al., 2022, p. 20357). Despite the prevalent adoption of monolithic and microservice architectures, there is no universally superior architectural style. The choice of an appropriate architectural style requires careful consideration of an organization's specific constraints, needs, and goals. It should be based on a clear understanding of what the business aims to achieve and how different architectural styles could impact those goals. The process should involve a deliberate evaluation of potential styles, considering how each option could support the organization's specific requirements and long-term objectives. Often, organizations struggle to determine if transitioning to or developing within a new architectural style is beneficial, typically due to a lack of understanding of the essential technical and business criteria involved in architectural decision-making. Without a solid grasp of these criteria and their careful consideration, companies may impulsively choose a style that seems beneficial but later leads to alignment issues with their specific needs, resulting in suboptimal outcomes. For instance, while cloud architectures have become popular following the trends of monolithic, SOA, and microservices architectures, there's been a notable shift with some businesses moving back to on-premise solutions due to the unforeseen challenges and costs associated with cloud services (Jewargi, 2023, p. 1). Moreover, the process of adopting a new architectural style, inclusive of its realization, is fraught with risk. Companies often pursue these changes in hopes of strategic or economic benefits. However, without deep knowledge of the critical criteria for selecting the right architectural styles, these decisions can lead to costly

complications, emphasizing the need for a thorough assessment of its suitability before proceeding to avoid future regrets. The complexity of such decisions underscores the need for a thorough evaluation process that considers both technical requirements and broader business objectives as the guiding criteria of the choice.

Strategic planning and thoughtful design is important to understand, why a particular architectural style is chosen and how it will be implemented before making any changes. The development of an architectural design can serve to assess the implications of the choice and estimate its impacts on the organization's operations. This proactive assessment helps estimate the impacts of the architectural choice on the organization's operations and ensures alignment with strategic goals and operational needs, potentially mitigating dissatisfaction, or uncertainty about the changes. However, many organizations might informally cut and adjust their services based on perceived needs rather than through a structured decision-making process. Uninformed architectural style choices and informal service delineation are common pitfalls that often lead to uncertainty about the outcomes and dissatisfaction with the results. The feasibility of implementing a particular architecture style also heavily depends on the starting conditions, such as legacy systems and the business's capability to support new technologies. Businesses must estimate the effort and costs involved in transitioning to a new architecture to ensure that they choose the most suitable option for their specific conditions.

In complex architectural styles such SOA and microservice architecture (MSA), the need for formal identification methods becomes more pronounced. These architectures are well-suited for a quality-driven approach, where quality measurements at design time are critically important. Such measurements provide essential insights that influence foundational decisions and are vital in environments where multiple services interact intricately. The experience and informal knowledge of architects, while invaluable, can be significantly enhanced by integrating objective, formal solutions. This is especially crucial in complex systems that may be beyond the comprehensive understanding of any individual, therefore requiring formal methodologies to ensure thorough evaluation and optimization beyond informal heuristics. In essence, adopting a strategic approach to design helps in effectively managing complex architectures and in crafting solutions that meet both current and future demands. The inherent complexity of patterns like SOA and microservice architecture implies the critical importance of objective, evidence-based quality assessments. These systems might require specialized quality metrics to effectively manage and

optimize their designs. Quality measurements during the design phase provide essential insights that influence foundational decisions. Interpreting metrics can be complex since they are often interrelated and can evaluate multiple quality criteria depending on their context. Practitioners need assistance in deciding which objective assessments are most appropriate for their specific needs in architectural evaluation.

This thesis provides crucial guidance on identifying key decision-making criteria and designing complex architectures with a quality-driven approach tailored to specific organizational conditions. It aims to enable effective quality assessments and ensure that architectural choices align with strategic goals and operational needs. Through a structured evaluation process that incorporates these criteria, organizations can make informed decisions that enhance both their architectural outcomes and operational efficiency. To achieve this goal, the thesis serves as a consultative resource to mitigate risks associated with transitioning to new architectural styles. It guides practitioners through the selection and design processes, focusing on understanding the relevant criteria essential for making informed decisions. This includes identifying crucial criteria and exploring various approaches for implementing the chosen architectural style, ensuring that architectural choices are technically sound and strategically advantageous. This guidance includes a detailed overview of architectural alternatives and differentiated approaches based on starting conditions and data availability, which helps organizations navigate the complexities of architectural transitions effectively. Furthermore, it examines various approaches to microservices' architectural design, establishing the basis for incorporating these in the decision-making process. This involves quantification of quality criteria, mapped to the initial conditions and the overall approach, to tailor strategies to organizational needs. By establishing these foundations, this thesis aims to enable software architects to make informed, data-driven decisions, ensuring that architectural changes align with the organizational goals. It ensures that the strategies are applicable and practical within the specific context of the organization, considering the company's pre-existing assets and future goals. This structured approach aims to streamline the architectural decision-making process and support the adoption of suitable architectural styles that meet the unique characteristics and requirements of the organization.

1.2 Related Work

Appendix Table A.1 shows the systematic literature reviews and mapping studies on service-oriented architecture, microservice architecture and quality attributes in their context, that are included in this chapter. It presents each study with its year of publication and research questions.

Joachim (2011) examines the conceptualization of SOA, the factors influencing its effective adoption on an organizational level and resulting business values, focusing only on a single architectural style. Niknejad et al. (2020) investigate significant success factors that organizations need to understand to maximize the benefits of SOA (p. 2).

Several empirical studies concentrate on quality attributes of service-oriented architecture: Mahdavi-Hezavehi et al. (2013) focus on the variability of quality attributes in systems based on service-oriented architecture and suggest practical implications for their handling (pp. 320–321). O'Brien et al. (2007) list quality attributes important to SOA and discuss them to identify those of most significance to most significant to this context.

Bogner et al. (2017) present the "maintainability Model for Services" (p. 2), aiming for "fast and automatic evaluation, simplicity, and practical applicability" (p. 4) to systems based on either service-oriented or microservice architecture. Their model is asserted to be equally applicable to service-oriented architecture and microservice architecture, highlighting the relevance of studies on SOA to MS (p. 3). Stating, that organizations "will greatly benefit from an explicit understanding of maintainability" (p. 1), the quality model also provides metrics to evaluate this quality attribute. The authors deem the selection of services an important and difficult problem for practitioners and highlight the importance of "having a reasonable number of measurements with fast feedback" (p. 1), especially in the context of microservice architecture, where the inadequacy of existing object-oriented quality metrics to the assessment of microservice makes a reliance on established metrics difficult.

The present study sets out to identify adequate quality criteria for both diverting architectural styles and widens the scope to monolithic architecture. It also provides a more comprehensive categorization of quality metrics applied during architectural design in the context of microservice architecture. It is also not limited in scope to examining only one quality criterion but includes all relevant quality criteria.

Extant systematic mapping studies regarding MSA provide an overview of the field and its intersections with the work at hand. The systematic mapping study

by Pahl and Jamshidi (2016) on the different types of microservices and motivations, techniques and challenges for adopting MSA focuses their taxonomical classification and comparison in their microservice characterization framework (pp. 139–140). They study practical motivations behind using microservices and "the main reasons for organizations to architect in a microservices style" (p. 139). The study only considers 21 studies, published until 2015 and presents research trends and benefits of microservices, but cannot reach a conclusive analysis, stating the low maturity of research on microservices as the reason (p. 146).

Di Francesco et al. (2017) present an early overview of MSA in their systematic mapping study, focusing trends, and potential directions of MS. They define and apply a classification framework to incorporated studies to investigate quality attributes addressed by them and to identify and confirm advantages of MSA (pp. 25–27). They "aim at characterizing the intensity of scientific interest on architecting with microservices" (p. 22), provide a foundation for classifying the research on architecting microservices to assess the readiness of the current state of the art on architecting microservices to be transferred to industrial adoption (p. 22).

In a later study, Di Francesco et al. (2019) build upon their earlier overview to suggest "academic venues where new results about architecting microservices may be better received" (p. 78) and "identify the research groups which are prominently contributing in the field" (p. 78). They discuss, how research on architecting microservices can be transferred back to industry (p. 79). Their coarse-grained classification framework for the field of microservice architecture aims to help practitioners position themselves according to organizational and technical needs (pp. 79–80).

Several systematic mapping studies on microservices focus on quality attributes. The systematic mapping study on MSA by Alshuqayran et al. (2016) gives an overview on architectural challenges, quality attributes, diagrams, and views (p. 1).

They highlight gaps in the state of research on microservices, name quality attributes presented in the literature and "alternative terms and expressions of similar meaning" (p. 6) for their categorization but provide little discussion about identified quality attributes and their characteristics.

S. Li et al. (2021) identify frequently concerned quality criteria relevant to microservices and tactics to improve their quality in respect to certain quality criteria (p. 8). However, they completely omit the examination of other

architectural styles, organizational effects and requirements, or considerations regarding microservice identification, in their assessment of quality attributes (p. 3).

Several studies on architecture development and migration to microservice architecture have been conducted. Appendix Table A.2 presents literature reviews on microservice identification, including studies on quality assessments in its context, with their research questions.

The state of research regarding SOA has matured for longer than research on microservices. Research on its predecessor is relevant to microservices, but limited in its applicability, due to the differences between the two. The domain of service identification in the context of service-oriented architecture is different from microservice identification and the suitability of SOA service identification approaches to microservice identification is limited, due to the diverging constraints of architectural conformance. However, the field of service identification approaches for SOA is more mature than research on microservice identification and provides a more comprehensive overview of the extended field of service identification that complements the subdomain of microservice architecture development. The present study focuses on the identification of microservices, and by extension the architectural design of microservices.

Klose et al. (2007) focus on the business perspective in their discussion of service identification in the context of SOA (pp. 1809–1812). As part of their discussion, they compare different service identification approaches (p. 1812).

Abdellatif et al. (2018) review literature on Service identification (SI) approaches for SOA migration. They examine the types of SOA and reasons for the migration of legacy systems to it and the types of these systems. Approaches used for "application migration, in general, and SI in particular" (p. 2) are also studied, regarding their inputs and techniques used for identifying services in legacy applications. With the additional examination of "desired services quality criteria for SI in industry" (p. 7), the study is similar in design to the work at hand, but in addition to pertaining to the field of SOA, it only touches on central topics of this thesis, such as initial environments of and reasons for the migration.

Abdellatif et al. (2021) conduct a literature review on service identification methods to produce service-oriented architectures and the subsequent development of service-based systems. They review 41 studies from 2004 to 2019 and limit their scope to identification approaches, that modernize monolithic legacy systems (p. 3). The resulting taxonomy classifies service identification approaches for the migration of monolithic legacy systems (p. 16). The study identifies

inputs used in said approaches, steps of the identification process, its generated outputs, and tries to assess the quality of resulting architectural designs (pp. 3–15). While the study limits its scope to legacy system migration and the domain of migration to SOA is different from microservice identification, their concepts are similar.

Reddy et al. (2009) establish core principles of SOA as guidelines for "evaluating the suitability of existing assets" (p. 51) for their migration to SOA and survey extant "metrics and guidelines that could be helpful in evaluating these principles" (p. 51). They aim to help organizations moving to SOA understand the necessary effort of migrating their assets to SOA and assist in proper service identification (p. 61).

Schröer et al. (2020) study microservice identification approaches, considering starting points and the evaluation of microservice architectures during the design phase. They give a broad overview on the comparative assessment of microservice architectures through an examination of which architectural styles their architecture is compared to (pp. 162–163). They observe monolithic architecture as the only architectural style used for the comparative assessment of microservice architectures and highlight its importance as a baseline (pp. 162–163). Their results highlight the importance of monolithic architectures as a means of comparison. The comparison of microservice architecture to other architectural styles, put in context to microservice identification approaches, resembles the approach of the work at hand, albeit older and considering only 31 studies.

There is evidence for companies choosing to keep their monolithic architecture or migrate back to it from microservices: Mendonça et al. (2021) describe reasons for the migration from microservices to monolithic architecture and assess the advantages and disadvantages of MSA in comparison to monolithic architecture without performing a systematic literature review (p. 20).

The identification of service boundaries within a monolith is also relevant to microservice identification, representing an approach both that can be both complementary or alternative to microservice identification. Goncalves et al. (2021) describe the modification of a monolithic system to prepare it for the migration to microservices (pp. 54–58).

As one of several secondary studies researching microservice identification and migration, Velepucha and Flores (2021) systematically map benefits and challenges in the process of microservice identification, but focus solely on migrating monolithic systems to microservices (pp. 137–140).

Fritzsch et al. (2019) review literature on microservice identification approaches and extensively classify approaches for the migration from monolithic to microservices-based systems. They inductively develop a classification framework based on incorporated literature and focus on different migration strategies (pp 133–135). Despite the explicitly stated focus on monolith refactoring, greenfield approaches are mentioned in the classification framework (pp. 133–135). However, no attention is paid to quality metrics and quality-driven microservice identification.

Oumoussa and Saidi (2024) examine and classify studies on microservice identification to "describe their objectives, evaluate methodologies used, and discuss trends" (p. 23390). Their presentation of the current state-of-the-art in microservice identification research for monolith decomposition, in terms of methodologies and tools, uncovers a host of challenges and complexities (p. 23395). They further categorize evaluation metrics and present annual tendencies of their use (p. 23394). Moreover, studies applying quality criteria pertaining to characteristics of the business domain and system are found to exhibit a clear lack of "transparent evaluation criteria" (p.23395) and highlight the reliance of structural metrics on the expertise of developers for their correct interpretation. This work expands upon the interpretation of structural metrics and examines the quality criteria directly measured by them, while additionally focusing on quality criteria of the organizational domain.

Ponce et al. (2019) study migration techniques for the transition from monolithic to microservice architecture based on the analysis of migration methods, which they unidimensionally categorize into model-driven, static, and dynamic analysis approaches, with their applications, validation techniques and migration challenges (pp. 1–7).

Waseem et al. (2021) focus on testing and monitoring of systems based on MSA. They conduct a survey and combine it with follow-up interview "to get deeper insights into the survey results" (p. 7). They touch on architectural design but give a broad view on the topic, examining job descriptions of practitioners in microservice identification and its prominence across different countries and industries (p. 4).

Cojocaru et al. (2019) review literature on the migration of monolithic applications to microservice architecture. They evaluate methods for the collection of data for microservice identification from monolithic environments and relate them to quality criteria used in the evaluation of microservice architecture (p. 86). Omitting greenfield approaches, they identify quality assessment criteria, and aim to define "a minimum required set of quality assessment criteria

applicable to any microservice resulted from migrating a monolithic application to a microservice architecture" (p. 85). They then refine the set by imposing meaningfulness, which "considers the applicability of a quality attribute to a microservice at component level" (p. 85) and feasibility of implementation, by means of workload descriptions identified in the literature "workload description was identified in literature" (p. 85).

Carvalho et al. (2019) report "a survey performed with specialists to assess what criteria are useful to extract microservices during the migration to a microservice architecture" (p. 29). They conduct interviews to analyze, how criteria are measured and categorize them regarding their applicability and appropriateness to different contexts and conclude, that the usefulness of quality criteria depends on their context (pp. 27–29).

Capuano and Muccini (2022) also study quality attributes in the context of microservice identification. They study quality-driven approaches to microservice identification and quality attributes analyzed during it, without paying attention to the initial environments of microservice identification (pp. 121–122).

Wolfart et al. (2021) investigate the migration from monolithic systems to microservices, and present a "process to conduct the migration, for guiding practitioners on modernizing legacy systems with microservices" (p. 150) with their roadmap, depicted in Appendix Figure A.1. They outline tasks across the four phases of initiation, planning, execution, and monitoring of the process (pp. 152–155).

Regarding the initiation phase, they identify driving forces of the migration (p. 151).

Abgaz et al. (2023) also research the literature on microservice identification approaches and devise a framework for monolith to microservices decomposition (p. 4230). The framework is structured along the phases of monolith decomposition process (p. 4230). It orders research areas in respect to their maturity, as indicated by the recency of research focusing them, to find out how microservice identification approaches have developed (p. 4230). Phase V of the framework depicted in Appendix Figure A.2 views architectural evaluation through the lens of the approaches taken in studies, that evaluate their results (p. 4223). They do not consider quality criteria and metrics used in the assessment of microservice candidates.

1.3 Research Gaps and Objectives

Understanding the criteria for choosing between architectural styles is crucial, with business requirements and organizational criteria playing a significant role. These elements, which are known in advance, inform the technical needs and desired qualities of architectural styles, guiding the selection process based on key quality criteria relevant to the business. Extant work on the driving forces for the selection of architectural styles and migration towards microservices is found to ignore the organizational and operational perspectives on architecture modernization (Wolfart et al., 2021, p. 150). Wolfart et al. (2021) therefore highlight the need for research "covering the entire modernization process and dealing with organizational, operational, and technical aspects" (p. 150) and its importance to "the proper use of microservices as a strategy to modernize legacy systems" (p. 150).

This study dives deeper into the first two phases of the roadmap by Wolfart et al., giving an enhanced view of the selection and development process up to the architectural design. It aims to bridge this gap in integrating the organizational, operational, and technical perspectives on microservices. It extends this perspective by also examining service-oriented architecture, the closest relative and direct ancestor of microservice architecture. The examination of both architectural styles is meant to delineate them more clearly and to supplement the individual assessments of each by the others. Extant research predominantly focuses on specific architectural styles, often predetermining their choice without providing a comparative assessment across different styles, revealing a significant gap in studies on selecting architectural styles. There's a notable absence of guidelines for choosing among alternative architectural styles, especially a comparative analysis of service-oriented, and microservice architectures based on quality criteria. This lack of comparative research hinders the ability to assess these architectures side by side, which is essential for informed decision-making. Addressing this gap, this thesis expands and builds upon the initiation phase of the roadmap by Wolfart et al. (2021, p. 150) in Appendix Figure A.1 and makes it a central theme, dedicating a significant share of it to the analysis of quality criteria as factors for the selection of architectural styles. It allocates one of its literature reviews to a thorough comparison of architectural styles, aiming to illuminate the process by considering the complexities involved in such a comparative assessment. Juxtaposing the architectural styles facilitates a deeper understanding of the factors influencing their selection and the distinct advantages of each, contributing to a more nuanced and comprehensive framework for architectural decision-making.

Monolithic architecture is not complex in the sense of software system decomposition into multiple decentral components, orchestrated or choreographed to form cohesive systems. It therefore falls out of the scope of complex contemporary software architectural patterns, which this work aims to study. However, the process of selection between complex software architectural styles, specifically of microservices, being strongly interconnected with the comparative assessment of architectural styles, makes monolithic architecture an important means for their comparison, as evidenced by Schröer et al. (2020, pp. 162–163). The incorporation of studies on the comparison between service-oriented architecture and specifically of microservice architecture to monolithic systems, serves to better inform the choice of microservices, setting it against the two predecessors it combines. This study significantly expands upon the coarse examination of how microservices are compared by Schröer et al. (2020), on how microservices are compared. It studies not only how MSA is compared, but extends to view to SOA, aiming to research the comparisons of both styles with each other, and with monolithic architecture.

Wolfart et al. (2021) further state, that "despite the growing interest in the modernization with microservices in both practice and research, there is still a lack of comprehensive studies on how to conduct such a modernization" (p. 150). This work aims to provide an overview of microservice identification approaches reported in the literature. It categorizes them to analyze their differences and to give context on their constituent activities, artifacts, and techniques. Examining starting points of identification methods, Schröer et al. (2020) state, that "approaches for greenfield development regarding microservices identification have still less attention in research" (p. 162). This work aims to address this research gap. In the context of the comprehensive perspective on the selection of microservices provided by the results of its first literature review, this work comprehensively studies microservice identification, including not only the migration from legacy systems to microservices, but also considering greenfield approaches. Schröer et al. (2020) further identify a need for research on the usage potential of artifacts produced in greenfield approaches, which this study also aims to service by uncovering interdependencies between artifacts, techniques, and activities in microservice identification approaches and discussing fitting sets of these concepts (p. 162). The examination of the applicability of parts of the development process to their initial environments guides the evaluation of the suitability of pre-existing assets for migration. The present thesis dives deep into the differences between approaches starting from different environments and aims to give an in-depth understanding of their effects on microservice architecture design. To achieve this, it differs from the

framework by Abgaz et al. (2023) in several ways: The gaps between process phases are bridged by means of an augmented taxonomy, qualifying complex concepts (p. 4230). Furthermore, the taxonomy of microservice identification approaches, while also being structured along the phases of the process leading up to microservice identification, also includes greenfield approaches. The subsequently produced framework takes these initial environments as the dimension, along which taxonomic concepts are ordered, and presents their suitability concerning the starting points of microservice identification. The taxonomy and framework of the present study focus on the architecture development process, leading up to and culminating in the design of microservice architecture. Thus, they exclude subsequent phases, such as Phase VI of the framework depicted in Appendix Figure A.2. Furthermore, the phases leading up to microservice identification are examined in more detail. This thesis thereby aims to highlight both the central quality criteria in the context of the chosen architectural style and the appropriateness of their choice through the feasibility of their implementation in respect to pre-existing assets.

This study addresses a notable gap in the literature on quality-driven microservice architectural design, particularly in relation to initial development environments. Schröer et al. (2020) highlight the necessity for formal methods in microservice identification and the application of quality criteria during this process, underlining the need for comprehensive research in this area (pp. 162–163). This research aims to bridge these gaps by exploring how microservice architectural design emerges before identifying relevant metrics. Quality-driven approaches in architectural design necessitate an understanding of the feasibility of evaluating quality criteria throughout the architectural design process. The applicability of such approaches to the adoption of microservice architecture depends not only on identifying relevant criteria but also on assessing the feasibility of their application, given the initial conditions. Furthermore, there's a specific research void concerning quality-driven greenfield approaches to microservice identification. The measurement of quality criteria during architectural design likely varies based on the initial conditions and available assets for the development process. Metrics can serve to evaluate a multitude of quality criteria, whose importance may differ based on the specific prerequisites and conditions of organizational environments. Practitioners' situations thus dictate the prioritization of these quality criteria. This work seeks to pinpoint quality metrics and the criteria they measure, including both operational and organizational aspects. By doing so, it aims to clarify which quality assessments are viable for initial environments, thereby guiding through the myriad of

identification approaches and illuminating various pathways for microservice identification and architectural design.

This study embarks on a comprehensive exploration of quality criteria across the two divergent complex architectural styles, extending its analysis to include monolithic architecture for comparison and offering a detailed categorization of quality metrics utilized during the architectural design phase in microservice architecture. It aims to identify all relevant quality criteria and discern the most pertinent ones for SOA, and MSA. This holistic approach aids in making more informed decisions between architectural styles by highlighting the feasibility of implementing and assessing various quality criteria during the design phase of microservice identification. Further enriching the architectural design process, this work examines the implications of architectural style selection on migration or implementation effectiveness. A quality-driven approach thus necessitates an understanding of the relevant assets and quality criteria, not only for the choice of an architectural style but also to its execution, particularly its architectural design. Overall, the goal is to assist organizations planning to transition to a complex service-oriented architectural style. This guidance extends to practitioners selecting an architectural style and developing an architectural design for microservices, clarifying the effort required to migrate their assets and aiding in effective microservice identification.

1.4 Research Questions

This section outlines the research questions of the systematic literature review designed to assess and compare the architectural styles SOA and MSA, with a focus on identifying the key quality criteria critical for their selection. These criteria are based on their benefits, challenges, differences, and appropriate contexts of use. The research aims to systematically understand, compare, and interpret the distinctive characteristics, strengths, weaknesses, and applicability of each style. To address these aims, the research questions presented in Table 1.1 have been formulated to guide this systematic literature review:

No.	Research Question
RQ 1	What quality criteria related to SOA and MSA are presented in the literature?
RQ 1.1	What are the most emphasized quality criteria for SOA?
RQ 1.2	What are the most emphasized quality criteria for MSA?
RQ 2	How are SOA and MSA compared in existing studies?
RQ 3	Which quality criteria most significantly differentiate SOA from MSA?
RQ 4	How do SOA and MSA compare in terms of their associated quality criteria?

Table 1.1: Research questions on the comparative assessment of SOA and MSA

RQ 1 seeks to examine the quality criteria related to SOA and MSA, including studies comparing them to each other or to monolithic architecture. The goal is to derive a foundational understanding for further analysis and to identify a structure for the comparative assessment of architectural styles that can guide their selection. The following sub-questions aim to identify the most significant quality criteria observed in the literature on service-oriented architecture and microservice architecture, with RQ 1.1 focusing on the exploration of the most frequently highlighted quality criteria in the literature for SOA. Sub-question RQ 1.2 identifies the most frequently emphasized criteria specifically concerning MSA. The inquiry of RQ 2 focuses on the comparative assessment of architectural styles, investigating how they are assessed either comparatively or in isolation within the literature. RQ 3 aims to compare the assessments of SOA and MSA to identify the criteria that show the greatest differences in emphasis between the two architectures, exploring which criteria are predominant in each and which are common to both. RQ 4 involves comparing architectural styles, using the quality criteria identified in RQ 1. This comparison integrates analyses of benefits, challenges, best practices, and recommendations as reported in the literature. It aims to inform and provide guidelines on when each architectural style is most appropriately and suitably used and to explore potential strategies concerning their quality criteria, thereby helping practitioners make informed decisions about their adoption. This inquiry extends to discussing the results of the comparison, seeking to give guidelines on when each architectural style is most appropriately used and to explore potential strategies for the application of the architectural styles concerning their quality criteria. Additionally, this question explores the underlying motivations for the importance of the

most frequently emphasized quality criteria, shedding light on why certain criteria are prioritized in the respective contexts of SOA and MSA.

Focusing on Microservice architecture design, this study sets out to answer the research questions presented in Table 1.2.

No.	Research Question
RQ 5	What approaches are suggested in the literature to support microservice architecture design?
RQ 5.1	What are the primary phases of the microservice architecture design process and the major constituent elements of those phases?
RQ 5.1.1	To which types of initial environments are the proposed microservice identification approaches applied?
RQ 5.1.2	Which analysis methods are used to extract data from initial environments?
RQ 5.1.3	Which types of system representations are engineered as the basis for microservice identification?
RQ 5.1.4	Which methods and techniques are used in microservice identification?
RQ 5.1.5	Which quality criteria are evaluated during microservice identification?
RQ 5.2	How do microservice identification approaches vary in suitability and their application across different initial environments?

Table 1.2: Research questions on microservice identification

In general, RQ 5 aims to examine, evaluate, and classify the literature on approaches to microservice identification. This question centers on the identification of approaches and practices to support microservice architecture design and presents state-of-the-art methodologies proposed in microservices identification research. With RQ 5.1, the study aims to classify the processes followed by microservices identification approaches. It aims to describe the processes that underlie the service identification approaches reported in the literature. This entails gathering information about them across five main dimensions: their initial environments, the methods of analyzing these starting points to collect data, its transformation into a system representation, methods of the subsequent microservice identification and quality evaluations during it. To answer this research question, a taxonomy of microservice identification approaches, i.e., a multi-layer classification of microservice identification approaches is presented. It is based on the systematic literature review and is structured along the software development process, considering several criteria. This classification helps practitioners in selecting a suitable microservice identification approach that corresponds to their migration needs. In the following, the research question is concretized. With RQ 5.1.1, I aim to identify the types of systems or contexts, that function as the starting points of microservice identification approaches and to which the proposed approaches have been applied.

With RQ 5.1.2, the targeted microservice identification approaches are classified based on their data collection methods. This question serves to elucidate ways, in which initial environments are analyzed to extract data artifacts. With RQ 5.1.3, the kinds of system representations, which collected data is transformed into as a basis for the microservice identification, are examined. Approaches are categorized in respect to their transformation of initially extracted data into structured system models used for microservice identification. With RQ 5.1.4, I aim to differentiate between the methods and techniques used in approaches for the identification of microservice candidates to categorize them. Here, ways of partitioning system representations, which produce the microservices architectural design, are examined. As part of this, I intend to identify the technologies used in identification methods and the degree to which identification processes are automated. With RQ 5.1.5, studies reported in the literature, that implement a quality-driven approach to microservice architecture development during its design, are analyzed. Quality metrics used for the quality assessment of microservice candidates are examined and categorized to identify the quality criteria directly evaluated during microservice identification. These quality measurements implement the optimization goals of identification processes, which this question aims to analyze in respect to the quality criteria they quantify and the frequency of their observation. With RQ 5.2, this study aims to investigate the variation in microservice architecture design approaches according to their initial environments. It seeks to uncover which concepts and strategies are best suited for specific project conditions, emphasizing the adaptation of these approaches to the unique requirements presented by different starting points. The objective is to discern how to effectively tailor or choose microservice identification strategies based on the distinct characteristics and pre-existing assets of the project's environment. This inquiry is geared towards facilitating a deeper understanding of how to navigate the selection and implementation of microservice architectures, being mindful of the developmental context.

The rest of this thesis is organized as follows:

Section 2 describes the theoretical background before Section 3 outlines the research approach taken to answer the research questions. It describes the methodology that was followed to conduct the two literature reviews and the methods used in the synthesis of their results. The subsequent chapters implement the methodology, starting with Section 4, which presents identified and synthesized quality criteria, before quantitatively analyzing and later using them in the comparative assessment of service-oriented and microservice architecture. This extends to the discussion of their differences and organizational suitability in Section 5. Approaches to microservice identification and architectural design are presented in Section 6, including their synthesis by a taxonomy and its application to the literature, enhanced by augmentative descriptors, is included for ensuring comprehensiveness and detail. Section 7 discusses these findings and presents the Microservice Architecture Design Framework, before Section 8 discusses limitations that culminate in a research agenda and presents practical implications of the findings, concluding the thesis.

2 Theoretical Background

2.1 Monolithic Architecture

Monolithic software applications consist of one piece (Götz et al., 2018, p. 168). Al-Debagy and Martinek (2018) more specifically define monolithic architecture as "an application with a single code base" (p. 149). Systems of many architectural styles fall under this definition, most prominently single-process monoliths, modular monoliths and distributed monoliths (Newman, 2021, pp. 14–15). Technically, a monolith describes an application deployed as "a single logical executable" (Blinowski et al., 2022, p. 20358), designed to run solely on one computational instance (Götz et al., 2018, p. 168). Thus, in their most basic form, these application systems are deployed to a computational system's environment as one single process, encompassing all code of the monolithic application (Newman, 2021, pp. 14–15). As such, monoliths are primarily units of deployment and any software system in which all functionality needs to be deployed as a whole is considered a deployment monolith (Newman, 2021, pp. 12–15; Wolff, 2018, pp. 2–3). This implies that all of its parts need to be released together, meaning that every time "a new version of an application is deployed, it replaces the previous version of the application in a single step" (Blinowski et al., 2022, p. 20358) (Wolff, 2018, pp. 2–3).

Consequently, the application can only be scaled by replicating instances of this process (Newman, 2021, pp. 14–15). Applications may distribute process threads across several CPUs of their host system and deploy multiple instances of the application process to scale up (Götz et al., 2018, p. 168). However, on a vertical level, each process as a comprehensive application instance is constrained to share both operating system and hardware (Götz et al., 2018, p. 168). Single-process Monoliths can be modularized and consist of separate modules that still need to be deployed together but allow for more independent development of modules (Newman, 2021, pp. 16–17). Well-defined module boundaries facilitate a higher degree of parallel work while retaining a simple deployment topology (Newman, 2021, pp. 16–17). A modular monolith's database can be decomposed along the same lines as its modules, as depicted in Appendix Figure B.1 (Newman, 2021, pp. 16–17). Although constrained to run within a single process, monoliths may still be distributed systems as they regularly rely on external databases (Newman, 2021, pp. 14–15). While in the basic case, an application in monolithic architecture "has all of its logic running on a single application server" (Benavente et al., 2022, p. 178), a distributed monolith consists of multiple services, which all need to be deployed in combination (Newman,

2021, pp. 17–18). Such distributed monoliths cause changes to the system to ripple across module boundaries (Newman, 2021, pp. 17–18).

2.2 Service-Oriented Architecture

While the origins of SOA can be traced back to the early days of distributed computing and web services, it first made "its presence felt" (Xiao et al., 2016, p. 60) in the advent of the 21st century, when it emerged as a way to combat the challenges of overly large monolithic application growth (Newman, 2021, pp. 5–6). SOA was meant to be a technical solution to changing business needs by modularizing the software architecture according to the structure of the business processes (MacLennan & van Belle, 2012, p. 3). The initial driving factor of SOA was to create a loosely coupled and modular architecture that could respond better to changes and evolving requirements in the enterprise.

MacLennan and van Belle (2012) highlight a lack of "consensus on SOA definition between industry practitioners, vendors, standardization organizations or academics" (p. 3). Since "there is no single definition that has been unanimously agreed upon by everyone" (Aggelopoulou & Pramatari, 2009, p. 435), SOA is "subject to a magnitude of different interpretations" (Beimborn et al., 2009, p. 4). The competing definitions take different perspectives on SOA, resulting in their differences (Niknejad et al., 2020, p. 1). Definitions range from those taking "a high-level business view to definitions focusing on technical aspects of SOA" (Aggelopoulou & Pramatari, 2009, p. 435), with definitions within academia falling mainly under the perspectives of "technology, business, and architecture" (Niknejad et al., 2020, p. 1).

Definitions that center on the business aspects of SOA focus on the driving factor of its rise to prominence and categorize it as a business process integration framework, meant to align business strategy with enterprise architecture (Beimborn et al., 2009, p. 4). As such, SOA is seen as a vehicle to enable "extensibility to future objectives" (Andriyanto et al., 2019, p. 282). Practitioners and researchers alike commonly adopt a combined view of SOA, incorporating a technological perspective into the abovementioned business view (Niknejad et al., 2020, p. 1). Joachim (2011) characterize it in this twofold manner, on the one hand viewing SOA as an approach to software system development, that transcends traditional development of IT systems and aligns "IT with business processes for enhanced efficiency and effectiveness" (p. 5), an alignment crucial to ensuring congruence of technological and organizational goals. The authors contrarily also highlight the aspect of SOA being constituted by independently

operating services forming a flexible and loosely coupled system to encapsulate business functionality (p. 5). Most scholars agree on this architectural concept's "focus on breaking each business process into smaller blocks" (Niknejad et al., 2020, p. 1) encapsulated by services. Services then constitute the "units of standard business functionality that are connected to each other to make a unified business process" (Niknejad et al., 2020, p. 1).

Services

Xiao et al. (2016) refer to the early IBM definition of SOA as "an application framework that takes everyday business applications and breaks them down into individual business functions and processes, called services" (p. 61), which proposes that SOA as an architectural style interprets software components as a set of services (Erickson & Siau, 2008, p. 2). It argues that service definition should be driven by business needs and "the value proposition be centered around the re-usability and flexibility of the defined services" (Erickson & Siau, 2008, p. 2). The technological perspective on SOA through the lens of services completes its fundamental definition as an approach to the implementation of business and technical processes as services (Mahmood, 2007, p. 498). A service can be any "piece of self-contained business functionality" (Xiao et al., 2016, p. 61), allowing for encapsulation of said functionalities at arbitrary levels of complexity.

Interoperability

What separates services in the context of SOA from modules within the same deployment unit is their method of communication: services as completely separate processes call each other via messages across a network, rather than through method calls within a process boundary (Newman, 2021, pp. 5–6). The OASIS Group defines SOA as "a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains" (Erickson & Siau, 2008, p. 2). Therefore, "services' functionality can be exposed from existing systems, purchased from third parties or developed from scratch" (Aggelopoulou & Pramatari, 2009, p. 435). From this perspective, SOA becomes a paradigm of standardized encapsulation of functionalities through web services for their integration, irrespective of their implementation. Centering on the concept of distributed ownership domains, XML.com defines SOA as "an architectural style whose goal is to achieve loose coupling among interacting software agents" (Erickson & Siau, 2008, p. 2). This makes interoperability "a key aspect of the SOA effort to enhance communication between existing systems" (Joachim, 2011, p. 4), which can take on the roles of provider and/or consumer (Erickson & Siau, 2008, p. 2).

In this sense, a service is characterized as a computation "done by a service provider to achieve desired end results for a service consumer" (Erickson & Siau, 2008, p. 2), with software agents realizing both roles on their owners' behalf (Erickson & Siau, 2008, p. 2). The 2007 Object Management Group definition conforms with focusing on SOA's interoperability and views it as an architectural style for a community of service providers and consumers, meant to achieve mutual value (Erickson & Siau, 2008, p. 2). Such a community is firstly characterized by the interplay of service providing components that publish services to produce outputs based on the execution of business functions, and the consuming components that input business function parameters with their requests to use the published services (Mahmood, 2007, p. 498). Technical service design therefore centers on facilitating interoperability through network communication (Raj & Ravichandra, 2018, p. 1533).

Being exclusively accessible via network communication, service calls are always performed by invoking functions formally exposed through a service's interface (Mahmood, 2007, p. 498). Mahmood (2007) highlights the importance of the ability of services to separate business logic from their interface (p. 498). By this defining principle, services hide all but the required information from their service contracts, obfuscating as many underlying details of their implementation as possible (Raj & Ravichandra, 2018, p. 1532). The SOA component model interrelates the functional units of service-based applications through "well-defined interfaces and contracts" (Joachim, 2011, p. 4), adding a further component to the system.

Whereas a service's underlying logic is hidden behind its interface, a service contract formally exposes the terms of the information exchange it offers (MacLennan & van Belle, 2012, p. 3). These contracts contain service description documents, outlining meta information of the service (Raj & Ravichandra, 2018, p. 1532). Service registries as agents of service discovery are the repositories containing service descriptions, informing consumers to know how services can be accessed (Andriyanto et al., 2019, p. 282; Mahmood, 2007, p. 498). They facilitate the interoperation among the various technologies used to realize providers and requestors (Andriyanto et al., 2019, p. 282). These registries provide enabling functions of efficient service management regarding "service discovery, lifecycle management and governance functions" (Xiao et al., 2016, p. 61). The vision of SOA is to overcome the technical diversities through a universally accepted standard (Andriyanto et al., 2019, p. 282). This moves the focus to the technologies the facilitate the community interactions (Erickson & Siau, 2008, p. 2).

Technological Evolution

Participants within a SOA interact according to the find-bind-execute paradigm (Mahmood, 2007, p. 498). SOA makes use of the growing prominence of the internet during the early 21st century and the emergence of web services technology and various communication protocols such as the Simple Object Access Protocol (SOAP) and Object Request Brokers according to the Common Object Request Broker Architecture (Aggelopoulou & Pramatari, 2009, p. 436; Andriyanto et al., 2019, p. 282). Early SOAP-based web services relied on the Web Service Description Language for the self-description of their interfaces (Erickson & Siau, 2008, p. 2). Growing dissatisfaction of complex heavyweight invocation protocols, such as SOAP and increasing pervasion of the world wide web, led to an "en masse switch" (Jamshidi et al., 2018, p. 26) to REST interfaces utilizing the simpler HTTP protocol focus, and their establishment as an alternative communication standard (Andriyanto et al., 2019, p. 282).

By 2009, the Enterprise Service Bus (ESB) had become one of "the most prominent technologies that implement the SOA architectural approach" (Aggelopoulou & Pramatari, 2009, p. 436). The ESB as an architectural construct fulfills tasks related to "routing, mediation, logging and [...] security" (Beimborn et al., 2009, p. 4), in order to support service-orientation. As a mediator, this centralized architectural construct enables the interaction of services, "fulfilling tasks such as routing, mediation, logging and ensuring security" (Beimborn et al., 2009, p. 4) (Xiao et al., 2016, p. 61). This places the ESB as a central transit system at the very core of an SOA (Xiao et al., 2016, p. 61).

Design Principles

MacLennan and van Belle (2012) further characterize services as being loosely coupled, reusable and composable (p. 3). Decrease of coupling between services is a key concept of SOA (Xiao et al., 2016, p. 61). This pursuit of dependency minimization underpins SOA and is expressed in its service design through the abstraction of service descriptions and the proxy-based decoupling of services, components and functionalities (Xiao et al., 2016, p. 61). "The idea being that loose coupling of services allows for their flexible combination into larger applications or processes" (Joachim, 2011, p. 4). Reusability is another central concept of this architectural style, urging for services to be built to be reusable across business applications (Raj & Ravichandra, 2018, p. 1532). Being reusable, services can also "be composed into other services" (Aggelopoulou & Pramatari, 2009, p. 435) and may consequently "require un-derlying services providing a certain sub-functionality. A service

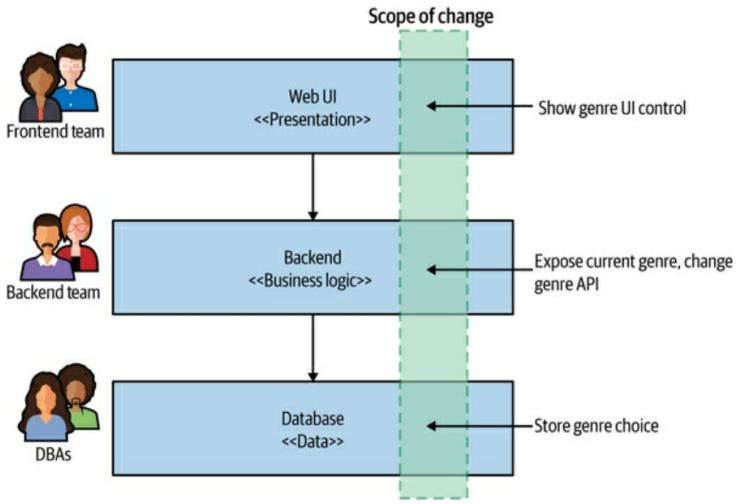
which relies on a set of services is also called aggregated service" (Götz et al., 2018, p. 168). The principle of service composability builds on this, describing the structure of service-based applications as aggregates of services, and by extension, com-posed services, forming complete software systems (Raj & Ravichandra, 2018, p. 1533). Service Composition is realized by "orchestration of complex services to achieve business goals" (Joachim, 2011, p. 5) by means of their sequential invocation.

2.3 Conway's Law in Structural Patterns

Conway's law proposes the central thesis that architecture and organizational structure are intrinsically linked, acting as reciprocal facets of the same entity (Wolff, 2018, pp. 42–43). According to this law, organizations are constrained to produce system designs, that mirror the organization's communication structures (Newman, 2021, pp. 11–12). It suggests that organizational structures dictate the design of their systems and vice versa, advocating for modularization to reduce inter-team communication needs (Wolff, 2018, 39–40). The establishment of interfaces between parts of the architecture requires coordination between development teams, incurring communication relationships between organizational units (Wolff, 2018, pp. 39–40). Communication between those working on the same module is not considered problematic, but coordination between teams responsible for different modules is ideally reduced to discussions about interfaces (Wolff, 2018, pp. 39–40).

Three-Tiered Architecture

In layered three-tiered architecture, service boundaries of enterprise applications are based on related technical functionality and traditionally consist of layers regarding presentation, application logic and data (Blinowski et al., 2022, p. 20358; Newman, 2021, pp. 7–8). In accordance with Conway's Law, such partitioning resembles a frequently observed organizational structure of frontend, backend, and database teams, with the technical architecture mirroring organizational division according to its members' core competencies (Newman, 2021, pp. 11–12; Wolff, 2018, p. 41).

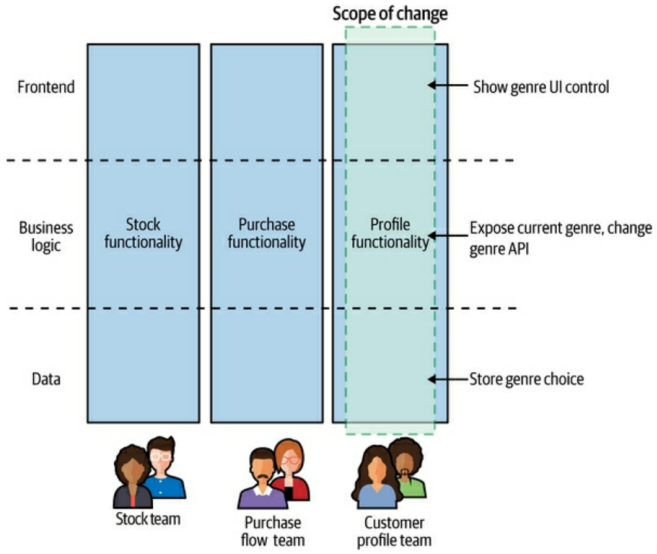


(Newman, 2021, p. 10)

Figure 2.1: Typical scope of changes to a three-tiered architecture

As depicted in Figure 2.1, three-tiered architecture separates frontend developers, backend developers and database administrators into different teams that work rather isolated from one another (Newman, 2021, pp. 11–12). Partitioning teams around familiarity along a three-tiered architecture is organizationally sound (Newman, 2021, pp. 11–12; Wolff, 2018, p. 40). It enables team members to support each other with ease and simplifies technical discourse (Wolff, 2018, p. 40). However, its impact on the architecture is easily overlooked by organizations that employ this approach (Wolff, 2018, p. 41). Layered architecture inherently supports technical cohesion and favors changes that affect multiple functionalities within the same layer but complicates changes, that necessitate modifications across its technical layers (Newman, 2021, pp. 7–8, 11–12). A key disadvantage of three-tiered organizational and architectural structures lies in the fact, that typical features as changes to a software system's functionality represent changes to business functionality, which commonly span all tiers of layered architectures and therefore require modifications in the frontend, the backend, and the database (Newman, 2021, pp. 7–8, 11–12; Wolff, 2018, p. 41). As such, when a new business functionality is introduced in such a system, interdependent changes to the affected layers need to be managed by different teams, which have to coordinate their development and deployment (Newman, 2021, pp. 11–12; Wolff, 2018, p. 41). This makes layered architectures a suboptimal choice for modifications to business functionalities and decreases the efficiency of organizations utilizing it, when it comes to rapidly deploying new features (Newman, 2021, pp. 11–12; Wolff, 2018, p. 41).

Structural Alignment with Business Functionalities



(Newman, 2021, pp. 12–13)

Figure 2.2: Architecture aligned with business functionality

In pursuit of independent development of components, the system can be divided along business functionality (Wolff, 2018, pp. 41–42). These components can serve as a basis for team formation, as Figure 2.2 demonstrates with separate teams for different processes, each working on their respective components, which encompass all layers of three-tiered architecture (Wolff, 2018, pp. 41–42). Grouping organization and architecture along business functionality gives teams end-to-end responsibility for changes to their business functionality, limiting the scope of change to one team (Newman, 2021, pp. 11–12). This can be achieved by encapsulating the business functionality across all relevant layers within a single module or service (Newman, 2021, pp. 12–13). Completely encapsulating business functionality in services ensures the prioritization of high cohesion of business functionality over high cohesion of technical functionality through the arrangement of architecture in order to make changes to business functionality that span several layers more efficient (Newman, 2021, pp. 7–8). This approach aligns with the concept of cross-functional teams, as demanded by methodologies like Scrum, with the resulting teams being designed to encompass various roles and cover a broad spectrum of tasks (Wolff, 2018, pp. 41–42). Thus, the business domain becomes the primary force driving system architecture, easing changes to business functionality as well as alignment of teams and lines of business (Newman, 2021, pp. 12–13).

It is conceivable to align architecture with functionalities and assign the responsibility for a functionality to a team, without using microservices (Wolff, 2018, pp. 43–44). Each team could develop their functionality inside the same

deployment monolith, but microservices are particularly apt in supporting the approach of functional division via the application of Conway's Law (Wolff, 2018, pp. 43–44). Coupled with the functional division, teams become more independent and require less coordination (Wolff, 2018, pp. 43–44). Alignment of architecture with communication structures implies that a change in one necessitates a change in the other (Wolff, 2018, pp. 43–44). This decreases flexibility of microservices, as shifting functionality from one microservice to another results in the functionality being maintained by another team (Wolff, 2018, pp. 43–44). Thus, aligning architectural design with organizational communication structures, as dictated by Conway's Law, not only impacts the development, deployment and maintenance of microservices but also requires a balance between architectural flexibility and functional division (Wolff, 2018, pp. 43–44).

2.4 Microservices

The Road to MSA

Independence and autonomy of services are frequently stated as defining criteria of SOA (Aggelopoulou & Pramatari, 2009, p. 435; MacLennan & van Belle, 2012, p. 3; Niknejad et al., 2020, p. 1). Despite that, it was the failure of SOA's services to sufficiently achieve their purported autonomy and independence that drove the rise of a "second-generation SOA" (Andriyanto et al., 2019, p. 282): microservice architecture.

While large systems have been partitioned into small modules long before microservices' inception to make software easier to create, understand, and further develop (Wolff, 2018, pp. 1–2), aspirations around faster development cycles had changed the way in which software is modularized and given rise to a new architectural style of functionally segmented modules, modeled around business domains and built by cross-functional teams organized around business capabilities (Baškarada et al., 2020, p. 2; Newman, 2021, pp. 3–4, 11–12; Wolff, 2018, p. 41). Early industry experts referred to explorations of the emergent alternative architectural approach under several guises (Jamshidi et al., 2018, p. 25).

According to Jamshidi et al. (2018), preceding concepts were conveyed through industry terms such as "SOA done right" (p. 25), being both testimony to microservices' firm rootedness in SOA and the dissatisfaction of practitioners with the shortcomings of SOA that gave rise to this new architectural style. Microservices as an approach to software modularization are a specific type and "evolution of SOA" (Waseem et al., 2020, p. 2), with the term *microservices*

commonly not only referring to architectural components but also to microservice architecture as a whole (Götz et al., 2018, p. 168; Wolff, 2018, pp. 1–2).

Appendix Figure B.2 presents a timeline of upcoming technological trends, that posed challenges, which increased the need for an appropriate architectural style and are addressed by microservice architecture (Blinowski et al., 2022, p. 20358). One of the factors leading to the advent of cloud-native microservice architecture was its aptness to be deployed and efficiently operate in cloud infrastructure (Waseem et al., 2020, p. 1). It emerged from "recent software engineering and ICT practices, including continuous delivery, on-demand virtualization, infrastructure automation, and small autonomous teams" (Andriyanto et al., 2019, p. 282), with Sellami, Ouni, et al. (2022) even defining microservices as "a realization of Service-Oriented Architecture (SOA) principles that are better suited for agile software development" (pp. 2–3).

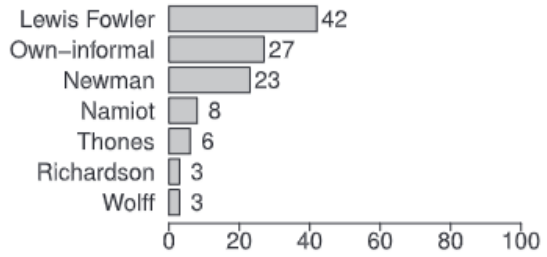
One central driving factor in the emergence of this architectural style is the empowerment of development teams to independently develop and deploy their services without reliance on other teams (Wolff, 2018, pp. 33–34). The pursuit of service development in small and independent teams as a driving factor for the paradigm shift is also highlighted by (Wolff, 2018, p. 1). Domain-driven design, guided by the principle of bounded organizational contexts, is another software development concept that played a key role in the emergence of microservices (Andriyanto et al., 2019, p. 282; Jamshidi et al., 2018, p. 26). Netflix, a pioneer of the architectural style referred to their approach as "loosely coupled service-oriented architecture with bounded contexts" (Blinowski et al., 2022, p. 20357). They adopted the microservices style in the cloud and gained many benefits from it (Waseem et al., 2020, p. 1). Similarly, Amazon described their adaptation of service-oriented architecture as a combined encapsulation of business logic with the data it operates on, with services being only accessible through published service interfaces, even before the term *microservices* gained traction in 2011 (Jamshidi et al., 2018, p. 25). Approaches to achieving agility at scale and infrastructure automation and end-to-end product ownership emerged to solve organizational issues of large-scale companies, going hand-in-hand with the principles of the emergent architectural style (Jamshidi et al., 2018, p. 26). The successful transition to "an ecosystem of small, independently deployable and independently scalable microservices" (Di Francesco et al., 2019, p. 78) of Netflix in 2014 paved the way of microservice architecture as a viable choice for other companies (Blinowski et al., 2022, p. 20357).

2.4.1 Definitions

Microservices have thus been defined different ways, with multiple definitions commonly being reported in the same study (Di Francesco et al., 2019, p. 85). Blinowski et al. (2022) highlight the organizational dimension in their definition, noting that "microservice architecture decomposes a business domain into small, consistently bounded contexts implemented by autonomous, self-contained, loosely coupled, and independently deployable services" (p. 20357).

Auer et al. (2021) echo this perspective, underscoring the approach of microservice architecture to partition applications into a concur and pronounce Microservices' proposal of strict decomposition of applications "into a subset of business-driven independent services" (p. 1).

Waseem et al. (2020) focus on technological and operational facets of the architectural style as an approach to development and deployment of software applications (p. 2). They define it as collections of "small, independent services that can be integrated through lightweight communication mechanisms like RESTful APIs [...] allowing services to be scaled independently and implemented with different technology stacks" (p. 2). These partially diverging definitions are testimony to the fact that "there has not been a wide acceptance of a specific definition" (Di Francesco et al., 2019, p. 78).



(Di Francesco et al., 2019, p. 85)

Figure 2.3: Microservice definitions

Figure 2.3 highlights that, while such informal or custom definitions are among the most frequently observed, the two most prevalent standard definitions are the 2014 Lewis & Fowler definition and the one given by Newman in 2015 (Di Francesco et al., 2019, p. 85)

The definition by Lewis and Fowler was "one of the first attempts to describe the microservice architectural style" (Blinowski et al., 2022, p. 20359), and is still popular (Di Francesco et al., 2019, p. 78). Lewis and Fowler define the microservice architectural style as "an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API" (Lewis & Fowler,

2014). They further characterize microservices as independently deployable services built around business capabilities.

At a glance, Newman Newman (2021) defines microservices as a type of service-oriented architecture composed of "independently releasable services that are modeled around a business domain" (p. 3), and states that each microservice "encapsulates functionality and makes it accessible to other services via networks" (p. 3), forming the building blocks of complex systems. He further ascribes a focus on technology agnosticism to this architectural style (pp. 3–4).

Eberhard Wolff (2018) defines microservices as a modularization concept used to partition large software systems, that realize modularization through the employment of individual programs as modules, running as their own processes (pp. 1–2). Microservices aim for independent deployments, autonomous teamwork and replaceability (pp. 38–39).

2.4.2 Design Principles

Microservices are structured along a set of vital concepts that are vital to/in making them function (Newman, 2021, pp. 5–6). The following details these central characteristics.

Autonomy and Size

Götz et al. (2018) define the scale of individual services as small and strive for a fine-granular decomposition of application module boundaries (p. 168). Most current definitions concur with this view and highlight the small scale of services as a primary defining principle (Benavente et al., 2022, p. 178; Sellami, Ouni, et al., 2022, pp. 2–3; Xiao et al., 2016, p. 62). Wolff (2018) argues against the use of the size of a microservice's codebase as a primary factor used for its definition, as it can vary widely (pp. 2–3). The term *microservices* inherently suggests that it revolves around the size of the services, indicating that the services are supposed to be small (pp. 31–32). Although seemingly implied by their name, microservices do not need to be small in regard to most metrics applicable to them; size alone is a rather insignificant factor in defining microservices (Newman, 2021, pp. 9–10).

Team Autonomy

Teams can only deploy their changes independently when working on their own deployment unit, which corresponds to their own microservice (Wolff, 2018, p. 37). To achieve encapsulation of business functions across all relevant layers within a service, it helps when it is managed and owned by a single team (Newman, 2021, pp. 12–13).

the clear allocation of each microservice to one dedicated cross-functional team is highlighted in the microservice principle of "single responsibility per service" (Blinowski et al., 2022, p. 20359), as to avoid communication between teams and maintain their independence. Even if a business functionality is divided into multiple microservices, the team responsible for said business functionality is supposed to own all of the services realizing it (Newman, 2021, pp. 12–13). In this case, the team is still a stream-aligned team, mitigating the problem of communication for changes that span several services (Newman, 2021, pp. 12–13).

Implementing Conway's law, architectural independence of microservices serves to let teams, organized around business units, "operate as independently from each other as possible" (Haselböck et al., 2017, pp. 1–2) by making the development of individual microservices always "the responsibility of only one team" (Haselböck et al., 2017, pp. 1–2).

Microservices follow the tenet of focusing "on products rather than projects" (Baškarada et al., 2020, p. 2), with cross-functional teams being responsible for their microservices' complete lifecycle.

Independent Deployability

"Each microservice [...] can be deployed and scaled independently as an autonomous component" (Andriyanto et al., 2019, p. 282; Mendonça et al., 2021, p. 17). The concept of being able to change and put microservices into production, irrespective of others, is called independent deployability (Newman, 2021, pp. 6–7)(Newman, 2021, pp. 3–4; Wolff, 2018, p. 2). In microservice architecture, this describes the default release approach rather than a theoretical ability (Newman, 2021, pp. 6–7). This also implies the independent replication of instances, which makes for more flexible scaling on the level of fine-grained microservices (Sellami, Ouni, et al., 2022, pp. 2–3).

Independent deployability is achieved through loose coupling by use of well-defined and stable interfaces/contracts between microservices (Newman, 2021, pp. 6–7). One of the main implementation choices by which microservice architecture loosens coupling and differentiates itself from regular SOA is by isolation of data between services (Newman, 2021, pp. 6–7; Sellami, Ouni, et al., 2022, pp. 2–3). Encapsulating a dedicated database with each microservice makes it easier to recombine microservices in different ways (Newman, 2021, pp. 6–7). The focus on independent deployability requires a desire for stable interfaces between loosely coupled services, which achieves ancillary benefits in defining service boundaries (Newman, 2021, pp. 6–7). Xiao et al. (2016)

require data stewardship of each microservice for all data within its contextual boundary as a prerequisite for independence (p. 62). While the tactics and prioritized aspects in eliminating dependencies diverge, Haselböck et al. (2017) emphasize the overarching goal of "designing microservices as independently from each other as possible" (pp. 1–2) as a unifying concept. Microservice architecture is thus an architectural pattern for distributed applications, comprised of autonomous microservices "that are small application in themselves" (Xiao et al., 2016, p. 62). This brings up one central defining feature of microservice architecture: Each microservice is supposed to be an independent monolith and can consequently be managed, deployed and scaled independently, with individual implementation technologies, release lifecycles and development methodologies for individual microservices (Xiao et al., 2016, p. 62).

Componentization via Services

Another central tenet of microservice architecture is the principle of componentization via services; applications are decomposed into independent services (Baškarada et al., 2020, p. 2). Microservices run in separate processes or independent machines to facilitate their technological independence (Wolff, 2018, pp. 2–3). This necessitates their communication over networks (Baškarada et al., 2020, p. 2; Wolff, 2018, p. 3). Services enable this by encapsulating their functionalities and offering them on one or more service endpoints (Newman, 2021, pp. 3–4). These endpoints are represented by service interfaces, abstracting all implementation details (Blinowski et al., 2022, p. 20359). The exposure of functionalities via interfaces is not unique to microservices; it is rather their principled avoidance of depending on functionalities provided through the interfaces of other services that sets them apart (Sellami, Ouni, et al., 2022, pp. 2–3).

Database Isolation

Maximizing their independence requires a push towards stateless microservices, with each managing "its own layer of persistence" (Di Francesco et al., 2019, p. 78) to avoid sharing resources (Baškarada et al., 2020, p. 2). Because of their autonomy, microservices are self-contained; this extends to all dependencies, meaning the implementation details hidden behind their "clearly defined interfaces" (Haselböck et al., 2017, pp. 1–2) include all their resources (Blinowski et al., 2022, p. 20359; Newman, 2021, pp. 3–4). Thus, all data of a microservice is stored within its own database (Xiao et al., 2016, p. 62).

As microservices do not share databases, a Microservice trying to use data held by another microservice needs to request said data from it (Newman, 2021, pp.

3–4, 7–8). This gives the microservice holding the data the power to decide with which data to respond to the request, allowing for clear separation of internal implementation, which can change freely, and external contracts used by consumers (Newman, 2021, pp. 7–8). In the pursuit of independent deployability, external contracts are supposed to change infrequently to limit the amount of backward-incompatible changes breaking compatibility with upstream consumers (Newman, 2021, pp. 7–8). Cleanly delineating internal implementation and external contracts reduces the need for such (backward-incompatible) changes (Newman, 2021, pp. 8–9).

Evolutionary Design

Thereby, microservices aim to be treatable as a black box, exposing only necessary information via external interfaces while hiding as much information pertaining to their implementation as possible (Newman, 2021, pp. 3–4; Wolff, 2018, pp. 32–33). Their "evolutionary design" (Baškarada et al., 2020, p. 2) enables extensive changeability of implementation details as long as those changes do not affect its exposed interfaces in a backward-incompatible fashion (Newman, 2021, pp. 4–5). In order to ensure backward compatibility, clear separation between parts that are supposed to be easily modifiable and parts that can remain difficult to change, is required (Newman, 2021, pp. 4–5). This creates clear and stable service boundaries that stay unaffected by internal implementation changes and loosen coupling between services while strengthening their internal cohesion (Newman, 2021, pp. 4–5).

Technology Agnosticism

Concealing their entire implementation and data structure enables microservices architecture to maintain agnosticism regarding the specific implementation details of individual microservices (Baškarada et al., 2020, p. 2). It enables heterogeneous technology stacks and data models across microservices of the same system, without restricting them to any particular platform or programming language (Baškarada et al., 2020, p. 2; Wolff, 2018, p. 2).

Network Communication

Microservice-based applications are composed of multiple, collaborative microservices that interact through lightweight communication methods (Andriyanto et al., 2019, p. 282). This way, consumers (e.g. other services) can access proffered functionalities over the network with protocols deemed appropriate to support loose coupling (Newman, 2021, pp. 3–4; Wolff, 2018, pp. 2–3).

Choreography

Microservice interfaces need to be discoverable, so that consumers can find microservices without explicit knowledge of their underlying implementation or physical location (Xiao et al., 2016, p. 62). This approach aligns with the principle of "smart endpoints and dumb pipes" (Baškarada et al., 2020, p. 2), which advocates for the autonomy and decentralization of microservice functions. By eliminating centralized orchestration infrastructure, like the ESB, the responsibility for choreography shifts directly to the individual services themselves. (Waseem et al., 2020, p. 2). This design philosophy underscores the importance of empowering each microservice to manage its interactions independently, enhancing the overall flexibility and scalability of the system (Waseem et al., 2020, p. 2).

Resilience

To increase the independence of microservices, they need to be able to tolerate failures of other services (Baškarada et al., 2020, p. 2; Haselböck et al., 2017, pp. 1–2). Their orchestration via distributed calls makes them prone to failure due to network unavailability or the unreachability services (Wolff, 2018, pp. 32–33). Consequently, this necessitates that consumer microservices implement appropriate error handling mechanisms to maintain resilience and ensure continuity of operations (Wolff, 2018, pp. 32–33).

Performance Overhead

Integrating microservices through network calls introduces significantly higher latency than within-process calls (Wolff, 2018, p. 32). This has a profound impact on their performance and, therefore, is crucial to their design (Wolff, 2018, p. 32). The alignment of microservice architecture with the domain it represents has wide-reaching effects, such as on organizational structure and the refactoring of functionalities beyond service boundaries (Wolff, 2018, pp. 31–33). Their structural independence affects a host of aspects relevant to their organizational suitability, some of which have been described for the purpose of their characterization. Their differences in boundary definitions and situational appropriateness interweave many aspects of their quality. Certain criteria may be unique to the microservice architectural style, while others might remain largely unchanged by this paradigm shift.

3 Research Approach

This chapter outlines the research methodology and discusses the chosen methods and their combination along the steps of the literature review process. The descriptions of the approach are arranged to maintain their conciseness and avoid redundancies: Unless specified differently, all descriptions are equally applicable to both conducted literature reviews encompassed by this study. To account for the differences in the respective processes, these universal descriptions are complemented by individual in-depth explanations, whenever such dissimilarities manifest. Literature reviews provide a comprehensive overview of existing literature and identify gaps in knowledge that this study aims to fill. Therefore, literature reviews form a substantial part of the work at hand. The methodology developed by vom Brocke et al. (2009) has been adopted to guide the literature review process. This chapter will provide an in-depth explanation of this methodology and discuss how it was applied in the context of the current study. The literature review methodology, as depicted in Appendix Figure C.1, is a structured approach intended to facilitate comprehensive and transparent research (p. 8). It serves as a framework for conducting systematic literature reviews in the field of information systems and particularly focuses the literature search process (pp. 5, 8). Its emphasis on the importance of rigor in documenting the literature search process, made it the methodology of choice for this study (p. 8).

3.1 Definition of Review Scope

vom Brocke et al. (2009) propose to draw on the "established taxonomy for literature reviews presented by Cooper" (p. 9) to clearly define "an appropriate scope and flavour of the review" (p. 8). The six constituent characteristics, each containing certain categories, that comprise Cooper's taxonomy, are depicted in Appendix Figure C.2 (p. 9). The systematic literature review examines the existing literature and analyzes it.

Characteristic	Categories			
Focus	Research Outcomes	Research Methods	Theories	Practices or Applications
Goal	Integration		Criticism	Identification of Central Issues
	Generalization / Conflict Resolution	Linguistic Bridge-building		
Perspective	Neutral Representation		Espousal of Position	
Coverage	Exhaustive	Exhaustive with Selective Citation	Representative	Central or Pivotal
Organization	Historical	Conceptual		Methodological
Audience	Specialized Scholars	General Scholars	Practitioners or Policy Makers	General Public

Based on Cooper (1988, p. 109)

Table 3.1: Definition of review scope according to Cooper's taxonomy

The following details the research scope, as defined in Table 3.1.

The work at hand focuses on research outcomes and their application in practice as guidelines on (the use of) different architectural styles "as the material that is of central interest" (Cooper, 1988, p. 108). This research serves the goal of synthesis and integration of findings from various sources by "formulating general statements from multiple specific instances" (Cooper, 1988, p. 108) to provide a comprehensive understanding of the applicability of different architectural styles. The perspective taken in this study does not advocate for the use of a certain architectural style but takes a neutral perspective in an attempt to "ensure that all sides are represented [...] in a manner that reflects their relative prominence in the literature" (Cooper, 1988, p. 110), before "ultimately taking a strong position based on the cumulative evidence" (Cooper, 1988, p. 110). This work does not aim for exhaustive coverage of all information on the topic. It encompasses literature that is representative of many other works in the field and presents a sample that illustrates it (Cooper, 1988, p. 111). Literature is organized conceptually, based on the concepts of Microservices, SOA and Monolithic architectures and the criteria along which they are compared, so that

"works relating to the same abstract ideas appear together" (Cooper, 1988, p. 112). Regarding its audience, this thesis addresses general scholars in the fields of information systems and computer science, aiming to provide results that are of use to those communities.

3.2 Conceptualization of Topic

This second phase involves identifying key concepts, that will be explored in the literature review (vom Brocke et al., 2009, p. 10). Concept mapping represents "a reasonable way for identifying key concepts" (vom Brocke et al., 2009, p. 10) and helps discover related concepts to identify relevant search terms to be used during keyword search. The following concept maps present a visual summary of key ideas and their relationships in an efficient manner. For clarity, each concept is presented in only one grammatical number, the choice of which is guided by the way these concepts are most commonly referred to in the literature. To avoid redundancies incurred through verbification, concepts are presented as nouns, including gerunds. Appendix Figure C.3 presents the concept map for the literature review on the comparative assessment of architectural styles, whereas Appendix Figure C.4 shows the concept map for the literature review on microservice identification.

3.3 Literature Search

This third phase outlines the procedure to find relevant literature and involves "database, keyword, backward, and forward search, as well as an ongoing evaluation of sources" (vom Brocke et al., 2009, p. 10), as illustrated in Appendix Figure C.5.

3.3.1 Journal Search

The search process commences with an identification of relevant journals to ensure a focus on peer-reviewed articles "published in scholarly journals or proceedings of renowned conferences" (vom Brocke et al., 2009, p. 10). Renowned rankings and lists of premier journals were identified and consulted to guarantee for the inclusion of works from highly respected scholarly journals and conferences in the field of information systems. Specifically, all journals named in the Association for Information Systems (AIS)' Senior Scholars' List of Premier Journals (often referred to as "Basket of Eight") were deemed highly relevant and comprehensively covered by the literature review (Association for Information Systems [AIS], 2023). A comprehensive list of encompassed publication venues can be found in Digital Appendix 1: Basket of Eight. Additionally, all journals rated either A+ or A in the German Academic Association for Business

Research (VHB)'s JOURQUAL3 individual ranking for the sub-discipline of Business & Information Systems Engineering also need to be included (Hennig-Thurau & Sattler). These are listed in Digital Appendix 2: JOURQUAL3. After removal of journals, that are also included in the AIS' Senior Scholars' List of Premier Journals, this augments the pool of publications to be searched by the venues depicted in Digital Appendix 3: Journal and Database Search.

Furthermore, the inclusion of prestigious journals from the adjacent fields of computer science and business administration accounts for the complexity and interdisciplinarity of the topic of this study. The analysis of architectural styles and the development of their architectural design straddles not only the field of information systems, but also encompasses elements of computer science and business administration. The inherent interdisciplinarity of the subject is reflected by the inclusion of both computer science journals and business administration journals. Computer Science journals serve to elucidate technical aspects that underpin the choice and realization of software architecture. Concurrently, business administration journals offer insights into the implications of employing different architectural styles. Since architectural styles are not used in a vacuum, but within complex organizational contexts, understanding their practical, strategic, and managerial aspects is equally essential. With the incorporation of journals stemming from all three fields into the review, this study aims for a balanced and comprehensive understanding of the architectural styles, accounting for both their technical complexity and their strategic impact on organizational efficiency and functionality, facilitating the comparative assessment of architectural styles and the classification of microservice architecture development. Lastly, the prioritization of high-ranking scientific journals and conferences (as indicated by the rankings mentioned above) guarantees, that the analysis is underpinned by renowned studies of exceptional quality levels. This focus on high-ranking scientific articles facilitates the incorporation of studies with a high-quality grade.

3.3.2 Database Search

This step is meant to aid in the following identification of proper databases, that provide access to the previously identified journals (vom Brocke et al., 2009, pp. 10–11). This ensures the inclusion of high-quality sources in the review (vom Brocke et al., 2009, p. 11). To execute a systematic and comprehensive literature search, a carefully chosen selection of databases was employed. The chosen databases are IEEE Xplore, ACM Guide to Computing Literature, Association for Information Systems eLibrary (AISeL), EBSCO Business

Source Complete, and ScienceDirect. The databases IEEE Xplore, ACM Guide to Computing Literature, AISeL, and EBSCO Business Source Complete were chosen to provide complete coverage of the journals outlined in the VHB-JOURQUAL3 ranking, listed in Digital Appendix 2: JOURQUAL3. The addition of ScienceDirect allows for a comprehensive coverage of the AIS' Senior Scholars' List of Premier Journals, listed in Digital Appendix 1: Basket of Eight. Moreover, these databases collectively encompass the fields of information systems, computer science and business administration, in alignment with the interdisciplinary nature of this study. As leading sources in the field of computer science, IEEE Xplore and ACM Guide to Computing Literature pro-vide access to a vast array of highly cited journals and proceedings of confer-ences within this field. AISeL, the digital library of the Association for Infor-mation Systems, ensures coverage of the top-rated information systems litera-ture. EBSCO Business Source Complete covers journals and conference pro-ceedings from the field of business administration. Lastly, ScienceDirect pro-vides access to highly regarded journals published by Elsevier, a leading pro-vider of scientific and technical information. This combination of databases ensured an inclusive and comprehensive coverage of the interdisciplinary lit-erature essential for this study. The chosen databases collectively enable a bal-anced understanding of relevant topics, spanning technical complexities and their strategic impact on organizations.

3.3.3 Keyword Search

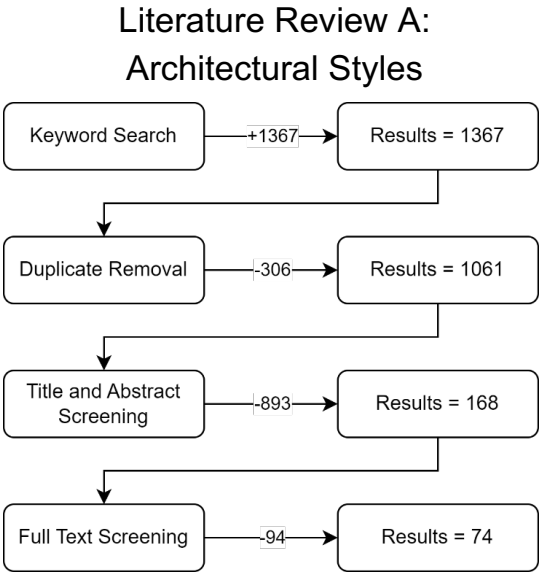


Figure 3.1: Search process of the architectural styles literature review

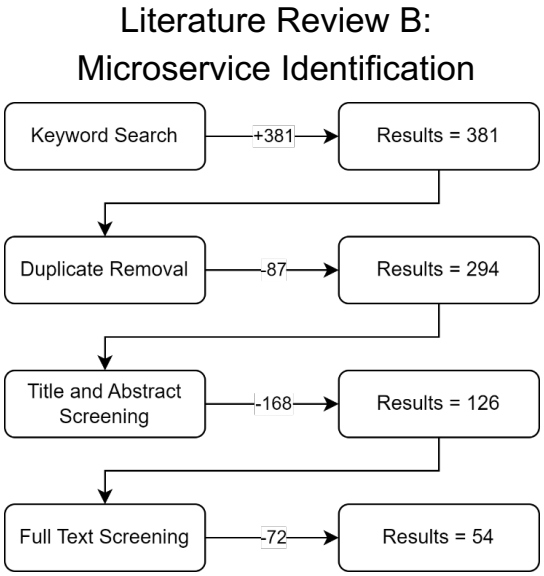


Figure 3.2: Search process of the microservice identification literature review

Figure 3.1 outlines the search process of the first literature review and Figure 3.2 presents an overview of the second literature review's search process. Both are detailed in the following sections. Search strings were formulated and applied in querying the selected databases to exclude irrelevant contributions during keyword search (vom Brocke et al., 2009, p. 11). Synonyms for microservices were chosen in accordance with Di Francesco et al. (2017, p. 22). The search strings employed in this study are presented in their generalized, concise version, illustrating the primary concepts. To ensure transparency and reproducibility, a comprehensive list containing the exact search strings used in both keyword searches can be found in Digital Appendix 4: Full Search Strings. All searches were conducted during August 2023 and were filtered to cover research since the year 2000. This encompasses the rise and evolution of SOA and the shift towards microservices, while still encompassing recent discussions. The research questions of literature review A center on the identification of quality criteria of service-oriented and microservice architecture for their assessment and comparison with each other and monolithic architecture. To answer them, statements relevant to their qualities needed to be extracted. Thus, the study aims to research the distinctive characteristics, strengths, weaknesses, and applicability of each style, to identify the central quality criteria that are most relevant to each as guidelines for their selection. Therefore, search strings were formulated to find studies on their benefits and challenges, their similarities and differences, guidelines on their organizational suitability and

appropriate usage contexts. A concept map of the search terms can be found in Appendix Figure C.6.

Advantages and Disadvantages

```
microservice* OR micro-service* OR SOA OR service-oriented architecture
OR service oriented architecture OR monolith*) AND (advantage* OR dis-
advantage* OR pros OR cons OR strength* OR weak* OR benefi* OR chal-
leng* OR why OR smell* OR evaluat*)
```

Comparison

```
((microservice* OR micro-service*) AND (SOA OR service oriented archi-
tecture OR service-oriented architecture)) OR ((microservice* OR micro-
service*) AND (monolith*)) OR ((SOA OR service oriented architecture OR
service-oriented architecture) AND (monolith*))
```

Guidelines and Suitability

```
(microservice* OR micro-service* OR SOA OR service-oriented architec-
ture OR service oriented architecture OR monolith*) AND (use case OR
use cases OR when to OR guid* OR suitable OR suitablility OR best prac-
tice OR best practices OR choos* OR choice OR (pattern AND select*) OR
appropriat* OR criteri* OR condition* OR perspective OR assessment OR
coping OR cope* OR adopt*)
```

The search strings for the literature review on microservice identification are tailored to the research questions about microservice identification approaches. For a concept map of these search terms, refer to Appendix Figure C.7.

Comprehensive Microservice Identification

```
(microservice* OR micro-service*) AND (identif* OR design OR designing
OR migrat* OR extract* OR decompos* OR evolution OR evolv* OR modulari-
zation OR refactor* OR build OR building OR "monolith first" OR mono-
lith-first OR transform* OR transition OR "breaking down" OR granular-
ity OR synthesizing OR synthesis OR generate)
```

Phrase-Centric Microservice Migration

```
(in)to (a) microservice(s) OR (in)to (a) micro-service(s) OR (in)to (a)
micro service(s)
```

Data Extraction and Compilation

Selected databases were queried using the specified search strings to ensure comprehensive and accurate research material. Search results were extracted, downloaded, and compiled for title and abstract screening. For efficient data analysis, bibliography exports were uniformly integrated into a compatible format. Thus, they were transformed into Comma Separated Values (CSV) files and imported into Microsoft Excel. While IEEE Xplore and EBSCO Business Source Complete provided exports directly in CSV, other databases did not. ACM Guide to Computing Literature and ScienceDirect exports, initially in BibTeX format, were converted to CSV using the official BibTeX to CSV converter tool (Paperpile, 2023). For AISeL exports, available only as text files, a custom Python script, included in Appendix F, was developed to transform

these into CSV format and facilitate their analysis in Excel. To account for the multiple, possibly overlapping search strings applied to each scientific database, exports were then aggregated and deduplicated by their respective databases to obtain the sets of unique results per database, presented in Table 3.2.

Database	Search Results
IEEE Xplore	501
ACM Guide to Computing Literature	328
AISeL	116
EBSCO Business Source Complete	138
ScienceDirect	284

Table 3.2: Search results of the architectural styles literature review per database

The tables of each database were then uniformly formatted and integrated into one table containing 1367 results. For the literature review on microservice identification, EBSCO Business Source Complete and ScienceDirect were found to always ignore stop words, such as *into*, *to*, and *a*. Therefore, phrase-centric microservice identification search string was applied to the results of this search string in the Microsoft Excel tables stemming from the two databases unable to implement it, to allow for the correct application of specified search strings to all results. This was achieved using the following Excel formula, with "D7" representing the column containing the document title:

```
= OR(ISNUMBER(SEARCH("to microservice";D7));
ISNUMBER(SEARCH("to micro service";D7));
ISNUMBER(SEARCH("to a microservice";D7));
ISNUMBER(SEARCH("to a micro-service";D7))
```

This realized the inclusion of the substrings *to microservice*, *to micro-service*, *to a microservice* or *to a micro-service*, were selected, which notably also includes the terms *into*, *microservices* and *micro-services*, ensuring the correct implementation of search strings. Thus, 291 rows were removed from the initial phrase-centric bibliography export from ScienceDirect, leaving eight results. The phrase-centric EBSCO Business Source Complete bibliography export was reduced to 3 results, filtering out 101 rows. Bibliography exports were then aggregated by database, as depicted in Table 3.3.

Database	Reduced Search Results
IEEE Xplore	104
ACM Guide to Computing Literature	202
AISeL	13
EBSCO Business Source Complete	21
ScienceDirect	41

Table 3.3: Search results of the microservice identification literature review per database

Uniformly formatting and integrating the results yielded a total of 381 results.

Duplicate Removal

Results were checked for duplicates. For the architectural styles literature review, 306 duplicate results were identified and removed, making for a total of 1061 studies undergoing title and abstract screening. For the microservice identification literature review, the duplicate removal resulted in a total of 87 publications being marked as duplicates, refining the obtained research to a set of 294 unique results.

Selection Criteria

Upon reviewing unique studies, their "titles, abstracts or even full texts" (vom Brocke et al., 2009, p. 11) are analyzed to evaluate their content and to further limit the amount of relevant literature. Vom Brocke et al. state that "the process of excluding sources (and including respectively) has to be made as transparent as possible" (p. 4). Thus, inclusion and exclusion criteria were explicated and used to assess their relevancy. These are based on the relevance of sources to the research questions, and their academic quality.

Selection criteria	Literature Review A: Architectural Styles	Literature Review B: Microservice Identification
Inclusion	- Studies that focus on the advantages or disadvantages of monolithic, service-oriented or microservice architecture. - Studies that focus on the direct comparison between any of the architectural styles of monolithic, service-oriented or microservice architecture - Studies that discuss the suitability of the considered architectural styles in organizational/practical contexts - Studies that provide/present guidelines on the selection of monolithic, service-oriented or microservice architecture - Peer-reviews studies published in a scientific journal or proceedings of conferences, symposiums, or workshops - Studies that are written in English or German language. - Studies that are accessible electronically	- Studies that focus on the detailed presentation of an approach or method of microservice identification as their primary research topic - Peer-reviews studies published in a scientific journal or proceedings of conferences, symposiums, or workshops - Studies that are written in English or German language. - Studies that are accessible electronically
Exclusion	- Studies that only mention architectural styles in passing without providing information relevant to their assessment - Studies that focus on variations of monolithic, service-oriented or microservice architecture - Studies using homonyms of terms related to monolithic, service-oriented or microservice architecture, referring to different concepts in unrelated contexts	- Studies that only mention microservice architecture in passing and focus on unrelated aspects of software development - Studies starting out with pre-existing microservice architecture without describing their identification - Studies using homonyms of terms related to microservice identification, referring to different concepts in unrelated contexts

Table 3.4: Selection criteria of both literature reviews

Title and Abstract Screening

Selection Criteria, as outlined in Table 3.4, were applied to the pool of results to filter out irrelevant studies. This resulted in the removal of 893 publications during title and abstract screening of the architectural styles literature review, reducing the number of publications moving into full-text screening to 168. For the literature review on microservice identification, the number of 294 publications not marked as duplicates regarding methods for microservice identification were reduced by 168 removed during title and abstract screening to a total of 126 publications for full-text screening.

Full-Text Screening

During Full-text Screening of the first literature review, the research pool was reduced to 74 studies, removing 94 results. 70 results were removed during full-text screening of studies on microservice identification, reducing the number of studies included in the literature review to 56.

Backward and Forward Search

After identification of potentially relevant articles, older sources cited in them were reviewed during backward search and newer literature that has cited them was reviewed during forward search (vom Brocke et al., 2009, p. 11). Based on the results, one round of backward and forward search was performed. The results of the forward backward searches of each literature review are listed in Digital Appendix 5: Forward Backward Searches. Due to the limited scope of this thesis, thereby identified studies were not included in the analysis.

3.4 Literature Analysis and Synthesis

This phase involves the actual review of the literature, where the data is analyzed and synthesized. Vom Brocke et al. only mention this step in passing and suggest the integration of specialized data synthesis methods, such as a concept Matrix according to Webster and Watson (2002) to subdivide relevant concepts into units of analysis to better allow for their arrangement, discussion and synthesis, as exemplified by Appendix Figure C.8 (vom Brocke et al., 2009, p. 11).

3.4.1 Comparative Assessment of Architectural Styles

During the first round of review, statements relevant to concepts defined with the search strings were excerpted. Included studies were then organized according to the architectural styles discussed by them and the quality criteria they emphasize. In case a statement pertains to multiple quality criteria, these were tagged in the statement. Quality criteria also considered by the excerpt were

marked for the study, with a reference to the quality criterion in which the original statement is documented. To identify relevant quality criteria, extracted statements needed to be generalized and categorized. Alternative statements referring to the same concept were therefore iteratively hierarchized and refined to obtain the final set of 20 quality criteria.

To facilitate a structured and visual representation of the literature's coverage on the architectural styles, this study employs a concept matrix, as proposed by Webster & Watson. The concept matrix serves as a tool to map out the thematic presence of architectural styles across the reviewed papers, subdividing the incorporated literature into research domains according to the architectural styles discussed in each. The matrix is structured based on the concept matrix structure depicted in Appendix Figure C.8, with architectural styles and quality criteria as units of analysis and rows representing individual publications reviewed. A binary marking system is used, where a mark in a cell indicates the thematic presence of a particular architectural style or criterion in the corresponding paper. The concept matrix provides a high-level overview of the literature, highlighting gaps and overlaps in the thematic coverage. It also aids in identifying patterns and trends in the research landscape, offering insights into areas that may require further exploration.

Recognized as a dependable method for presenting an aggregated view of various studies, a summary table enhances the conciseness and organization of the literature review. As Younas and Ali (2021) advocate for the importance of these tables, stating they “provide the reader with the information at one glance” (p. 32) and emphasize their “utmost importance” in maintaining clarity and rigor during the review process. Further underscoring their significance, Tranfield et al. (2003) point out the necessity of considering essential details to “construct summary tables” (p. 217) when devising data-extraction methods. Beyond their organizational utility, summary tables also facilitate a methodical representation of each article's core content. In line with Younas and Ali (2021), these tables succinctly deliver a “synopsis of an included article” (p. 32). Acknowledging the specific nature of every review, they further recommend that tables be “tailored to meet the needs of the individuals’ review” (p. 34). In line with this guidance, the summary tables in this study have been meticulously curated to align with its literature reviews’ distinct requirements.

In the analysis of quality criteria within relevant studies, criteria were prioritized based on their significance and frequency of discussion. Criteria emphasized in less than 20% of studies within a respective research domain are deemed insignificant and thus excluded from detailed evaluation. Conversely,

criteria highlighted in more than 50% of the relevant studies are categorized as a domain's most significant and frequently concerned quality criteria. This categorization helps identify the most critical aspects that demand attention and are of substantial interest across various studies, ensuring that the analysis focuses on the most impactful quality factors. The most significant quality criteria for MSA and SOA were derived from the research domains discussing each architectural style in isolation from other considered architectural styles to ensure a clear mapping of sentiments to architectural styles. The research is then segregated into two classes, with one encompassing all studies that (among others) discuss microservices and the second class being made up of the rest of the studies. This allowed for a direct juxtaposition and identification of criteria most significantly differentiating the two classes. To validate the results and to ensure that the inclusion of studies comparing microservices and SOA in the first of the two classes does not falsify results, the first class was then split further into two classes, with one encompassing studies on microservices without SOA and one class including studies that discuss both architectural styles. The results of both classifications were then assessed in comparison.

3.4.2 Microservice Identification

A taxonomy of microservice identification approaches was produced. The taxonomy highlights common characteristics of microservice identification approaches and their differences. It reflects the approaches proposed in the incorporated studies and is used to classify them along the phases of the process leading up to and including microservice identification, yielding an architectural design constituted by microservice candidates. The phases of the process, followed by examined microservice identification approaches, make up the top-level categories of the taxonomy, with the main techniques, activities, and artifacts applied during each phase as sub-categories. Categories and sub-categories were developed to follow the software development process and were continuously refined based on full-text screening of the incorporated studies. Implementation and operation phases are excluded because they are out of scope for answering the research questions because the identification of microservices, while dependent on preceding process steps, is mainly a design topic (Schröer et al., 2020, p. 154).

For a comprehensive synthesis of the underlying literature, the taxonomy was applied to all included studies, and all considered studies were mapped to its concepts. To account for the complexity of certain activities, techniques, or artifacts in certain phases, the literature synthesis matrix enhances the taxonomy with qualifying fields. These additions provide supplementary textual

descriptions, which expand upon the standardized concepts, elucidating the taxonomy's intricacies by offering deeper insights. While the phases of the process are strictly sequential, the activities encompassed by them allow for iteration and parallelization.

The applied augmented taxonomy of microservice identification approaches as the literature synthesis matrix is then used as an analysis tool to produce a framework of approaches to microservice architectural design, including microservice identification. The framework builds on the taxonomy and additionally functions as a guideline for practitioners by differentiating between the main points of departure for the design of microservice architecture.

4 Results: Architectural Styles

4.1 Overview

This subsection presents the profile of incorporated research in terms of date and venue of publication.

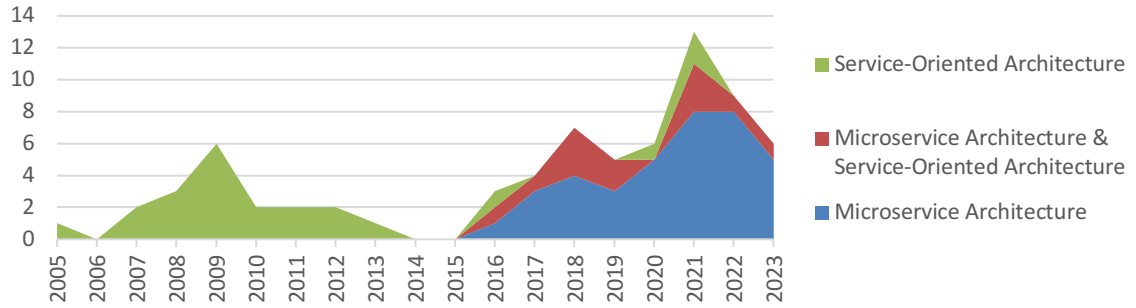


Figure 4.1: Temporal distribution of incorporated research

As depicted in Figure 4.1, studies published on service-oriented architecture show a peak in 2009 with six studies. First incorporated research on microservices and their comparison with service-oriented architecture was published in 2016. Studies on service-oriented architecture reemerged with the inception. From 2016 on, the number of works dedicated to the microservice architectural style has consistently risen. Studies on the comparison of microservices with service-oriented architecture clearly outnumber studies on both topics. Incorporated research on microservice architecture started with one conference paper, followed by an even distribution of publication venues in the subsequent year, including one study resulting from a workshop in 2017. For a complete overview, refer to Appendix Figure D.1. As detailed in Appendix Figure D.2, the scholarly interest in SOA has strongly decreased but matured since its peak in 2009, with all included studies since 2017 being published in journals. In the domain of studies on both SOA and MSA, conference proceedings are the dominant publication venues until 2019. From then on, scientific journals constitute the predominant venue of incorporated literature, reflecting the increased maturation of the topic.

4.2 Identified Quality Criteria

Appendix Table D.1 presents the concept matrix of architectural styles and quality criteria emphasized in their context. Several studies concern more than one of the architectural styles under analysis. The concept Matrix contains a column for each of the three architectural styles service-oriented architecture, microservice architecture, and monolithic architecture. Similarly, studies can discuss

multiple quality criteria for multiple architectural styles. Thus, each quality criterion is represented by a dedicated column in the concept matrix.

Quality Criterion	Alternative Terms and Sub-Criteria
Analyzability	Debugging (Fault localization), Architectural complexity, Understandability, Transparency
Cost	Strategic cost (Economic Impact) Cost Reduction and Savings Potential, Operational Cost, Execution Cost, Infrastructure Cost, Runtime Cost
Development	Required Skills (Learning Curve), Development methods, Development method (Agile development, Devops), Development Process (Time to Market), Team Structure, Autonomy and Coordination of Development, Complexity of the development process, Required effort for development
Deployment	Deployment flexibility, Deployment Constraints
Granularity	Size of services or components, Number of Services or Components, Distribution of service sizes, degree of centralization
Governance	Legal and ethical considerations, Compliance, Formalized Control
IT Business Alignment	Strategic Alignment (Cultural alignment), Business - IT Communication, IT-Business Alignment
Interoperability	Communication (Interface, Direct Communication between nodes, Lightweight Communication Management of Complex Calls), Infrastructure (Virtualization), Choreography and Orchestration, Discoverability (Service Discovery Technologies)
Performance	Capacity (Number of requests supported, System Capacity), Latency, Response time, Resource Utilization (CPU Utilization, Resource Efficiency, Resource Consumption, Resource Savings), Throughput (Transaction Rate)
Reliability	Availability, Stability, Resilience (Fault Tolerance, Recoverability, Robustness, Survivability)
Modifiability	Application agility (Future flexibility of the application), Adaptability, Changeability, Customizability, Expandability (Potential for future expansion in features and capabilities), Extensibility, Evolvability
Modularity	Data Storage (Statelessness; Database Schema Dependency, Data Consistency), Coupling, Cohesion (Variety/variability, similarity), autonomy/independence, Separation of concerns, boundaries between components and services
Maintainability	Testability (Feasibility, Required Workload, Testing Tools), Monitoring (Logging)
Portability	Decomposition, Portability to Cloud and Serverless, Replaceability

Quality Criterion	Alternative Terms and Sub-Criteria
Integration	Integration of third parties, Interoperability between businesses, Integration of external applications
Organization	Business agility (Flexibility to adapt to changing requirements / market changes), Business Process, Business strategy
Security	Attack Surface
Scalability	Elasticity
Technology Heterogeneity	System Heterogeneity
Reusability	Composability

Table 4.1: Identified quality criteria

Table 4.1 shows the final set of identified quality criteria. Each quality criterion is shown with its constituent first-level sub-criteria. Second-level sub-criteria of first-level sub-criteria are included in parentheses behind their respective first-level sub-criteria.

Analyzability

Analyzability of software, in terms of its understandability, is a measure for the difficulty of completely understanding it, depending on the architecture's complexity (Blinowski et al., 2022, p. 20358). This criterion also covers fault localization for debugging, which implies the capacity to locate issues, that need to be resolved (Xin Zhou et al., 2022, p. 6).

Cost

This criterion refers to financial investments and impacts, encompassing, inter alia, the provisioning of infrastructure and monetary expenses related to the execution and operation of software systems, accrued during runtime (Joachim, 2011, p. 5). Sentiments on architectural styles' potentials for cost reduction and savings, and economic considerations, including spendings on personnel, are also observed (G. Liu et al., 2020, p. 634).

Development

Development encompasses the skills, coordination and effort required to implement realize applications in an architectural style, affecting the speed of development and onboarding of new developers (Baškarada et al., 2020, p. 6). It describes the breadth of knowledge that developers need to have to understand the range and consequences of their actions (Baškarada et al., 2020, p. 6). Furthermore, it pertains to an architectural style's aptness to be employed in specialized development methodologies (Baškarada et al., 2020, p. 6). It additionally includes themes regarding the development teams, taking into account the

scope of development, with developers possibly focusing on "small parts of an application" (Baškarada et al., 2020, p. 5) or being required "to reason about complex dependencies" (Baškarada et al., 2020, p. 5). Expecting a perspective akin to the former of the two examples from developers flattens the learning curve, especially for junior team members (Baškarada et al., 2020, p. 5).

Deployment

This criterion considers the simplicity of the deployment process, affected by its continuity and automation (Niño-Martínez et al., 2022, pp. 632, 644). Sentiments of studies discussing, how "straightforward" (Niño-Martínez et al., 2022, p. 632) and frequent the deployment of applications into productive environments is, also fall under this category (Laigner et al., 2021, p. 3352).

Granularity

This criterion refers to the size and distribution of services or components within an architecture, including the number and size of services. It describes the distribution of architectural elements in respect to their relative sizes, determining how fine-grained or coarse-grained the design of services is (Gan et al., 2019, p. 4; Joachim, 2011, p. 5; Sorgalla et al., 2021, p. 3). Finer grain of services describes a larger number of smaller services; coarse-grained services are fewer and larger (Munaf et al., 2019, p. 83).

Governance

This concept examines, how management is centralized or distributed (Munaf et al., 2019, p. 83). Distributed governance involves individual service owners who are responsible for their respective services and the contractual agreements that govern them (Baškarada et al., 2020, pp. 6–7). Additionally, governance encompasses the integration of formalized technological control mechanisms in an organization's IT governance structures, including policy management processes and definition of roles and responsibilities (Hustad & Olsen, 2021, p. 602).

IT Business Alignment

Compatibility of software systems with business objectives falls under the umbrella of this criterion (Becker et al., 2011, p. 208). The structures aligning an organization's IT and business departments are also encompassed within Business-IT Communication (Hustad & Olsen, 2021, p. 601). It describes communication "between the business and the IT arms" (Beydoun et al., 2013, p. 4) with a "common language" (Becker et al., 2011, p. 208), meant to foster a culture of partnership for efficient cooperation in joint projects (Hustad & Olsen,

2021, p. 601). Statements on the fit between technological aspects and the wider organizational strategy, ensuring top management support and compatibility across the organization (Boh & Yellin, 2010, p. 20; Joachim et al., 2009, p. 7). The criterion also involves the technical alignment of business processes through their abstraction via technical interfaces (Andriyanto et al., 2019, p. 286).

Interoperability

This criterion examines the mechanisms and structures that enable effective communication between distributed services, such as middleware (Raj & Ravichandra, 2018, p. 1533). Platforms, that underly applications, streamline the interactions of software agents (Erickson & Siau, 2008, p. 2). The ways, in which system components communicate and exchange data, including the protocols guiding their cooperation, are also included (Mahmood, 2007, p. 499). Communication between distributed services involves remote calls over a network, which connects services via interfaces (Aksakalli et al., 2021, pp. 16–17; Kleftakis et al., 2022, p. 353). This setup can be orchestrated by centralized integration components or components can be empowered to actively choreograph the interplay of functionalities across the system, shifting complexity from the integration level to the component or service level (Blinowski et al., 2022, p. 20358; Cerny et al., 2017, p. 230). In such architectures, each service is responsible for the correctness of its interactions with others, often leading to point-to-point communication setups (Aksakalli et al., 2021, p. 15; Blinowski et al., 2022, p. 20358; Cerny et al., 2017, p. 230). These arrangements have significant implications for the discoverability of available services, affecting their localization and use (Aksakalli et al., 2021, p. 16).

Performance

This criterion covers various aspects of system performance, focusing primarily on throughput, capacity, resource efficiency, latency, and response time. Throughput measurements assess the total number of requests a system can serve, providing a basis for evaluating system capacity and transaction rate (Al-Debagy & Martinek, 2018, p. 154). The Capacity aspect emphasizes the maximum number of requests that can be effectively supported (Al-Debagy & Martinek, 2018, p. 149). Performance extends beyond mere effectiveness to efficiency, involving the meritorious utilization of computing resources to optimize performance (Al-Debagy & Martinek, 2018, p. 152). Statements regarding latency and response time, crucial factors in distributed systems where network overhead can significantly impact performance, are also contained within the criterion (Aggelopoulou & Pramatari, 2009, p. 440; Al-Debagy & Martinek,

2018, p. 149; Oliveira Rosa et al., 2020, p. 2). The time required for system startup and readiness to serve requests is also considered (Weerasinghe & Perera, 2021, p. 143).

Reliability

This criterion encompasses the system's ability to provide predictable reliability across its different areas, ensuring consistent performance under various conditions (Al-Debagy & Martinek, 2018, p. 149; Benavente et al., 2022, p. 183). It also includes the concept of system resiliency, which ensures that the failure of individual components does not compromise the integrity of the entire system, thereby maintaining operational continuity and enhancing survivability (Al-Debagy & Martinek, 2018, pp. 149–150). Fault tolerance is another key aspect, reflecting systems' "resistance to connection losses or failures of service" (Benavente et al., 2022, p. 183), preventing interruptions to their general functionality. This ability to tolerate failures is crucial for maintaining the availability of the services, that are not faulty (Benavente et al., 2022, p. 177). To this end, systems can be designed to accommodate partial failures by isolating faults to limit their "propagation to other building blocks of the architecture" (Laigner et al., 2021, p. 3348).

Modifiability

Modifiability is significantly influenced by the architectural choices that dictate whether modifications require a complete rebuild of the overall application (Andriyanto et al., 2019, p. 282). This criterion also considers the complexity involved in coordinating changes, as these modifications often impact other parts of the system, complicating both the process and the outcomes (Benavente et al., 2022, p. 178). Furthermore, the inherent complexity of applications can decrease their changeability; code of complex applications, can "behave in unexpected ways" (Blinowski et al., 2022, p. 20359) when modified, affecting the ability of the application to evolve and incorporate upgrades to its functionality, thereby influencing future flexibility and expandability (Y. Wang et al., 2021, p. 5).

Modularity

This criterion comprehends several structural aspects of components and services, particularly their boundaries, which facilitate their independence (Al-Debagy & Martinek, 2018, 1, 4). Modularity emphasizes that increasing the cohesion of a module's functionality can significantly reduce the complexity of coupling it with other modules (Zhang & Tanniru, 2005, p. 2268). Similarly, decreasing its need for coupling in terms of inter-service interactions, increases

the module's cohesion (Zhang & Tanniru, 2005, p. 2268). Coupling describes the relationships of services within or across codebases, with the minimization of their coupling directly increasing their independence. (Al-Debagy & Martinek, 2018, p. 149). Code sharing, on the contrary, can violate independence and the closely related concept of behavioral autonomy (Y. Wang et al., 2021, p. 24). Modularity also covers sentiments on the independence of data storage and database schemas (Baškarada et al., 2020, p. 2). In such cases, the absence of a "single source of truth" (Baškarada et al., 2020, p. 2) suggests the potential for data consistency issues, which are also considered under this criterion (Billawa et al., 2022, p. 6).

Maintainability

Maintainability addresses the complexity inherent in managing systems, including considerations of feasibility and requirements for testing, which can accrue technical debt over time (Christoforou et al., 2022, p. 15). This criterion also encompasses standardized means and options for monitoring, such as logging, profiling, and health management, which are essential for effective system maintenance (Di Francesco et al., 2019, p. 87; Niño-Martínez et al., 2022, p. 638).

Portability

This criterion encompasses several aspects, including replaceability, which describes the ease of replacing services and components "with alternative implementations" (Sorgalla et al., 2021, p. 3) "without impacting other services" (Raj & Ravichandra, 2018, p. 1533). Additionally, "the extent to which an organization's IT infrastructure resources" (Hustad & Olsen, 2021, p. 600) are can be seamlessly transferred to different environments plays a crucial role in their flexibility. Furthermore, portability involves the decomposition of systems during their migration to another architectural style or cloud platform, ensuring that the system remains functional in its new environment (Camilli et al., 2023, p. 1; Xin Zhou et al., 2023, p. 16).

Integration

This criterion focuses on the integration of third parties, external applications and services into the existing architecture (Joachim, 2011, p. 7). Realizing the integration of "features created by third parties" involves interoperability between organizations and avoids reimplementation of existing functionality (Benavente et al., 2022, p. 179). However, the efficiency and effectiveness of integrated components and systems depends on the compatibility of systems across

various platforms, possibly allowing for the integration of legacy systems into new applications (Baškarada et al., 2020, p. 6; Joachim, 2011, p. 5).

Organization

This quality criterion covers topics surrounding the suitability of architectural styles for scaling organizations along with their business processes and technical architecture to facilitate their growth (Blinowski et al., 2022, p. 20358). This criterion also addresses business agility, which is defined as an organization's ability to strategically respond to changing business needs (Oliveira Rosa et al., 2020, p. 2).

Security

This quality criterion revolves around the complexity of implementing reliable security mechanisms across various services (Oliveira Rosa et al., 2020, p. 2). This includes managing a system's "exposed area" (Benavente et al., 2022, p. 179), which represents the potential attack surface of the system, with its "multiplicity of attack points" (Billawa et al., 2022, p. 6), that could potentially be exploited. Furthermore, it encompasses the identification and mitigation of security risks and code vulnerabilities, along with the enforcement of security measures necessary for privacy protection (Trihinas et al., 2018, p. 67).

Scalability

This criterion describes a system's mechanisms to manage an increasing workload through the replication of services or components based on demand (Sorgalla et al., 2021, p. 3). Known as elasticity, these adjustments help fluctuations in user demand (Papakonstantinou et al., 2020, p. 2; Sorgalla et al., 2021, p. 3; Trihinas et al., 2018, p. 67). Additionally, scalability involves duplicating the entire system or parts of it when reaching capacity limits to ensure continuous efficient operation (Baškarada et al., 2020, p. 2; Götz et al., 2018, p. 168). Horizontal scalability pertains to scaling by duplicating only the overwhelmed parts of the system (Blinowski et al., 2022, p. 20372; Götz et al., 2018, p. 168; Trihinas et al., 2018, p. 67). When scaled vertically, an additional complete application instance is brought up to serve requests instead of the more technically complex but precise horizontal scaling of system components or services (Auer et al., 2021, p. 4).

Technology Heterogeneity

Munaf et al. (2019) characterize technology heterogeneity as "an opportunity to use the technology of own choice for implementation" (p. 83) of software components. This criterion describes the support for using different technologies

for different services within the same system, allowing for the use of heterogeneous technologies across services (Al-Debagy & Martinek, 2018, p. 149; Niño-Martínez et al., 2022, p. 632). It describes the ability to choose the best-suited technology stack for each service, without restricting the system to any particular technologies, which results in the heterogeneity of components (Baškarada et al., 2020, p. 5).

Reusability

Reuse of services or components across different applications, without changing their functionality is the focus on this criterion, which also considers implications of repurposing assets in various configurations or versions of applications (Hustad & Olsen, 2021, pp. 600–601; Ilk et al., 2010, p. 4). Additionally, the composability of these components or services is essential; it refers to their ability to be easily combined to form hierarchical composite services (Reddy et al., 2009, p. 59). Reusability also applies to entire legacy applications, enabling their incorporation into new uses without significant reengineering, thus preserving investment in existing technologies (Aggelopoulou & Pramatari, 2009, p. 442).

4.3 Most Emphasized Quality Criteria

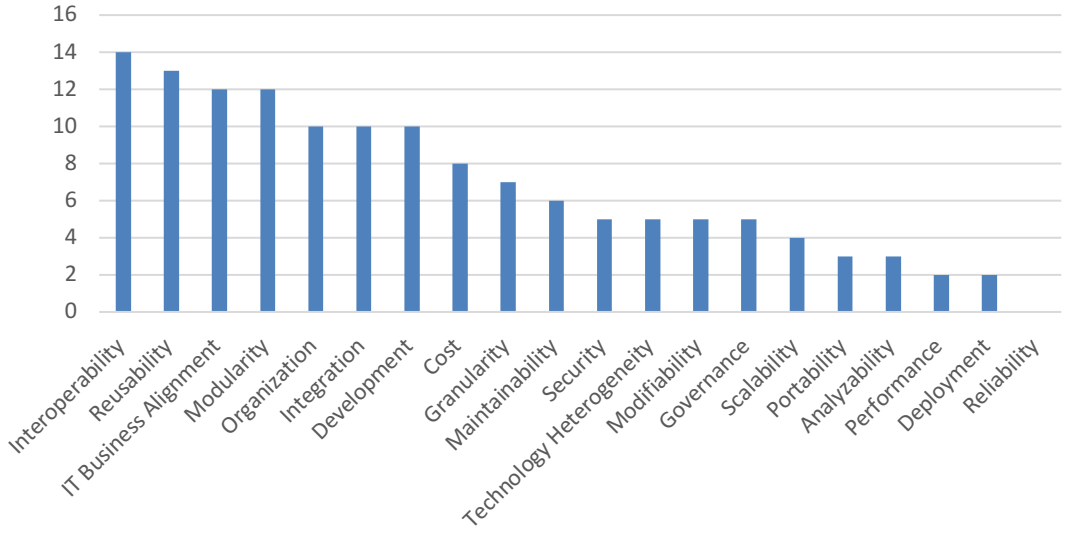


Figure 4.2: Observation frequency of quality criteria in studies on SOA

Figure 4.2 presents the frequency count of quality criteria discussed in 19 studies that specifically pertain to service-oriented architecture. Reliability is not discussed in any of the studies. Further applying the 20% cutoff, performance and deployment, each mentioned in only two studies, along with portability and analyzability, each cited in three studies, are classified as insignificant for this analysis. The most prominent quality criterion is Interoperability, referenced in

14 out of the 19 studies, demonstrating its significance in SOA discussions. Reusability follows closely, appearing in 13 studies. it business alignment and modularity are tied as the third most discussed criteria, each featured in twelve studies. Organization, integration, and development are identified as the fourth most frequently mentioned, each occurring in ten studies. Less commonly cited but still significant are cost, granularity, and maintainability, with respective mentions in eight, seven, and six studies. security, technology heterogeneity, modifiability, and governance, each discussed in approximately one-third of the studies, indicate moderate significance. Scalability, mentioned in four studies, is deemed a relevant quality criterion within SOA but is the least frequently observed among the significant criteria.

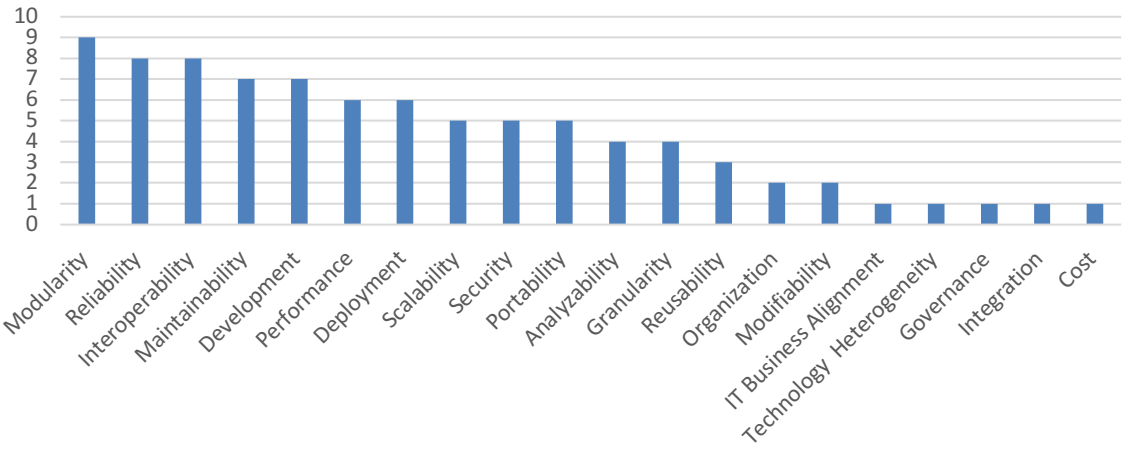


Figure 4.3: Observation frequency of quality criteria in studies on MSA

Figure 4.3 provides the frequency of quality criteria mentions in twelve studies related to only microservice architecture. Applying the 20% cutoff, it business alignment, technology heterogeneity, governance, integration, and cost are each discussed in one study and therefore removed from this sample. it business alignment and technology heterogeneity are mentioned twice, also categorizing them as insignificant according to the predefined threshold. Conversely, modularity stands out as the most significant quality criterion, emphasized in nine out of twelve papers. Reliability and Interoperability are also prominent, being highlighted in eight papers each. Maintainability and development are noted in seven papers, securing their positions among the most significant criteria. Criteria such as performance, deployment, scalability, security, portability, analyzability, granularity, and reusability are mentioned less frequently but still merit attention in the context of microservice architecture. According to this analysis, there is no overlap in the sets of insignificant criteria between the different architectural styles studied, thus all identified criteria are included in their comparative analysis.

4.4 Comparison

4.4.1 Overview

Domain	Number of studies	Most frequently observed Quality Criteria	
		Frequency of Observation	Quality Criterion
Comparison of Microservices and Monoliths	25	22	Performance
		19	Scalability
		17	Modularity, Development
		16	Reliability, Deployment
		15	Interoperability
		14	Portability
Service-Oriented Architecture	19	14	Interoperability
		13	Reusability
		12	IT Business Alignment, Modula-
		10	Organization, Integration, Deve-
Microservices	12	9	Modularity
		8	Reliability, Interoperability
		7	Maintainability, Development
Comparison of Microservices, Service-Oriented Architecture and Monoliths	9	7	Modularity, Performance, Scala-
		6	Interoperability, Development, Granularity, Deployment, Modifi-
		5	Reliability, Maintainability, Portability, Technology Heterogeneity,
Comparison of Service-Oriented Architecture and Monoliths	4	3	Modularity, Interoperability
Comparison of Microservices and Service-Oriented Architecture	3	3	Reusability, Modularity
		2	Scalability, Portability, Interoperability, Granularity, Develop-

Table 4.2: Research domains and their most significant quality criteria

Table 4.2 depicts incorporated research, organized along their distinct research domains. These domains are defined by the architectural styles discussed in their constituent studies. For each domain, the quality criteria observed in more than 50% of studies are presented with their frequency of observation. For an overview across all domains, refer to Appendix Figure D.4. Details regarding the observation frequencies in the domain of comparison of microservices and monoliths are depicted in Appendix Figure D.6. For an analogous perspective on studies comparing SOA and monoliths, refer to Appendix Figure D.5. Similarly, the observation frequencies of quality criteria for the domain of comparisons between microservices and SOA, see Appendix Figure D.7. Finally, the observation frequency of quality criteria in studies discussing microservices

together with both monolithic architecture and SOA is presented in Appendix Figure D.8.

4.4.2 Comparison of Architectural Styles

As highlighted in Appendix Figure D.9, microservices are primarily studied in comparison with other architectural styles, with 75.5% of studies contrasting them with monolithic architecture and/or SOA. Conversely, SOA is more often considered in isolation, with 54.3% of the studies concerning it mentioning neither monolithic nor microservice architecture.

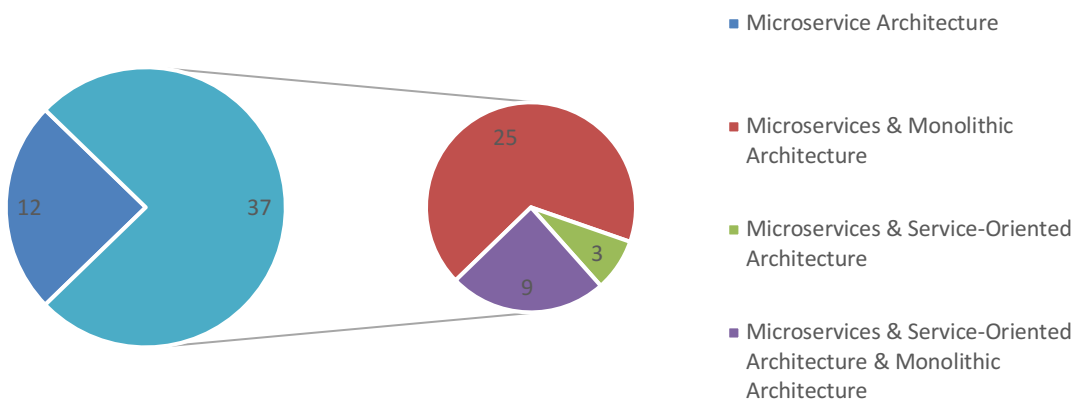


Figure 4.4: Proportion of studies focusing exclusively on MSA and comparisons

Because microservices are being predominantly discussed in comparison to other architectural styles, as Figure 4.4 confirms, the distribution of domains, including this architectural style, are examined in more detail. Of the 37 comparative studies, 25 compare microservices with monolithic architecture, which is the largest subset, indicating a significant interest in contrasting these two styles. Additionally, nine studies explore microservices alongside both SOA and monolithic architecture, showing an integrated approach to understanding these architectural paradigms. A smaller segment of the literature, with only three studies, compares MSA and SOA exclusively, highlighting a reduced targeted interest in the relationship between these two styles without the consideration of monolithic architecture.

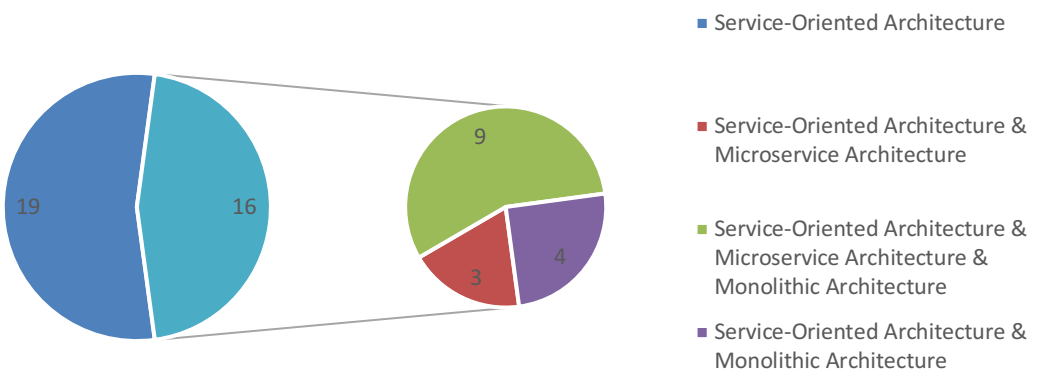


Figure 4.5: Proportion of studies focusing exclusively on SOA and comparisons

Figure 4.5 demonstrates the focus on monolithic architecture in comparative studies. Out of 41 studies, 38 engage in a comparison with monolithic architecture, underlining its continued relevance as a baseline for architectural analysis. Within this subset, the majority (34 out of 38 studies) compare monolithic architecture with microservice architecture, signifying keen academic interest in contrasting the older monolithic paradigm with the more contemporary microservices approach. Additionally, 13 of the 38 studies include SOA in their comparison, suggesting a comprehensive exploration of architectural evolution. This convergence in research highlights the critical examination of monolithic architecture in the context of emerging architectural styles. Appendix Figure D.10 demonstrates the focus on monolithic architecture in comparative studies. Out of 41 studies, 38 engage in a comparison with monolithic architecture, underlining its continued relevance as a baseline for architectural analysis. Within this subset, the majority (34 out of 38 studies) compare monolithic architecture with MSA, signifying keen academic interest in contrasting the older monolithic paradigm with the more contemporary microservices approach. Additionally, 13 of the 38 studies include SOA in their comparison, suggesting a comprehensive exploration of architectural evolution. This convergence in research highlights the critical examination of monolithic architecture in the context of emerging architectural styles.

4.4.3 Comparison by Quality Criteria

4.4.3.1 Differentiating Quality Criteria

The following shows an analysis of the identified quality criteria, with the incorporated research being structured into two classes. The first class is constituted by all research, including microservices. The second class encompasses all other research, which always concerns service-oriented architecture,

specifically the service-oriented architecture domain and the domain of comparison of service-oriented architecture and monoliths, as depicted in Table 4.2. The following tables present the quality criteria, that are observed with at least 20% difference in relative frequency across the two classes as the most significant differentiators for service-oriented architecture and microservice architecture.

Quality Criterion	Microservice Architecture	Service-Oriented Architecture	Difference
Performance	0.73	0.13	0.60
Reliability	0.59	0.04	0.55
Deployment	0.57	0.09	0.48
Scalability	0.67	0.22	0.46
Portability	0.53	0.13	0.40
Analyzability	0.39	0.13	0.26

Table 4.3: Significant differentiating quality criteria that are primarily concerned in MSA research

Quality Criterion	Service-Oriented Architecture	Microservice Architecture	Difference
Reusability	0.61	0.16	0.45
IT Business Alignment	0.61	0.18	0.43
Integration	0.48	0.12	0.36
Organization	0.48	0.24	0.24
Governance	0.30	0.08	0.22

Table 4.4: Significant differentiating quality criteria that are primarily concerned in SOA research

4.4.3.2 Differences

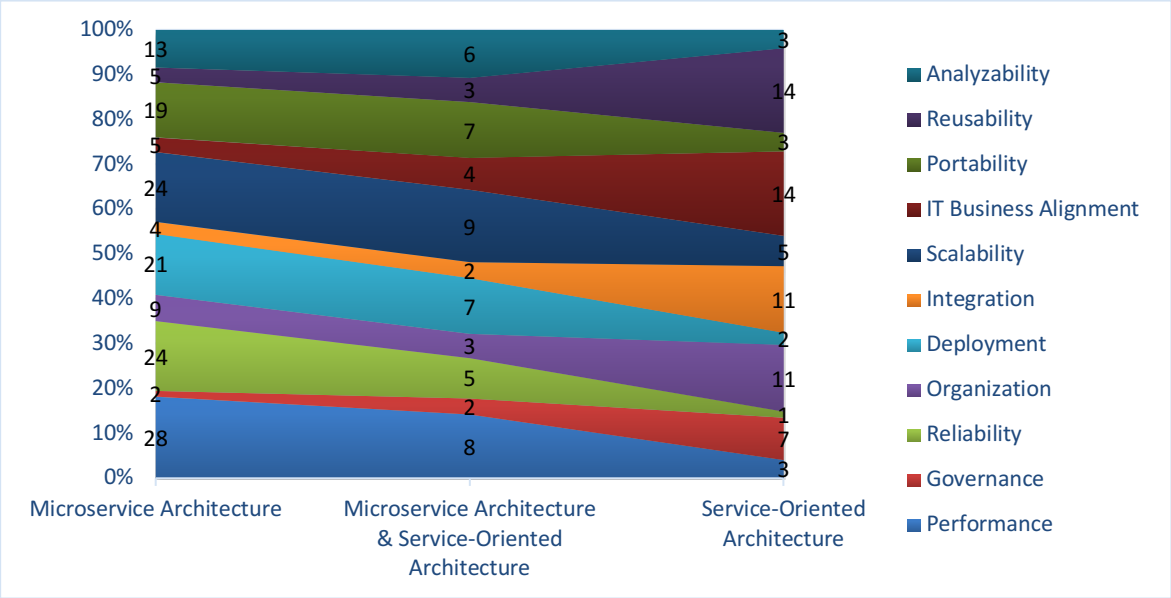


Figure 4.6: Distribution of differentiating quality criteria

Figure 4.6 expands upon the criteria with at least 20% difference in observation frequency, identified in the previous, more coarse-grained juxtaposition of the two architectural styles. It differentiates more precisely between the research domains by dividing the class of studies on microservice architecture into two categories, with the first category pertaining to studies on microservices without mention of service-oriented architecture, and the second group concerning studies that discuss both microservice architecture and service-oriented architecture. Thus, the two architectural styles can be compared more precisely, and the results of their previous juxtaposition can be validated. The category of service-oriented architecture remains unchanged, absolute values pertaining to it are therefore only stated once. The results of these comparisons are analyzed in the following.

Performance

Examining Table 4.3, performance is the most significant differentiating factor between MSA (36/49) and SOA (3/23), with a 60% higher frequency of observation in studies on MSA, than on SOA. Moving to the analysis of Figure 4.6, performance is of no significant concern to the academic field of SOA. A growing trend to focus on this aspect is observed in studies comparing at least MSA and SOA (8/12), culminating in 28 out of 37 MSA studies concerning the performance of microservices, making it the primary focus of this category of research. Notably, performance is considered insignificant as a quality criterion for the assessment of SOA. It is however one of the most frequently concerned quality criteria in the research domain of comparison between MSA and monolithic architecture and in the domain of comparative research on microservice architecture with monolithic and service-oriented architecture. Focusing on performance in the domain of microservices, Benavente et al. (2022) measure lower response times for the microservice equivalent of a monolithic application, with the performance advantage of microservices increasing at a higher number of requests (p. 181). This shows the potential performance optimization achieved in microservice-based systems. Albeit, Oliveira Rosa et al. (2020) emphasize, that inter-service communication makes it is more complex to achieve high performance in microservices, potentially making monoliths have better performance in smaller applications (p. 2). Villamizar et al. (2017) concur, stating the potential increase in response time of microservice applications running in cloud environments, as compared to monolithic systems (p. 243). This is also thematized by Aggelopoulou and Pramatari (2009) in the domain of SOA with an emphasis on the communication overhead of standard messaging formats and intermediaries (p. 440).

Reliability

This criterion is notably not identified at all in the 19 studies concerning service-oriented architecture only in isolation. In the wider categorization applied for the analysis in Table 4.3, which expands the number of studies under the umbrella of SOA to 23, it is observed once, owing to a single study on the comparison of SOA to monolithic architecture by Mahmood (2007), stating the possible of a single well-used service in a service-based application potentially taking out the entire system (p. 499). On the contrary, it is observed in 23 out of 49 studies discussing microservice architecture or comparisons thereof. The focus on reliability as a differentiating factor is signified by its discussion in five out of twelve studies on both architectural styles. Similarly, the research domains of Microservices and the comparison of microservices with monoliths are the only

domains, where this criterion is among the most concerned criteria. The focus on MSA reliability research is due to the complexity of maintaining it across all services; Xin Zhou et al. (2023) discuss the decrease in availability of services as a result of suboptimal service cuts lengthening call chains and increasing the likelihood of failure (p. 5).

Deployment

This criterion under comparison in Table 4.3 is observed in significantly increased frequency among the class of studies including microservices as compared to the remaining studies on SOA. Expanding upon this comparison in Figure 4.6 does not significantly affect the relative frequency of deployment in MSA research (21/37) and consequently highlights its similar frequency among studies on both SOA and MSA (7/12). Al-Debagy and Martinek (2018) highlight microservices' independent deployability as compared to monolithic deployment, which can cause stability challenges in deployment coordination (p. 1). The insignificance of deployment as a criterion for the assessment of SOA is also of note. Cerny et al. (2017) count bringing services into production independency as one of MSA's main differences to SOA (p. 230). SOA can typically be seen as a monolithic system from a deployment perspective (Cerny et al., 2017, p. 233; Munaf et al., 2019, p. 83)

Scalability

Going by the comparison in Table 4.3, scalability is observed at significantly higher frequency among MSA studies, than in the other studies on SOA. Expanding upon the comparison in Figure 4.6 minutely alters the view, as nine out of twelve studies comparing SOA and MS discuss scalability, making it the primary quality criterion concerned out of the differentiating criteria in this domain. scalability is accordingly identified as one of the most relevant quality criteria in three domains shown in Table 4.2. These are: MSA with monolithic architecture, MSA with SOA and monolithic architecture, and MSA and SOA. The need to completely duplicate monolithic applications in order to scale them leads to inefficient hardware usage and resource wastage (Al-Debagy & Martinek, 2018, p. 149; Götz et al., 2018, p. 168; Papakonstantinou et al., 2020, p. 1). Scaling of monoliths can be conducted automatically by a continuous deployment system but is inflexible due to their lacking horizontal scalability (Götz et al., 2018, p. 168). Contrastingly, individual microservices can be scaled independently, which increases the ability of the system to adapt to capacity peaks by scaling only the bottlenecked services that require it and reduces resource wastage at scale through their elasticity capabilities (Al-Debagy & Martinek, 2018, 1,4; Benavente et al., 2022, p. 178; Oliveira Rosa et al., 2020, p. 2;

Papakonstantinou et al., 2020, p. 2). “This makes the reaction to demand peaks faster and can handle the demands more precisely” (Götz et al., 2018, p. 169). Scaling SOA-based systems is more problematic, especially regarding the scalability of centralized components like the ESB (Weerasinghe & Perera, 2021, p. 140). Thus, Benavente et al. (2022) argue, that ”an architecture based on microservices presents a better option on response times in conditions where the number of requests has a high volume” (p. 183). In the same vein, Villamizar et al. (2017) suggest, that ”microservice architectures should be employed in businesses that need to scale their applications to hundreds of thousands or millions of users“ (p. 245). Gravanis et al. (2021) state that systems “with high and/or dynamic scalability requirements” (p. 42) likely significantly benefit from MSA, whereas it likely introduces unnecessary network overhead “for systems with low traffic requirements” (p. 42). Consequently, the benefits of microservices at scale may be outweighed by their challenges in “small scale systems” (Blinowski et al., 2022, p. 20358).

Portability

Portability is identified as a clear differentiation criterion in the binary classification and comparison of studies, with 26 of 49 MSA studies concerning it, in contrast to only three out of 23 SOA studies. This result still holds in Figure 4.6, additionally highlighting portability as a focus of comparative research on the two architectural styles (7/12). Notably, portability is also not included in the set of significant quality criteria for the assessment of SOA, but is one of the most concerned quality criteria of studies in all research domains that pertain to comparisons of microservices with other architectural styles. Sorgalla et al. (2021) emphasize that microservices are “seamlessly replaceable with alternative implementations” (p. 3), whereas the reduced focus on independence of SOA makes it “difficult to replace the service without impacting other services” (Raj & Ravichandra, 2018, p. 1533).

Analyzability

Analyzability research is significantly biased towards the category of microservices (23/49), as defined for the binary comparison in Table 4.3; only three out of 23 of the remaining SOA studies concern analyzability. This is in accordance with its classification as an insignificant quality criterion for SOA assessment. The more specific comparison depicted in Figure 4.6 does not significantly alter this finding (13/37) and additionally shows analyzability to be a significant subject of the incorporated research discussing both SOA and MSA, regardless of the inclusion or exclusion of monolithic architecture (6/12). The

difficulty of understanding MSA holistically leads to high time when trying to locate issues (Blinowski et al., 2022, p. 20358; Xin Zhou et al., 2022, p. 6).

Reusability

As shown in Table 4.4, reusability is focused in significantly divergent frequency across the two categories, with only eight of the 49 studies classified as MSA research, but 14 out of the remaining 23 SOA studies highlighting this criterion, which is also identified as the second-most concerned quality criterion in the domain of SOA and additionally included in the set of most concerned quality criteria in the domain of microservices & SOA. The expanded comparison in Figure 4.6 does not significantly alter this general result but highlights the increased relative volume of studies comparing SOA and MSA, that discuss reusability (3/12) than in the category of MSA (5/37). Encapsulating the functionality of old systems in different services with interfaces allows for its reuse and avoid creating new applications from scratch (Hustad & Olsen, 2021, p. 600). Ilk et al. (2010) highlight, that reusability in SOA reduced the complexity of the overall system (p. 4). SOA promotes parallel reuse of services (Becker et al., 2011, p. 205). Joachim et al. (2009) discuss the trade-off between a service's "reusability and the expenditure necessary for its realization" (pp. 12–13). Beydoun et al. (2013) elaborate on the tendency of "developers to opt out of reuse in favour of creating new software components" (p. 3), which hampers achieving a critical mass of reuse required to justify the efforts of making functionalities reusable.

IT Business Alignment

The results of the comparison presented in Table 4.4 show a significant bias of research emphasizing it business alignment towards the category of SOA (14/23) as compared to MSA (9/49). Expanding on the category of MSA in Figure 4.6 yields a significant shift in the distribution towards comparative studies on MSA and SOA (4/12) and a reduced share remaining in the category of MSA (5/37). Notably, it business alignment is classified as an insignificant assessment criterion in the domain of MSA but belongs to the group of most concerned SOA criteria (see Table 4.2). organizations need to analyze the appropriateness of SOA for their particular, organization-specific setting, and ensure, that development is driven by business needs (Joachim et al., 2009, p. 16; Y. Wang et al., 2021, p. 22). Beydoun et al. (2013) further stress the importance of "proper communication between the business and the IT arms of an organization" (p. 4) for successful SOA adoption.

Integration

The Integration Criterion is observed in significantly lower relative frequency among MS studies (6/49) than in its opposition (11/23) in the binary classification and comparison of incorporated research. This holds in the more precise threefold classification depicted in Figure 4.6. Notably, it is not identified as a significant quality criterion for the assessment of microservice architecture but represents one of the most significant quality criteria in the specific domain of service-oriented architecture, as shown in Table 4.2. Integration in SOA often involves legacy systems that may operate on a variety of platforms, providing a compatibility advantage (Joachim, 2011, p. 5). Baškarada et al. (2020) emphasize SOA's advantageous capabilities in integrating with monolithic systems and commercial off-the-shelf software over microservices, which "imply decomposition of monoliths" (p. 6). Zhang and Tanniru (2005) contrastingly emphasize, that "in an inter-organizational context, [the] more inter-organizational interactions a process has, the less cohesive it is" (p. 2268), highlighting the tradeoff between microservices' independence through decoupling and SOA's integration strengths. Accordingly, Jamshidi et al. (2018) observe, that "SOA is viewed mostly as an integration solution, whereas microservices are typically applied to build individual software applications" (p. 25).

Organization

The is criterion, as shown in Table 4.4 is observed at significantly higher frequency in the SOA category (11/23), than in microservice studies (12/49). The threefold comparison in Figure 4.6 exhibits the same relative frequency across the separated categories of microservices (9/37) and microservices in comparison with SOA (3/12). Notably, this criterion is included in the set of most concerned quality criteria of research in the domain of SOA and consequently identified as a significant criterion for its assessment (see Table 4.2) Andriyanto et al. (2019) characterize the promotion of organizational agility as one of SOA's main benefits (p. 286). Taking a different perspective on organizational aspects, Blinowski et al. (2022) discuss the move to microservices as a response to growth pressures on the business and the need of an accompanying technology making microservices "a compelling solution [...] despite the added complexity" (p. 20358).

Governance

Governance is observed significantly more often in the category of research not including microservices (7/23) than in its opposing category (2/49), as presented in Table 4.4, while also being an insignificant criterion in the assessment

of MSA. The expanded comparison depicted in Figure 4.6 slightly shifts the distribution of the MSA category towards comparative studies on SOA and MSA (2/12) as compared to the 2 out of 37 studies on MSA without considering SOA. Notably, it is identified as one of the most impactful quality criteria for any of the research domains. Establishing effective SOA governance is necessary for its successful adoption, but requires new competencies, structures, and processes (Hustad & Olsen, 2021, p. 602). Combining SOA with cloud services further complicates its governance, which can be resource-demanding to handle (Hustad & Olsen, 2021, p. 602).

4.4.3.3 Similarities

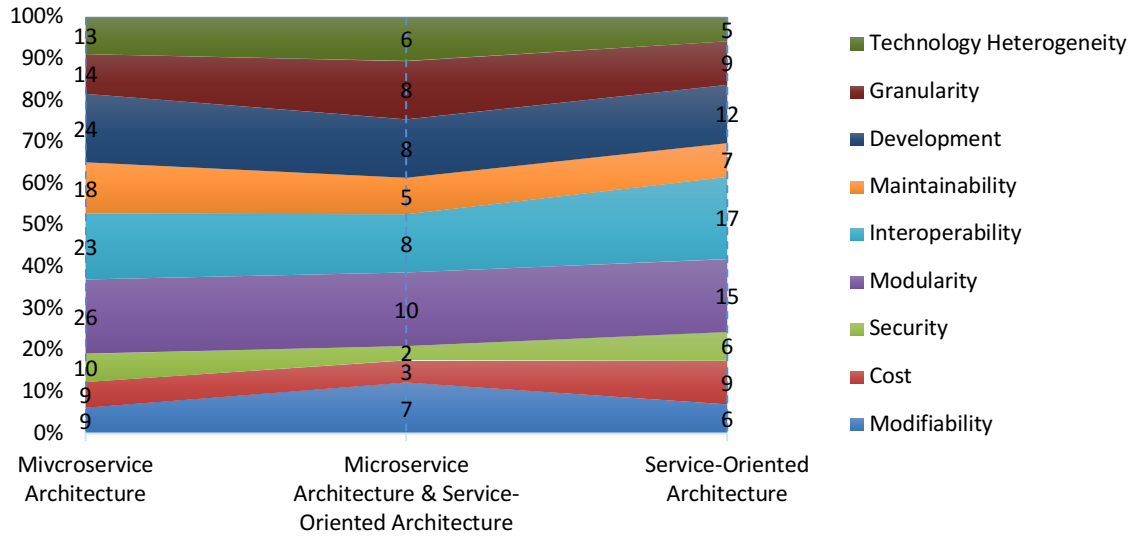


Figure 4.7: Distribution of quality criteria observed in similar frequency across SOA and MSA

Figure 4.7 presents the quality criteria not identified as significant differentiators between the two architectural styles. These are examined more closely in the following. The criteria are divided into three categories, akin to the categorization conducted in Figure 4.6, differentiating more precisely between the research domains by further dividing the class of studies on microservice architecture into two groups. The first category, pertaining to studies on microservices in isolation of service-oriented architecture, and the second group concerning studies that discuss both MSA, and SOA.

Technology Heterogeneity

This is not one of the most frequently concerned quality criteria in any of the different domains and is classified as insignificant in the domain of studies exclusively on microservices. In the Juxtaposition of all studies including microservices and the rest of the studies, there is no significant difference between studies including microservices (19/49) and the rest of the incorporated

research on SOA (5/23). Such difference does not manifest when examining the body of research more closely. Technology heterogeneity is most relevant in the domain of comparisons between microservices, SOA, and monoliths. “Existing monoliths are generally based on relatively fixed technology stacks, some of which may be relatively old” (Baškarada et al., 2020, p. 5). In contrast, microservice architecture gives the “possibility for heterogeneous technology stacks”, which enables experimentation and innovation (Baškarada et al., 2020, p. 5). This flexibility improves developer productivity by “giving developers the freedom to choose their own tools” (Baškarada et al., 2020, p. 5), since individual developers are more productive in particular technologies. (Munaf et al., 2019, p. 83) delineate these architectural styles along this criterion, stating “SOA has more homogenous technology than microservices” (Munaf et al., 2019, p. 83).

Granularity

According to the above juxtaposition of the studies including microservices (22/49), and those that do not (9/23), this criterion is concerned in similar frequency. The more specific examination shows a slight tendency of increased emphasis on granularity in studies considering both microservices and SOA, than in studies on only one of the architectural styles. It is among the most frequently observed quality criteria in the domains of studies on Service-Oriented architecture in combination with microservices, in studies on SOA and monolithic architecture and in studies on all three patterns. Fine-grained services in SOA are possible (Xiao et al., 2016, p. 64). But since SOA services reflect the complexity of the business operations they realize, the granularity of services corresponds to the complexity of the business it represents, making granularity an abstract perspective on service-based systems (Xiao et al., 2016, p. 64). With their inflexible boundaries, inappropriate granularity levels in microservices can result in many consecutive problems, making proper boundary definition a critical concern in MSA (Xin Zhou et al., 2023, p. 16).

Maintainability

Another quality criterion observed in similar frequency in the above juxtaposition of studies including microservices (23/49) and the rest of included research on SOA, (7/23), is maintainability. It is among the most frequently observed criteria in the domain of studies, that consider microservices, monolithic, and service-oriented architecture. It becomes more difficult to setup tests and trace their execution when system elements are distributed, possibly running on different platforms (Aggelopoulou & Pramatari, 2009, p. 439). Thus, Camilli et al. (2023) describe maintenance, and testing of microservices as “painful activities” (p. 41).

Security

Security is also concerned in similar frequency, judging by the binary classification and juxtaposition of included research, with 12 of 49 studies in the microservice category and 6 of 23 studies in the SOA category emphasizing it. However, it is not identified as one of the most concerned quality criteria in any on the research domains laid out in Table 4.2. Figure 4.6 depicts a decreased frequency of observation in research comparing microservices and SOA, emphasizing a reduced focus on security, when comparing the two architectural styles. In the context of microservices, network communication between services "raises significant security risks" (Trihinas et al., 2018, p. 67), potentially leading to code vulnerabilities and privacy leaks, that are prevalent in monolithic architectures. According to Oliveira Rosa et al. (2020), the implementation of reliable security mechanisms across multiple microservices can be complex (p. 2), since MSA "requires dealing at scale with the security of numerous self-contained services" (Trihinas et al., 2018, p. 67). The "multiplicity of attack points available in microservice architecture" (Billawa et al., 2022, p. 6) and their distributed, evolving nature makes it difficult to achieve security "through the traditional/monolithic methods" (Billawa et al., 2022, p. 6). Microservices' database isolation also makes them prone to SQL injection attacks (Billawa et al., 2022, p. 6). Contrarily, Yarygina and Bagge (2018) argue that smaller codebases result in a smaller attack surface, offering some security advantages (p. 15).

Cost

Examining the binary classified juxtaposition uncovers a slight bias of SOA research towards cost considerations (9/23), compared to studies including MS (12/49). This trend of a slight increase in the frequency of observation towards SOA still holds when examining the more precise comparison in Figure 4.7. Notably, it falls short of the 20% cutoff in difference, while being differently classified in the identification of relevant quality criteria in Section 4.3, where it is considered a relevant quality criterion for service-oriented architecture, but conversely not incorporated into the pool of significant MS quality criteria. Joachim (2011) highlight the "substantial investment" (p. 5) involved in SOA adoption. While monolithic systems come at lower initial cost, they can become more expensive in the long run (p. 5). Contrarily, SOA has the potential to reduce operational and IT cost in the long term (Joachim, 2011, pp. 5, 7; Niknejad et al., 2020, p. 2). Microservices may offer cost savings when scaled horizontally across multiple smaller instances compared to larger instances required for

monolithic vertical scaling (Jatkiewicz & Okrój, 2023, p. 1040). Benavente et al. (2022) subsume, that "a microservices-oriented solution reduces infrastructure costs compared to using a monolithic version" (p. 182), with AWS Lambda-based microservices incurring one third of the cost of their monolithic versions.

Modifiability

When comparing the incorporated research binarily, modifiability is not identified to be a significant differentiating factor between SOA (6/23) and MS (16/49). Increasing the granularity to single out studies that concern both MS and SOA, highlights an increased focus on modifiability in this domain. This is particularly evident as modifiability is highlighted as one of the most relevant criteria in the domain that encompasses microservices, SOA, and monolithic architectures, evidenced in 6 out of 9 studies. In the expanded analysis, this underscores the bias of studies on both MS and SOA towards modifiability, which is depicted in Figure 4.7. Minor modifications of a monolithic system might require a complete rebuild (Andriyanto et al., 2019, p. 282). Traditional SOA shares this limitation, with changes to its services also requiring a rebuild of the entire application (Andriyanto et al., 2019, p. 286). As applications grow, making modifications to the source code can become increasingly problematic, often leading to "unexpected behavior in other modules and a cascade of errors" (Blinowski et al., 2022, p. 20359). Therefore, modifying and upgrading small microservices is easier than dealing with monoliths, since they are easier to understand and can be changed independently (Y. Wang et al., 2021, p. 5).

Development

Development is observed at similar relative frequency across the two groups of studies discussing microservices (32/49), and those that do not discuss microservices, but only SOA (12/23). Expanding upon the juxtaposition, a similar distribution of studies is observed across studies on microservice architecture (without SOA) (24/27), and studies on both architectural styles (8/12). Being one of the most frequently observed criteria of discussed architectural styles (see Table 4.2), it is one of the central categories of quality criteria identified in this thesis and represents the discussion of most operational aspects, that are relevant to the assessment of focused architectural patterns. Thus, its constituent sub-criteria are examined more closely. Appendix Figure D.11 depicts the sub-criteria aggregated in the development criterion. It highlights the different distributions of sub-criteria across studies in service-oriented architecture and microservice architecture. The relative frequency of these sub-criteria in respect to the total amount of occurrences of the development criterion is now examined more closely.

Agile development is a factor more frequently concerned in studies on SOA (2/12) and especially its comparison to MS (2/8), than in studies on MS only (2/24). The difference does not meet the 20% significance threshold. The SOA sentiment, that "the typical SOA with coarse-grained services and ESB is considered not to be agile enough since it is still monolithic and interdependent" (Andriyanto et al., 2019, p. 286) prompts an architectural solution, represented by MSA. In contrast to SOA, all characteristics of MSA "are focused on supporting the achievement of its primary goal, i.e., agility or reduce the delivery time" (Andriyanto et al., 2019, p. 286).

The skills required from developers are emphasized in 3 out of 12 studies on SOA development and in 3 out of 24 studies on MS development. They are not mentioned in any comparisons of MS and SOA. According to Baškarada et al. (2020) "developing individual microservices may not presuppose any particular skillsets" (p. 6) , but "effectively implementing and efficiently managing a large application developed using microservice architecture requires significant advanced skills in distributed system development and DevOps" (p. 6). Therefore, the "assumption that, due to their relative simplicity, individual microservices are easy to develop" (p. 6) is misleading. Microservice developers "need to have a solid understanding of a range of non-functional aspects" (p. 6), as opposed to monoliths, where "developers can specialize in particular technologies" (p. 6). A stark difference is observed in the DevOps sub-criterion, which is never observed in research only on SOA development. It is a significant aspect of the development criterion in studies on MS (5/24), but mostly concerned in studies comparing the two architectural styles (3/8), highlighting the differentiating aspect of DevOps in development between SOA and MS. Baškarada et al. (2020) highlight, that the architectural complexity within individual microservices "cannot be managed manually" (p. 6) and require the use of DevOps tools.

Sentiments on the complexity, effort, and speed of the development process itself are observed in 5 out of 12 SOA development studies, making it the primary development sub-criterion of this architectural style. 6 of 24 Microservice development studies and 2 of 8 Studies comparing SOA and MS development exhibit this sub-criterion, showing the higher emphasis on development process measurements in SOA. Salii et al. (2023) count improved development velocity and team productivity among the most important benefits of MSA (p. 1675).

The autonomy and coordination of developers is the primary development subcriterion of Microservices, in 12 of 24 studies, since microservices "push towards design autonomy with plenty of small teams" (Cerny et al., 2017, p. 233).

Similarly, 3 of 8 studies comparing SOA and MS development concern autonomy and independence of development teams, but only 1 of 12 SOA development studies mention this sub-criterion, showing a higher emphasis on developers' autonomy, independence, and coordination in microservice architecture. Auer et al. (2021) highlight the ease, with which existing independent teams can migrate to and benefit from the independence and autonomy provided by microservices (p. 4). The structure of development teams, however, is not discussed in any studies comparing SOA and MS development, while being emphasized in 2 out of MS development studies and in 2 out of 12 SOA development studies. Without clear technical and organizational boundaries, "it is increasingly harder for the development team to retain a modular structure of the application" (Blinowski et al., 2022, p. 20359), which can hamper productivity when applications and the number of their developers grow. "Microservice-based applications scale well horizontally, not only in the technical sense, but also concerning the organization's structuring of developer teams, which can be kept smaller and more agile" (Blinowski et al., 2022, p. 20359). This highlights the arrangement of services and their respective developers around business capabilities as a strategy for agility (Andriyanto et al., 2019, p. 286).

Modularity

Being emphasized in 36 out of 49 studies including microservice architecture and in 15 of the remaining 23 studies on SOA, the criterion of modularity is also not identified as a differentiating aspect between MS and SOA. The minimal dissimilarity in the frequency of observation across the categories depicted in Figure 4.7 confirms this finding. The identification of most frequently concerned criteria in Table 4.2 highlights its relevance as one of the most frequently concerned quality criteria in every of the six discrete research domains. This highlights the importance of this complex criterion and makes it prone for further examination. The interrogation of modularity sub-criteria is depicted in Appendix Figure D.12 and shows the differences in their relative distribution across modularity studies. It depicts a slightly decreased emphasis of Ssorage mechanism & data management in SOA research (4/23), as compared to the academic work on both microservices and SOA (4/12) and slightly more so in MSA studies (12/37). Maintaining data consistency across multiple microservices, each with their own database, can become problematic and "may lead to inconsistencies in individual data stores" (Baškarada et al., 2020, p. 2; Leonard, 2009, p. 1021). Implementing services with a shared database avoids this problem, but makes the system a service-oriented, rather than a microservice architecture (Baškarada et al., 2020, p. 2). In addition to consistency challenges,

fulfilment of data longevity requirements is difficult, for databases distributed across different platforms and services (Leonard, 2009, p. 1020)

The sub-criteria of coupling & cohesion are concerned in significantly higher frequency in SOA modularity research (9/23), than in studies also concerning Microservices (3/12), let alone MSA modularity papers (5/37), making service coupling and cohesion the primary concern of SOA modularity research. Especially the compromise between these two aspects of modularity is concerned: Higher cohesion of services in SOA is preferred, but is indirectly proportional to the number of service interfaces, exemplifying the trade-off between coupling and cohesion (Zhang & Tanniru, 2005, p. 2268) and the need to integrate contradictory design goals. Due to the coarse granularity of SOA's services, they cannot be easily optimized for efficiency (Mahmood, 2007, p. 499). Further, Raj and Ravichandra (2018) measure a reduction of service coupling by 36% after moving from SOA to microservices (p. 1536).

On the contrary, the sub-criterion of autonomy & independence is emphasized in only 2 out of 23 SOA studies, with significantly increased focus given to this aspect in MSA studies (10/37). Finally, research, that concerns both MSA and SOA, centers on autonomy & independence as the primary sub-criterion of modularity (5/12). Interdependence between SOA services comes by design and regularly makes service-oriented architectures deployment monoliths, whereas microservice architecture requires the independence of its constituent services (Andriyanto et al., 2019, p. 286; Y. Wang et al., 2021, p. 24). Where SOA aims for loose coupling, microservices call for complete independence (Xiao et al., 2016, p. 63). Modularity is both one of the final quality criteria identified in this work and one of its constituents, due to the ambiguous semantic behind the term. The modularity sub-criterion makes up the primary research focus in MSA modularity research and is observed in 17 out of 37 MSA studies. It is given a starkly decreased focus in SOA research (3/23) and is of higher relevance in studies incorporating both SOA and MSA (3/12). Being modular, services are both cohesive and loosely coupled (Xiao et al., 2016, p. 63).

Interoperability

Interoperability is the final quality criterion not classified as a differentiating factor between SOA (17/23) and MSA (31/49), according to their juxtaposition. However, it is identified as one of the most relevant quality criteria in all six domains of Table 4.2. Thus, an in-depth analysis of its constituent sub-criteria is conducted in the following. Appendix Figure D.13 synthesizes the results of this analysis. Communication is the primary sub-criterion of interoperability across all research domains, partially explaining the small dissimilarities of this

criterion. Direct point-to-point communication between microservices over interfaces “enables developers to define data types and interfaces in a language-independent format” (Aksakalli et al., 2021, p. 15), but becomes disadvantageous at a higher number of services (Aksakalli et al., 2021, p. 15). Sentiments on Choreography and Orchestration are not found in the research on service-oriented architecture and are concerned in one of 37 MSA studies. Out of the 12 studies on both MSA and SOA, 2 discuss choreography and orchestration. MSA uses orchestration and does not require middleware to support service interoperability (Raj & Ravichandra, 2018, pp. 1533–1534). None of the incorporated studies on both SOA and MSA thematize Discoverability, whereas one out of 23 SOA studies and four out of 37 MSA studies discuss it, although Aksakalli et al. (2021) state microservice discovery as “one of the most critical issues in communicating microservices” (p. 16). Infrastructure technology represents the second-most relevant Interoperability sub-criterion across the domains of MSA (6/37), SOA (8/23), and their comparison (3/12). The ESB is the most prominent middleware element in centralized SOA infrastructure (Andriyanto et al., 2019, p. 285). This reliance on middleware introduces additional complexity, compared to monolithic architecture (Erickson & Siau, 2008, p. 5). Lastly, microservices, running in different processes, are “frequently deployed on different virtual machines” (Baškarada et al., 2020, p. 2).

5 Comparative Assessment of Architectural Styles

In the evolving landscape of contemporary software architecture, microservices have emerged as a focal point, frequently compared to other architectural styles—primarily Monolithic architectures and, to a lesser extent, service-oriented architecture. Unlike SOA, which is commonly analyzed as a standalone concept, microservices are frequently considered in response to the limitations of earlier architectural styles, including SOA. Despite the overlap in their functionalities, direct comparisons between MSA and SOA are infrequent, likely due to the blurred lines in their definitions. SOA still regularly features in comparative studies, suggesting its enduring relevance in architectural discourse. However, the rise of cloud-based solutions and serverless computing, has shifted technical responsibilities for infrastructure and platforms to cloud providers, further obscuring the distinctions between MSA and SOA. These services, while often labeled as microservices, contribute to the semantic confusion by blurring the lines between microservice and SOA terminologies as they manage the intricacies of infrastructure more akin to SOA principles. As microservices gain prominence and SOA sees a decline in focus, the trend is shifting towards a binary preference in architectural comparisons—between microservices and monolithic architectures as the default choice, only to be avoided in light of convincing reasons. This shift underscores the need for practitioners to carefully navigate the landscape, especially as serverless architectures redefine application deployment and influence the choice of architectural styles. Microservices aim to address both the technical and operational challenges seen with SOA, adopting many principles of monolithic architectures. This alignment suggests that while MS might seem like a rebranding of older concepts, they represent a significant shift in architectural philosophy and technology, impacting how organizations operate. Their differences and appropriate usage contexts are explored further in the following discussion.

MSA optimizes for independent scalability, which aligns well with the pay-per-use pricing models of cloud providers and facilitates flexible horizontal scaling. This starkly contrasts with SOA, where scalability is an insignificant criterion. Although stateless SOA services can scale independently, this capability is neither a required trait nor a typical outcome of SOA design. The difference primarily stems from MSA's focus on structural independence, enabling services to scale autonomously—a feature less emphasized and not inherently necessary in SOA. This puts the aim for the reduction of coupling (and consequently the maximizing of cohesion in services) at the core of microservice tenets, extending beyond the modular approach initially established in SOA.

Conversely, microservices manage their own data, facilitating truly independent scalability and preventing the bottlenecks associated with a centralized database. This independence increases the complexity of microservices but also enhances their scalability capabilities.

granularity is crucial in distinguishing between MSA and SOA, more so than in comparisons with monolithic architectures. Discussions about granularity often arise from the interpretation of the term micro in microservices, implying a smaller size. Contrarily, microservices encapsulate their own databases, making them, in principle, larger or more complex than the smaller services allowed in SOA. This granularity and independence in microservices lead to redundant data management, potential concurrency issues and inconsistencies, particularly when databases are replicated. This outlines a trade-off in services' horizontal scalability, which involves incorporating additional complexity into the scaling unit, making it larger, although finer-grained components are preferred for more efficient and precise horizontal scaling. However, the smaller the unit, the less independent its scaling can be due to potential interdependencies, such as those arising from extracting data management from services that rely on it. MSA strives for units that are as fine-grained as possible to optimize efficient and precise horizontal scaling while maintaining the maximum independence needed for such scalability. SOA generally organizes services around business functions with shared data, which undermines true scalability. Discussions on storage mechanisms and data management show that MSA places more emphasis on these aspects than SOA, with both architectures prioritizing data consistency. However, MSA incorporates database isolation, which, while enhancing scalability, introduces redundancy that deviates from traditional SOA principles aimed at efficient reuse and reducing redundant development efforts.

The difference between SOA services and microservices does not lie in their granularity, but rather their interdependency: regardless of their size, services need to be deployed together if they need each other to function or if they are coupled to the same database. To scale horizontally, bottlenecked services need to be replicated and therefor additional services need to be deployed. deployment is a crucial differentiator in architectural discussions. In SOA, deployment is an insignificant criterion because SOA typically involves composite services that may depend on one another, limiting independent deployability. Conversely, deployment is a significant aspect of MSA and a key factor distinguishing it from both SOA and monolithic architectures. Microservices prioritize independent deployability of each service. This feature aligns with the focus on development and operations in MS research, where deployment effectively

bridges the gap between these two phases, contrasting sharply with the more integrated service dependencies seen in SOA.

Performance emerges as another pivotal criterion in the context of MSA, especially when contrasted with monolithic architecture. This focus is often due to the granular and decentralized nature of microservices, which introduces network overhead to achieve interoperability. Such overhead becomes justified and advantageous at scale. Microservices are particularly beneficial for large and complex applications, which are inherently challenging to manage as monoliths. In such contexts, microservices offer improved scalability and the capability to independently scale numerous services, enhancing performance efficiency as the system grows. This implies that microservices are not only easier to scale, but also yield more efficient performance outcomes with the system's expansion. Enhancement of performance at scale is not a central concern but rather a possible side effect for SOA, that microservices put into focus. This emphasis on performance in microservices must be weighed against the trade-offs they entail at other points, such as complexity in deployment and maintenance. In contrast, performance as a quality criterion appears to be of limited concern within the realm of SOA. Technical performance is not a key driver for SOA adoption, implying that any performance gains in SOA are often seen as a secondary benefit rather than a primary goal. Hence, in scenarios demanding high performance at scale, microservices stand out as the preferred architecture.

The transition from the strategic perspective characteristic of SOA to a more practical, technical orientation in MSA is also reflected by the contrasting emphasis on cost within research. SOA's focus on cost reflects its alignment with broader organizational factors, underscoring cost as a significant driver in its strategic considerations. In contrast, MSA studies often adopt a more technical lens, emphasizing performance and resource utilization over direct cost implications. This shift suggests that within MSA discussions, cost, as a strategic organizational element, is less central. Moreover, the rapidly evolving market of computing infrastructure complicates generalized statements about costs in the context of microservices, which is regularly subject to the diverse pricing models offered by cloud providers. This variability makes specific discussions on resource utilization more pertinent than broad financial cost analyses. MSA focuses on increasing productivity and efficient resource utilization, rather than solely on the financial outlay, aligning with the architecture's technical and performance-driven priorities where organizational cost considerations are secondary. This approach aligns with the technical and performance-driven

priorities of MSA, where organizational cost factors play a secondary role compared to the operational efficiencies gained through optimized resource utilization. The nuanced focus in comparative studies of SOA and MSA further highlights that cost cannot solely delineate these architectures, as it is subject to multiple influencing factors.

Integration challenges differ markedly between SOA and MSA, primarily due to their architectural principles and data management strategies. MSA's typical database isolation complicates the integration of externally purchased microservices. Microservices' requirement to manage their own data potentially leads to compliance and consistency issues, further complicating the integration of external partners. SOA focuses on incorporating external applications, aiming to leverage compatibility benefits such as outsourcing and reusing externally purchased solutions. This approach may require fine-grained coupling, which contrasts with the MSA philosophy that emphasizes independence and minimal coupling. Integration in SOA facilitates the reuse of existing components without modifying the existing code, thereby expediting development processes, and reducing time to market. SOA is primarily an integration solution, focused on linking disparate systems and services to create a cohesive whole. In contrast, microservices prioritize building internally consistent, independent applications and tend to overlook integration, both in individual studies and comparative research with SOA. This oversight highlights those strategic organizational considerations, such as the integration of purchased external applications and the interconnection of business operations, are not primary drivers in microservices architecture. MSA is primarily employed to build discrete software applications from the ground up, rather than external integration.

Reusability is a critical factor when comparing SOA and MSA to traditional monolithic architectures. Both SOA and MSA significantly enhance reusability through service encapsulation, yet it varies significantly between SOA and MSA, with each framework adopting different approaches based on their design principles. SOA is highly geared towards reusability, facilitated by its design that allows services to be flexibly parameterized and easily integrated into various systems. This modular design promotes the recycling of service functionality to reduce development time and effort. Reusability is discussed more frequently in comparisons and less in individual studies on MSA. This reflects the significant difference in architectural priorities, where SOA's design inherently encourages reusability as a strategic objective, contrasting sharply with MSA's emphasis on service independence and developer autonomy. While encapsulation in MSA serves to improve the independence of development and

operation for individual services, it does not necessarily optimize their reusability. The primary goal of encapsulation in MSA is to isolate services and maximize their independence, not necessarily to prepare services for reuse in different contexts. The balance between reusability and development effort starkly differs between MSA and SOA. Generalizing and encapsulating functionality for reuse involves significant effort and may not align with MSA's focus on independent service development. In contrast, SOA strategically embraces reusability, justifying additional efforts when there is tangible reuse potential.

organizational criteria are only significant in research on SOA in isolation, often focusing on enhancing business agility and architectural agility. This means that SOA not only aims to align closely with business processes but also seeks to adapt swiftly to changing organizational needs. SOA tends to support agile development, but its primary focus remains on broader organizational agility, which includes optimizing business processes and structural adaptability of services. Organization statements in MSA studies mostly pertain to the scaling of organizations, whose architecture does not allow its technology to sufficiently scale with the organization, or that resources could not be used efficiently enough anymore after scaling the organization. MSA focuses less on general strategic organizational agility and more on the agility at the level of development and operations, mirroring shifts seen in practices like DevOps. MSA aims to enhance the agility of developing and operating individual services rather than the entire enterprise. This shift is indicative of a paradigm where services are designed to be independently scalable and operationally efficient, potentially at the cost of broader organizational agility. MSA addresses some of the limitations seen in SOA by fostering greater independence and agility within development teams. While SOA supports agile teams, it does not promote their independence and agility as unequivocally as MSA. In contrast, microservices aim to make the development of individual services more agile, focusing on rapid deployment and easier maintenance, which can streamline operational processes but might not directly enhance overall business agility. The distinction between how SOA and MSA approach organizational agility underscores their different architectural priorities. SOA aims to achieve a harmonious integration with business processes and overall organizational goals, enhancing the agility of the enterprise itself. Meanwhile, MSA prioritizes the agility of service development and operations, optimizing these aspects to achieve faster time-to-market and more responsive service management, which may not always align with broader organizational agility goals.

Infrastructure plays a foundational role in both SOA and MSA, but the technologies and approaches used differ significantly. SOA Primarily relies on centralized infrastructural components, such as the ESB, to facilitate integration and communication between different services. This centralized approach is integral to SOA's strategy for achieving interoperability. Conversely, MSA adopts a fundamentally different approach by leveraging containerization and virtualization technologies. These technologies isolate services from each other to increase their independence. This allows for diverse infrastructure choices at the microservice level, but also introduces variability in how services are integrated and managed. Infrastructure automation, particularly in the context of MSA, is pivotal for efficiently managing the build and deployment processes. As automation increases, it impacts the limits of service granularity within the microservices' architecture. Microservices can be scaled up to grow larger as they become easier to understand and manage. Conversely, they can also be scaled down further and become smaller, due to the simplified management and reduced proficiency needed by team members. Without infrastructure automation, microservices come at the risk of overly large growth as a result of trying to avoid efforts in maintaining redundant infrastructure for additional small services. Maintainability is a less emphasized concern in SOA due to its centralized infrastructure akin to monolithic systems. In contrast, maintainability becomes a central issue in MSA, due to the decentralized nature and the need for individual upkeep of each service. The structural and behavioral independence required in MSA leads to redundancies and necessitates more personnel for independent maintenance of individual services. However, this independence also reduces the scope and complexity of maintained software, as maintainers are typically more familiar with their specific components. In summary, the choice between SOA and MSA in terms of infrastructure depends significantly on the desired balance between centralization and flexibility, with each architecture offering distinct advantages and challenges in infrastructure automation and service maintainability.

Communication is crucial for interoperability and is identified as the most relevant sub-criterion across all examined domains. Both SOA and MSA fundamentally require network communication to facilitate seamless interactions between services. The complexities involved in these interactions are a central focus of research, emphasizing the need for effective management of communication protocols. The aptitude of an organization and its developers in implementing web-based applications significantly influences the suitability of adopting SOA or MSA, as these architectures depend intrinsically on network communications between multiple components. While security is a significant

concern, it is not the most relevant quality criterion within any of the analyzed domains. It is considered moderately important in both MSA and SOA but does not distinctly delineate them, as indicated by the increased focus on security in studies focusing on these architectures independently rather than in combination. Their reliance on network communication heightens their vulnerability to cyber-attacks compared to monolithic architectures, where fewer external interfaces are exposed. The commonly observed misconception that smaller services in MSA reduce the attack surface is countered by the reality that the increase in the number of services and their network interconnectivity broadens the system's external interface. Each service contributes to a larger cumulative attack surface due to the distributed nature of functionalities across various service interfaces. This distribution creates numerous potential attack vectors, as functionalities exposed for service interoperability are more accessible and vulnerable compared to those in a monolithic architecture, which can more clearly delineate exposed surfaces to potential attackers unlike monoliths. The need for discoverability in services makes it easier to identify potential weak points, reducing the effectiveness of information hiding strategies that are more manageable in monolithic systems. The complexity inherent in microservices can, however, be leveraged as a defensive advantage. The interconnected nature of service chains facilitates advanced monitoring and detection mechanisms, providing a layered security approach that can make the architecture more challenging to penetrate effectively. This offers sophisticated options for defending against intrusions and with it the potential for enhancing the overall security posture of the system. In contrast, monolithic systems may offer simpler security management due to their fewer interaction points and more contained architecture. This simplicity can translate into stronger security control, facilitated by a more centralized system management framework, where security measures can be more uniformly applied and managed. The advantages of centralized security management also apply to service-oriented architectures to a lesser extent.

The freedom of implementation technologies enables each microservice development team to align the technology stack of their microservice with their preferences. However, microservices environments require a broadened skill set by developers, since small teams are responsible for managing the entire lifecycle of their services, necessitating a diverse range of skills within each team. Microservices are supposed to be a method of independently realizing even small business domains, which is only possible if the development team can cover a broad skill set with few team members, setting a lower boundary to service and team size. The skill sets of developers significantly influence the

communication dynamics within a team and consequently affect the choice of architectural styles. The suitability of architectural styles can depend heavily on whether developers possess broad or specialized knowledge. In environments where the development staff consists of specialists with deep knowledge in specific areas, and the system scope is limited, a monolithic architecture might be more suitable. This setup allows for tightly integrated design and management, where specialized skills can be deeply focused on specific parts of the system. Conversely, developers with a broad range of skills are better equipped to handle MSA. Covering a broad skillset with relatively few developers allows each team to independently handle various aspects of their service, especially when the domains represented by microservices encompass a relatively small functionality. The efficacy of a microservices team hinges on its ability to encompass all necessary skills for the service's functionality. If a team frequently needs to seek external help for skills not covered within, it suggests that the service may be too narrowly defined, contradicting the principles of microservices that favor broadly skilled, self-sufficient teams. As microservices grow more complex, the scope of skills that need to be covered by team members tends to narrow. This can open opportunities to incorporate more specialized developers, enabling their focus on specific functionalities with greater efficiency and expertise. Such growth can also increase the need to further modularize a service. Microservices themselves can be structured according to layered architectural patterns to better leverage the strengths of specialized team members. In essence, the composition of developer skill sets not only affect their ability to work within a given architecture but also shape their communication relationships and the architecture itself. Thus, the selection of architectural patterns should consider the breadth and depth of available skills. Broadly skilled teams are preferrable in MS settings for their flexibility and autonomy, while specialized teams are better suited for structured environments like monoliths or SOA, where deep expertise can be better aligned with technical and enterprise architecture.

MSA prioritizes operations and development, supporting technical independence and enhancing developer autonomy, closely aligning with DevOps practices. According to Conway's Law — which proposes that organizations design systems mirroring their own communication structures — MSA is ideal for companies with distributed, small teams that operate independently. In contrast, SOA necessitates more extensive communication and dependencies between services and their developers. This reliance makes SOA less suitable in environments where minimal inter-team communication is preferred, potentially impeding the autonomy and agility that MSA facilitates.

Microservices, designed as small, independent monoliths, face practical limits on growth due to the need for each team member to fully comprehend their service. This becomes challenging as services expand in complexity, particularly in large companies where business functionalities are extensive. When a business functionality outgrows a single service or cannot be decomposed into strictly separate domains, it may become too cognitively complex to be understood by each team member, necessitating its division into multiple modules managed by different teams, or resulting in oversized teams, both of which conflict with the core tenets of microservices. Restructuring of the service's functionality and employment of multiple teams for the same service then become desirable, akin to the management of larger monolithic systems by multiple specialized teams. A cohesive microservice, intended to remain small and manageable, might then need to be split, creating new services with dedicated teams. However, this leads to data consistency challenges, potential redundancy, and significant additional communication overhead due to the need for services to interoperate over networks.

Overly dogmatic adherence to microservice architectural principles can be counterproductive. An objective assessment of the individual problem space is a crucial prerequisite for deciding on an architecture, and potentially reveals a more flexible approach to be advantageous. This understanding may necessitate a reconsideration of the architectural strategy, perhaps shifting it towards different styles that might be more appropriate. Essentially, microservices are ideally suited to small business domains where each service can be effectively managed by a compact, independent team. When business domains expand beyond a manageable scope for a single service, they may require division into multiple services, potentially leading to redundant databases. In cases where the domain's complexity or size makes the microservice model impractical, managing it as a monolithic system could be considered. Services, in this sense, represent an additional layer of strong modularization, designed to manage architectures that are too complex for a single monolith. However, one fundamental limitation of microservices is their reliance on small, independent teams, which caps the maximum complexity that can be effectively managed within each deployment monolith. For highly complex systems, a combination of independent microservices and interdependent components, each developed by specialized teams, might be more advisable. In certain cases, particularly when aiming to enhance the reusability of specific parts of a microservice that grows too large, it may be strategically beneficial to encapsulate these components into separate services, without allocating redundant functionality and staff for each extracted service. This method of modularization represents a

partial transition to SOA, allowing organizations to retain the benefits of microservices for managing large business functionalities. SOA allows for a large, cohesive business domain to be developed across multiple teams, with some aspects like infrastructure or data management being centrally managed, introducing monolithic aspects to increase efficiency. Splitting a microservice that has grown too large into multiple interdependent services, makes the partition of the architecture constituted by them monolithic in some respects, such as scaling and deployment. In SOA, the transition between how many parts of the system, each potentially consisting of several interdependent services, are completely independent of others, is fluid. This hybrid approach allows for a fluid transition between independent and interdependent services, to optimize for reusability, autonomy, and systemic cohesion where necessary. This hybrid architecture allows for a flexible adaptation to varying system complexities and team structures, accommodating the diverse needs and expansive domains of large, growing enterprise structures. This hybrid approach can harness the strengths of both microservices and SOA aspects within a broader SOA strategy, offering a flexible and dynamic solution that adapts to evolving system demands and development processes. It allows organizations to scale and evolve their architectures without compromising on the benefits of decentralization characteristic of microservices or the systemic integration offered by SOA.

Modifiability in microservices presents both challenges and advantages. While MS excel at allowing quick, independent modification of functionality within services boundaries, adjusting the overall architecture or service boundaries themselves can disrupt the system more significantly than in SOA. Such changes often require extensive adjustments across multiple services and their interactions, increasing complexity and potential integration difficulties. In contrast, SOA can offer more flexibility in modifying service boundaries due to its more centralized control mechanisms and less granular nature. However, the interdependencies in SOA might complicate modifications within individual services, as changes could ripple through the interconnected services. While it is feasible to completely replace services in MSA, complex challenges arise when modifying their boundaries. In essence, MSA excels in modifiability within individual services but encounters difficulties with broader architectural changes, whereas SOA offers a balanced approach, with some flexibility in redefining service boundaries but less agility in modifying individual services.

Technology heterogeneity distinctly sets MSA and SOA apart from monolithic systems. Both MSA and SOA inherently allow for the use of diverse programming languages through service encapsulation, enhancing flexibility at the

individual service level. This capability is extended in MSA to include different technologies for data storage, which is not necessarily the case in SOA. However, on an architectural level, technology heterogeneity is generally not desirable as it reduces the overall flexibility of the architecture and the organizational structures mirroring it. While this diversity allows developers the freedom to work with their preferred technologies, it comes at the cost of flexibility in the development teams' structures and complicates refactoring, when service boundaries change. Such changes might require functionalities to be rewritten by different teams, using incompatible technology stacks. The heterogeneous technological environment across services and teams can similarly complicate relocating developers within the organization, possibly making it more feasible to replace them with new hires who are proficient in the specific technologies required for a different service.

Microservice developers, like their services themselves, become more replaceable than reusable because microservices often utilize mutually unknown technologies across different services. Their isolation can make onboarding new employees to their domain more straightforward than bringing a developer from one microservice team to another, which might involve not just teaching them a new technology but also a completely new domain, to which they might not have been exposed yet. As long as no refactoring is required, the freedom to select implementation technologies simplifies tasks for developers, enables rapid innovation and potentially increases developers' productivity. Developers need only understand their specific service and can focus solely on their assigned domain. MSA places them in an environment that they can more easily comprehend but limits their understanding of the system as a whole and other microservices in particular. Trading architectural flexibility for technological flexibility at the service level poses significant risks when the architecture needs to adapt or evolve beyond its initial boundaries. Opting for microservices is therefore justified when the benefits of rapid and agile development of services outweigh the risks associated with later refactoring across service boundaries. This decision locks in the boundaries of services, optimizing the system's scalability—until those boundaries need to change.

6 Results: Microservice Identification

6.1 Taxonomy of Microservice Identification Approaches

The microservice architecture development process described by a microservice identification approach spans multiple phases of the overarching software development process, partially encompassing the requirement analysis and design phase, excluding the implementation phase (Schröer et al., 2020, p. 154). Each phase encompasses multiple activities, which in turn are characterized by the application of a finite set of techniques to produce artifacts. These are the constituent elements of the process of microservice architectural design, beginning from its starting point as the initial state and leading to the design of a microservice architecture encompassing a set of candidate microservices. Because of the increasing number of different microservice identification approaches and in order to address RQ 5.1, this research has produced a taxonomy for microservice identification approaches.

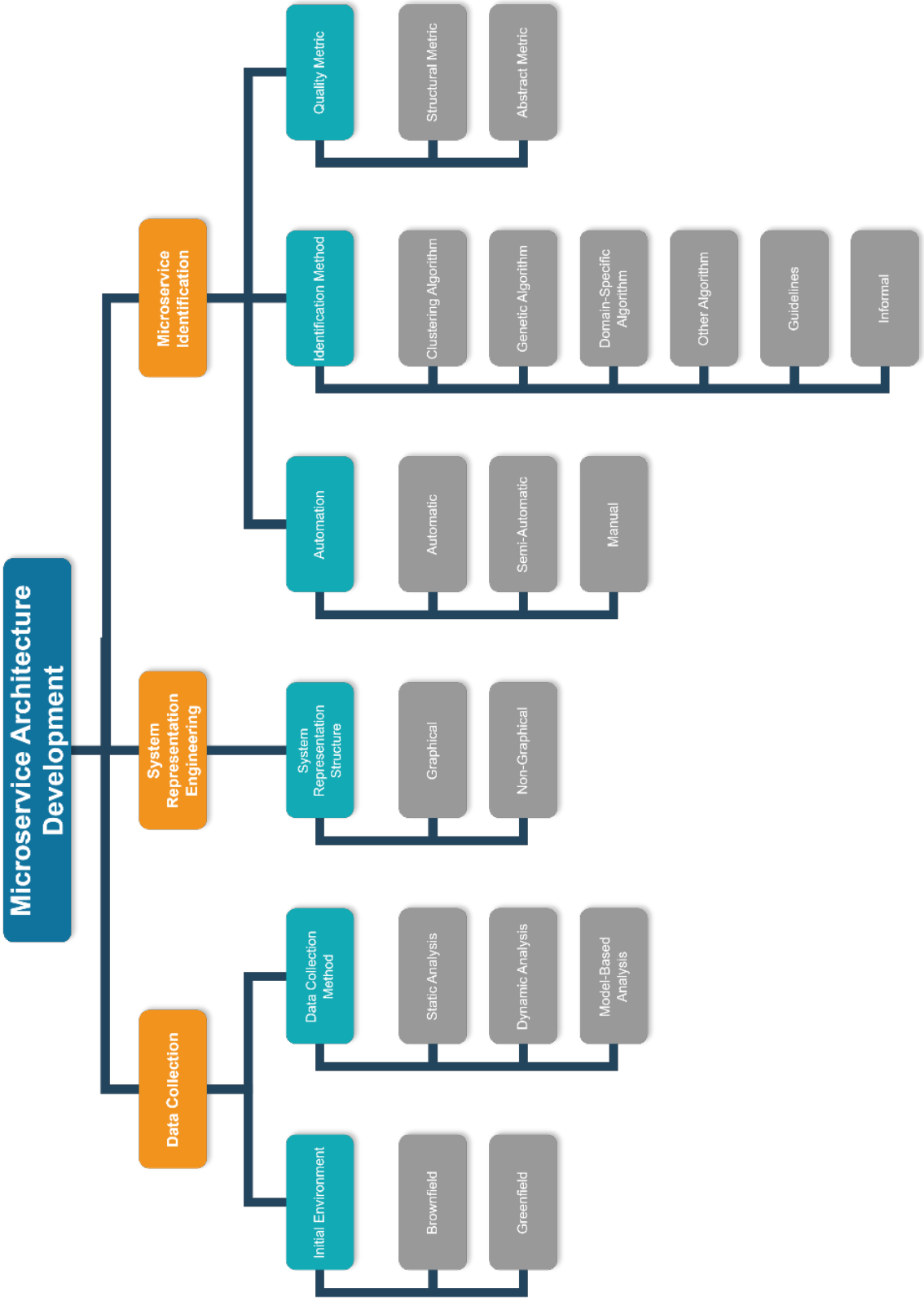


Figure 6.1: Taxonomy of approaches to the development of an architectural design for microservices

The taxonomy classifies the processes followed by microservice identification approaches. To aid the readers in conceptualizing the research topic and to set the stage before progressing to the detailed presentation of the findings, the taxonomy is presented as an introduction to the results. The composition of the

taxonomy, with its categories and their constituent concepts, is presented in Figure 6.1. The mapping of studies to categories of the taxonomy is presented with the analysis of the phases along which it is structured to inform its analysis and the formation of the framework. Appendix Figure E.1 presents the augmented taxonomy, including augmentative descriptors of the literature synthesis view, from top-level categories representing phases of the process, to subcategories representing activities, techniques, or artifacts, and lastly their constituent concepts. Some concepts are further categorized through sub-concepts, which the figure does not depict. For a comprehensive literature synthesis view, consult Digital Appendix 7: Microservice Identification Literature Synthesis Matrix. The following textual definition details the concepts, including sub-concepts of the augmented taxonomy and literature synthesis view.

Phase I – Data Collection

Acquisition and comprehension of data from the underlying starting point.

Initial Environment

Outlines the development approach and its starting points

- Brownfield: Pre-existing application system as basis
- Greenfield: Start without pre-existing legacy code or application system

Data Collection Method

Describes the method for analysis and extraction of data

- Static analysis: Examination of source code and analysis of data from the source code repository without executing the software system.
- Dynamic Analysis: Utilization of runtime information from executing the software system for data analysis.
- Model-Based Analysis: Analysis based on models rather than software, focusing on theoretical frameworks and representations.

Collected Data

Augmentative descriptor, detailing the data extracted from the initial environment using data collection methods.

Phase II - System Representation Engineering

Transformation of collected data into a system representation suitable for microservice identification.

Data Transformation

Augmentative descriptor, outlining the transformation of collected data into a coherent system representation, acknowledging the complexity and diversity of procedures involved in creating a representation of the system, and possibly accommodating multiple stages of processing and intermediate artifacts.

System Representation Structure

Final representation of the system, providing a clear distinction between the types of inputs used for microservice identification regarding the degree, to which they are structured.

- Graphical: Hierarchical, relational, and structured data, including matrices, trees, and networks.
- Non-Graphical: Literal, independent and less structured system representations.

System Representation

Augmentative descriptor detailing specific data used to represent the system, expanding on the structure of the system representation as the foundational input for the partitioning process in microservice identification.

Phase III - Microservice Identification

Delineation of the Microservice Architecture design, involving partitioning, evaluation, and optimization to finalize a set of microservice candidates.

Automation

Categorization of the microservice identification process according to its level of automation.

- Automatic: Fully automated process, eliminating human bias from the system representation onwards
- Manual: All steps and decisions are taken manually by the architect, with potential subjective influence
- Semi-Automatic: Blend of automated processes and manual decision-making, integrating elements of both

Identification Method

Methodology for interpretation of the system representation to identify microservice candidates, providing an overview of the strategies used for the generation of microservice architectures.

- Clustering Algorithm: Algorithmic grouping of similar entities based on their attributes.
    - Hierarchical Clustering
    - Centroid-Based Clustering
    - Density-Based Clustering
- Genetic Algorithm: Employs principles of natural selection to solve optimization and search problems by generating and evolving solutions over generations.
- Domain-specific algorithm: highly specialized custom methods, tailored to address unique characteristics of specific domains.
- Other Algorithm: Various specialized algorithms tailored to specific aspects of microservice identification.
    - Topic Modelling: Identifies topics to understand themes in large volumes of text.
    - Solver: Algorithmic solving of mathematical optimization problems.
    - Label Propagation: Assignment and propagation of label information for classification.
- Guidelines: Structured recommendations and best practices to guide the identification and partitioning of microservices .
- Informal: Less structured methods, often based on expert opinion or ad-hoc approaches, without strict adherence to formal algorithms or guidelines.

Identification Algorithm

Augmentative Descriptor, providing insight into specific algorithms and tools applied during microservice identification.

Identification Process

Augmentative descriptor, examining the identification methods in detail, shedding light on operational steps, and underlying principles, including the refinement of initial partitioning to design the microservice architecture and its constituent microservices.

Quality Metric

Categorizes the quantitative measures used to guide and assess the microservice identification process. Metrics are categorized as follows:

- Structural Metrics:

    - Coupling: Evaluates the dependencies and connections between microservices.
    - Cohesion: Assesses the internal coherence of microservices.
    - Modularity: Measures the degree to which the system is compartmentalized
    - Granularity: Considers the absolute and relative size and scope of microservices
- Abstract Metrics
    - Reusability: Gauges the potential for microservices to be reused in various contexts.
    - Complexity: Addresses the intricacy of microservices and the effort required for development and maintenance.
    - Network Overhead: Quantifies the data volume incurred by network communication between microservices.
    - Cost: Estimates expenses associated with microservices.
    - Performance: Combines metrics such as response time and throughput to evaluate the efficiency of microservices.
    - Organizational Impact: Estimates the effects of microservice architecture on team structure and resource allocation.
    - Functional Requirements: Measures the completeness and correctness of the microservices in fulfilling the specified requirements.
    - Reliability: Assesses the dependability and stability of the architecture.
    - Security: Considers the security implications and constraints during the partitioning process.

Metrics

Augmentative descriptor, detailing the specific metrics applied in assessment of the microservice architecture.

6.2 Overview of Selected Literature

This subsection presents the profile of incorporated research in terms of date and venue of publication.

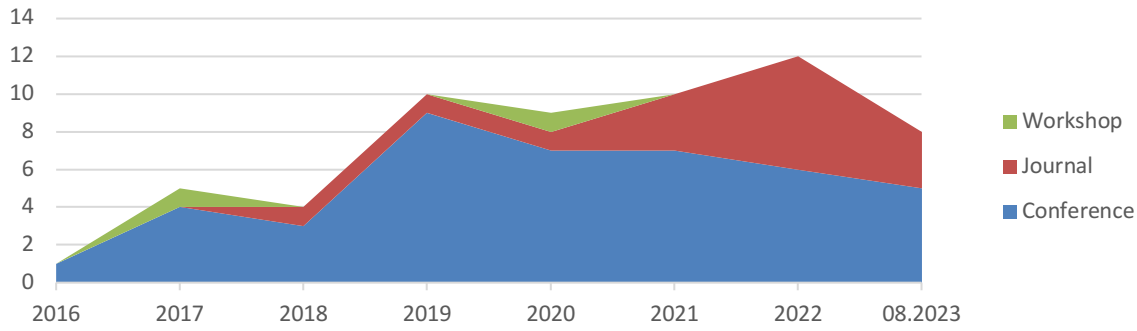


Figure 6.2: Distribution of studies per year and publication venue

Figure 6.2 depicts the temporal distribution of incorporated literature. It shows that none of the incorporated approaches were published before 2016. The analysis evidences the emergence of the topic of microservice identification approaches in 2016 with one single study and its significant gain of traction since 2019. Appendix Figure E.2 highlights the predominance of conferences as the most common publication venue (40/56), followed by journals and workshop proceedings. No included studies resulted from workshops after 2020. Starting from 2020, the number of studies published in journals increases significantly.

6.3 Data Collection

Appendix Figure E.3 shows that brownfield approaches are the most common (46/56), with 40 out of the 46 brownfield approaches specifically stating monolithic applications as legacy assets. More often than non-monolith brownfield approaches are greenfield approaches at a total of 10 (17.9%) of studies.

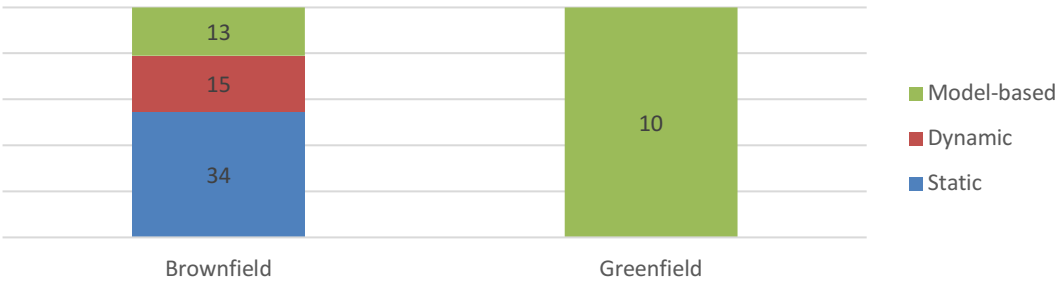


Figure 6.3: Data collection methods organized by the initial environments to which they are applied

As highlighted by Figure 6.3, all greenfield approaches apply model-based analysis. Brownfield approaches exhibit application of all three data collection methods, with most applying static analysis, followed by dynamic analysis and model-based analysis.

Appendix Figure E.4 depicts that, while most brownfield studies only use a single approach (67.57%), many brownfield approaches studies (30.44%) of studies apply multiple data collection methods in combination. Dynamic analysis is

predominantly applied in combination with other data collection methods, at 11 of 15 cases. Model-based analysis is also mostly used in combination with other approaches. 6 studies on the migration of legacy systems apply model-based analysis as the only data collection method. Dynamic analysis is seldom used as a singular approach and usually used in combination with other approaches. Static analysis, the most prevalent brownfield data collection approach is also the only approach mostly used by itself (22/34).

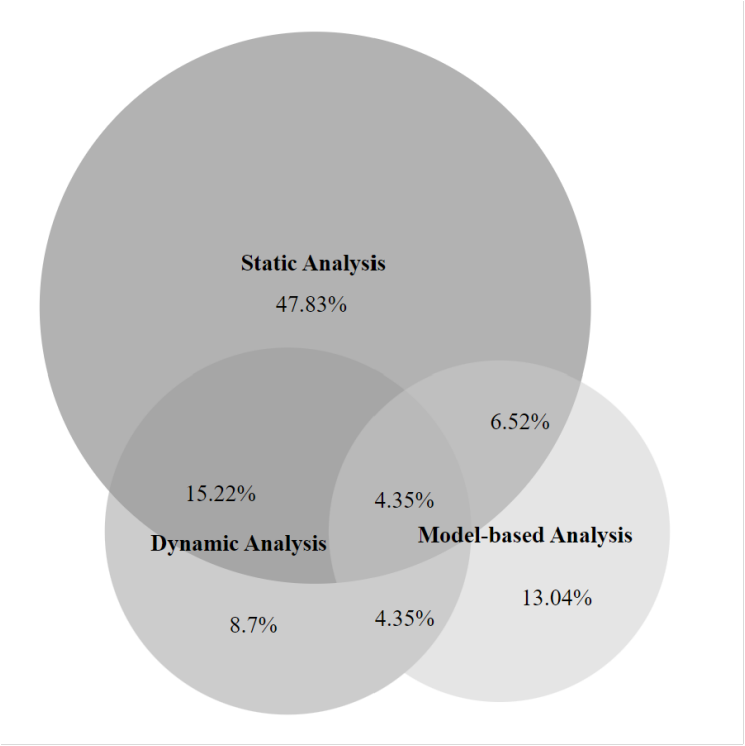


Figure 6.4: Combinations between data collection methods in brownfield studies

The Venn diagram pictured in Figure 6.4, shows an opposing trend between the combination of static analysis with model-based analysis, and the combination of static analysis with dynamic analysis. Firstly, dynamic analysis is used almost twice as often in combination with static analysis than as a standalone approach to data collection. Contrarily, model-based analysis is used as a stand-alone approach twice as often as in combination with static analysis. Dynamic analysis is mostly used to augment static analysis and is also the most frequent combination of approaches found in the literature. Model-based analysis contrasts with dynamic analysis in its application as a stand-alone approach when starting from brownfield.

6.4 System Representation Engineering

The data yielded by data collection methods is transformed via data modeling in 92.86% of studies, with only 4 out of 56 studies not engaging in any data transformation, as shows in Appendix Figure E.5. Engineering of a system

representation based on initial collected data is a prevalent practice among researchers, usually applied as a means of generating a structured graphical system model. This approach, in turn, underpins the microservice identification process, where utilization of graph-based models is widely adopted.

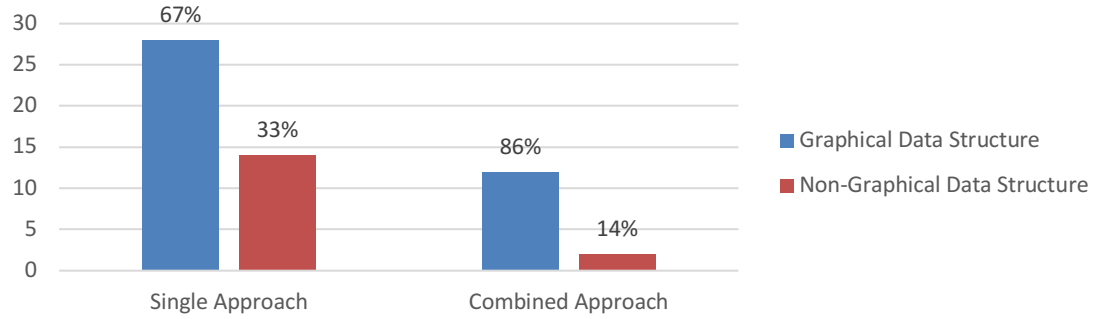


Figure 6.5: Graphical system representations and combined data collection methods

One third of studies utilizing a single data collection method do not transform data into a graph. This tendency is starkly reduced in studies that combine at least two approaches (14%), shown in Figure 6.5. Furthermore, both identified studies that combine all data collection methods integrate collected assets into graphical system representations.

6.5 Microservice Identification

At 61%, most examined studies automate the microservice identification process completely, followed by manual identification at 23% and finally a combination of both paradigms in semi-automation at 16%.

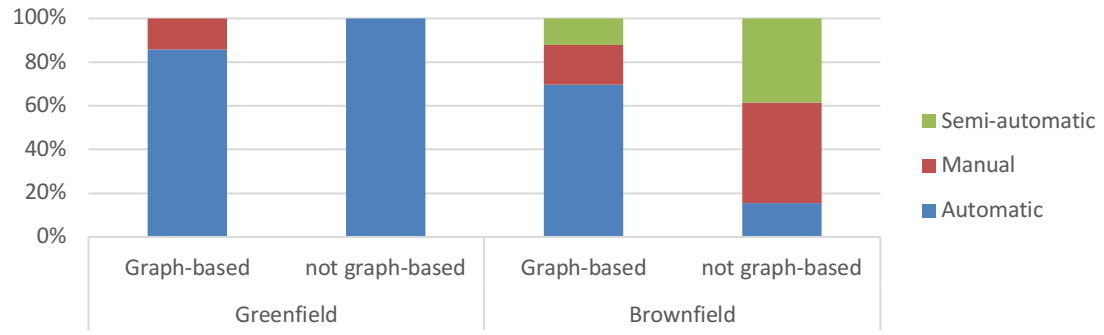


Figure 6.6: Automation of graph-based approaches across initial environments

Over 85.3% of automatic microservice identification methods illustrated in Appendix Figure E.6 are based on graphical system representations. 53.8% of manual identification methods are graph-based. Only semi-automatic methods do not mainly use graphs as system models, with only 44.4% of semi-automatic methods using graphs as system representations. At 85% and 100%, greenfield approaches, as presented in Figure 6.6 are highly likely to automate microservice identification processes fully. Graph-based brownfield approaches,

while not quite as likely to apply automatic microservice identification methods, stand in stark contrast to brownfield approaches making use of less structured system representations at 15%.

More specifically, within graph-based brownfield approaches relying solely on a single data collection method, the degree of automation depends on the data collection method applied, with model-based analysis (75%) being the most frequently used in automated identification methods, for static analysis, followed by static analysis (67%) and dynamic analysis (50%). Hierarchical clustering algorithms stand out as the most prominent method for microservice identification in Appendix Figure E.7, making them the leading choice among various clustering algorithms. Alongside centroid-based and density-based clustering, algorithmic clustering represents the most widespread approach to microservice identification. Interestingly, the application of guidelines is found to be as common as the use of genetic algorithms in this context, with both ranking second in popularity for microservice identification, trailing only behind hierarchical clustering. Within the spectrum of centroid-based clustering, k-means and its variants are the most utilized, while DBSCAN is the most prominent choice for density-based clustering. Notably, the employment of genetic algorithms is equally popular as the formulation and application of general guidelines, which provide a framework for rule-based identification in a less formalized manner than algorithmic approaches.

Appendix Figure E.8 presents microservice identification methods with the degrees of automation of identification processes, during which they are applied. It highlights genetic algorithms are exclusively applied in fully automated identification processes across all initial environments. This is also the case for approaches applying centroid-based clustering. While not being purely automatic, hierarchical clustering algorithms also exhibit a significant tendency towards full automation. However, some hierarchical clustering approaches require the final steps of the partitioning process to be performed manually. Studies that apply guidelines for partitioning are never completely automated, but in some cases support the decision-making process through technical visualizations. The statistic dependence between the degree of automation similarly holds for the system representation's degree of structure, as shown in Appendix Figure E.9. Notably, clustering algorithms are more often based on graphical system representations than genetic algorithms and informal microservice identification is slightly less frequently applied to graphical structures than guidelines are.

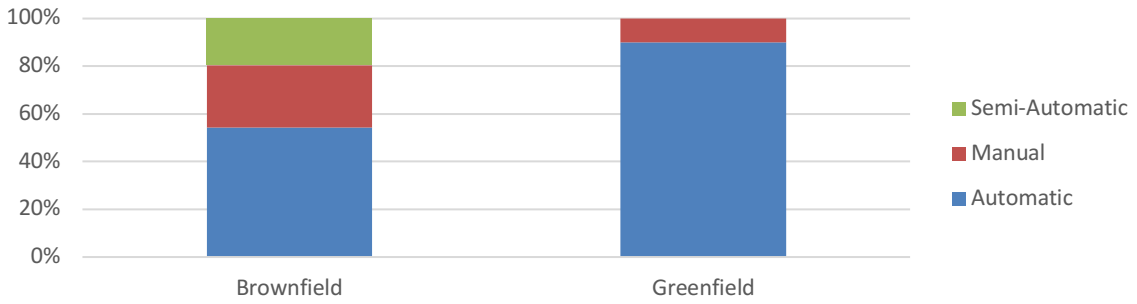


Figure 6.7: Automation of approaches, depending on their initial environment

In microservice identification, greenfield projects predominantly use automatic methods, occasionally employing manual techniques for specific needs, as illustrated in Figure 6.7. Brownfield development also favors automation to efficiently handle legacy complexities but frequently incorporates manual approaches for nuanced understanding and integration. Semi-automatic methods, combining both automated and manual processes, are less commonly used in brownfield projects.

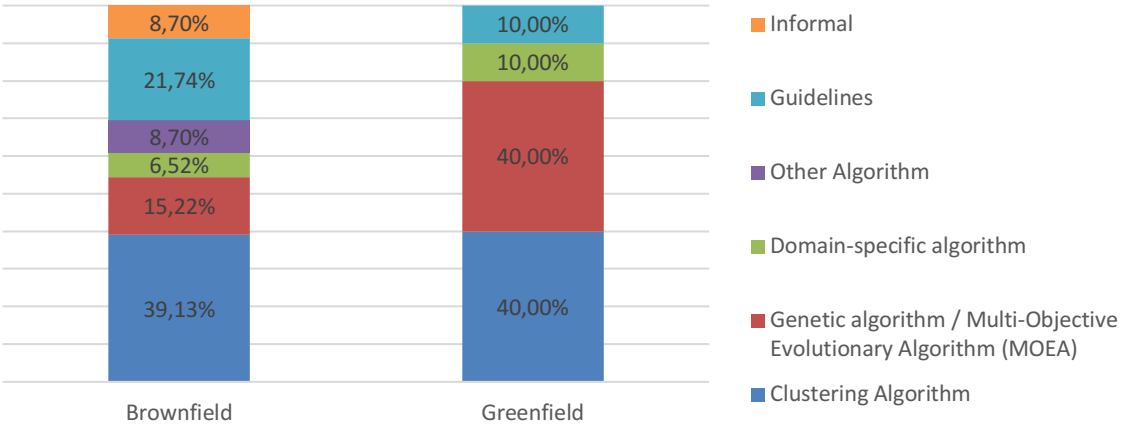


Figure 6.8: Microservice identification methods organized by initial environments

Clustering algorithms have emerged as the leading method for identifying microservices in both brownfield and greenfield projects. In greenfield scenarios, genetic algorithms are equally favored alongside clustering algorithms, with guidelines and informal methods also playing significant roles. Contrastingly, in brownfield projects, the application of guidelines ranks as the second most popular approach, followed by genetic algorithms and other specialized algorithms. Informal identification methods are exclusively utilized within brownfield contexts, highlighting the nuanced differences in approach based on the initial environment. Within microservice identification, a clear difference emerges in the application of non-clustering and non-genetic algorithms between greenfield and brownfield projects. Greenfield approaches predominantly use domain-specific algorithms, suggesting a targeted strategy for new

developments. Conversely, brownfield projects leverage a wider array of methods, including all identified sub-categories. When examining clustering algorithms specifically, greenfield projects stick to hierarchical clustering. Brownfield initiatives, however, utilize a broader selection of clustering techniques: hierarchical, centroid-based, and density-based clustering.

Appendix Figure E.10 outlines the frequency, with which identified quality metric categories are applied in considered quality-driven approaches. Cohesion and coupling are identified as the foremost metrics in architecture evaluation, highlighting their importance at design time. Granularity and size metrics follow, emphasizing structure and scalability. A notable trend is the combination of cohesion and coupling metrics to assess modularity, illustrating their comprehensive application. Network overhead, an extension of coupling through transaction volumes, is often linked to cost and performance predictions. Reusability follows in measured frequency as a standalone criterion, reflecting its significance. Additionally, a variety of functional requirements are analyzed together, showcasing the diverse and complex nature of software quality assessment.

Appendix Figure E.11 shows a clear increase in the diversity of metric categories used over the years. Initially, in 2016, the studies incorporated did not apply any metrics, indicating a baseline for the trend. This changed in 2017 with the introduction of 2 metric categories, followed by a steady growth to 4 categories in both 2018 and 2019. The trend accelerated in 2020 with 7 categories, continued to grow to 8 in 2021, and reached its peak in 2022 with eleven metric categories. Seven metric categories are applied in incorporated research published between January and August 2023. However, an extrapolation to the end of the year, points towards an increase to nine metric categories by year-end. Despite a slight reduction from the peak in 2022, 2023 is projected to have the second-highest measured diversity in metric categories. This progression underscores a trend towards broader and more nuanced approaches to evaluating microservice candidates.

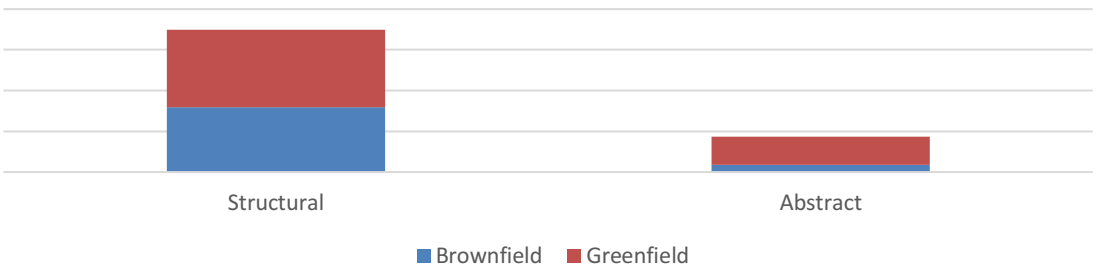


Figure 6.9: Structural and abstract quality metrics by their relative frequency in initial environments

Figure 6.9 compares initial environments by their use of abstract and structural metrics as aggregates of their respective quality metric categories. The absolute numbers are normalized by the number of studies per initial environment, to clearly present the relative biases of different starting points towards metric categories. A clear tendency of greenfield approaches to the application of the more varied category of abstract metrics can be observed. For a more detailed overview of quality metrics, organized by the initial environments of the approaches utilizing them, refer to Appendix Figure E.12. Static analysis and dynamic analysis exhibit a similar variety of metrics, with both data collection methods being applied in combination with eight metric categories. Model-based analysis is used in combination with a wider array of analysis methods, as depicted in Appendix Figure E.13. Expanding on model-based analysis, greenfield approaches apply 12 different metric categories, whereas across all brownfield approaches utilizing model-based analysis, nine different metric categories are applied. The variability of metric categories is still higher in approaches starting from greenfield, although brownfield approaches partially apply additional model-based analysis in combination with other data collection methods. Appendix Figure E.14 shows the difference between microservice identification processes regarding their degree of automation, by comparing the number of metric categories applied in studies exhibiting each of them. Across all examined studies, 12 metric categories are applied to automatic identification approaches, followed by six applied to semi-automatic identification approaches and four to manual ones. Furthermore, all identified greenfield approaches applying metrics are automated and all greenfield approaches that use metrics perform microservice identification using automatic genetic algorithms or automatic hierarchical clustering.

6.6 Quality Metrics

The following details the underlying metrics of quality metric categories. For an overview, consult the full concept matrix in Digital Appendix 8: Concept Matrix of Quality Metrics.

Coupling

"Coupling represents the dependencies and connections of a microservice to another" (Cojocaru et al., 2019, p. 87). It comprehends the manner of interdependence and interconnections between software modules or functionalities to describe their criticality to others (Carvalho et al., 2019, p. 87). Service dependencies, at their core, are directional. When coupling metrics are based on dependencies evidenced through relationships between services, the absolute

dependence of services on others is measured as the number of other MS that an MS depends on (Vera-Rivera et al., 2020, p. 91). Vera-Rivera et al. (2020) specifically measure it as the "number of microservices from which [a microservice] invokes at least one operation" (p. 95). Similarly, efferent coupling quantifies the directional dependence on the level of entities within a service: Bajaj et al. (2022) quantify efferent coupling as the number of classes within a respective microservice, that depend on at least one class outside of the microservice (p. 67015). Conversely, the absolute importance of a service, as used by Vera-Rivera et al. (2020) quantifies the "number of clients that invoke at least one operation of a microservice's interface" (p. 89), while Bajaj et al. (2022) measure afferent coupling through incoming dependencies as the "number of classes outside this microservice that depend upon classes within this microservice" (p. 67015). Afferent and efferent coupling are used together in all 4 studies that apply them, and absolute dependence is always used in conjunction with absolute importance across the three studies utilizing them. However, the two perspectives are mutually exclusives, as no studies combine them. Directional dependencies and connections are oftentimes aggregated. As such, the coupling instability index relates efferent coupling to the sum of afferent and efferent coupling, such that a service without outgoing dependencies is seen as completely resilient to changes of other services (S. Li et al., 2019, p. 12). In the case of extant efferent coupling, its stability rises with an increase in afferent coupling (S. Li et al., 2019, p. 12). Vera-Rivera et al. (2020) measure the number of interdependent microservices pairwise based on absolute importance and dependence to compute microservice interdependence (p. 89). Inversely, the metrics of lack of cohesion measures the number of independent microservice pairs (Vera-Rivera et al., 2021, p. 117184). Similarly, the interaction number metric gives perspective on the amount of calls between interdependent microservices and the strength of their coupling (Brito et al., 2021, p. 1414). Another class of coupling metrics measures the ways, in which a microservice enables coupling. Nine studies apply coupling potential metrics as the basis for dynamically introduced dependencies during runtime based on exposed interfaces and operations. An interface is any class that exposes functionality as an endpoint, with the methods for each interface being considered as operations (Brito et al., 2021, p. 1414). Saidani et al. (2019) measure the number of operations publicly exposed by an extracted microservices to other candidate microservices in order to compute the average operation number as an aggregate coupling metric of the whole MSA (p. 61). Brito et al. (2021) quantify the number of interfaces exposed by a given service, with a smaller interface number representing a higher

likelihood of a service having a single responsibility, representing a typical approach to interface number metrics (p. 1414).

Cohesion

This quality metric characterizes "the structural coherence of the microservices" (Sellami, Saied, et al., 2022, p. 211). Cohesion of inner elements is an important aspect of functionality modularization and a central aspect of microservice architecture and oftentimes "used as a secondary criterion in some academic solutions; coupling is considered the main criterion" (Carvalho et al., 2019, p. 23). At its core, cohesion measurements describe connections between assets within the same service. Where coupling describes connections between assets within different microservices, cohesion measures relationships of assets within the same service. "Cohesion is the manner and degree in which inner elements – data and behaviors – of a single module (or functionality) are related to each other" (Carvalho et al., 2019, p. 23). "Cohesion represents the degree to which the operations provided by a microservice cater to only one functionality" (Cojocaru et al., 2019, p. 87). It describes relationships among elements within a module or service. S. Li et al. (2019) make use of relational cohesion, defined as "the ratio between the number of internal relations and the number of types" (p. 12) within each service. Oftentimes, cohesion metrics measure the similarity or commonality of entities without considering their interconnections, e.g. B. Liu et al. (2022) measure it through both similarity between interface parameters and their domain similarities (p. 394). No study using similarity metrics combines more than one type thereof. The applied similarity metrics refer to different entities and closely related to this is the concept of cluster purity, where microservice architectures are comprehensively scored based on the wholistic encapsulation of complex entities within respective services, such as database objects or business contexts, with the most prominent purity metric, business context purity measuring "the entropy of business use cases per partition" (Nitin et al., 2022, p. 8).

Modularity

Modularity measures the "strength of division of a network into clusters/communities" (Brito et al., 2021, p. 1413). "Higher modularity represents dense connections within nodes in a community but sparse connections between different communities" (Brito et al., 2021, p. 1413). While coupling merely measures inter-service dependence and cohesion describes intra-service coherence, modularity aggregates the two. Thus, modularity encompasses metrics that purportedly measure cohesion, consider both internal cohesion and external coupling relationships. Nitin et al. (2022) compute cohesion as the ratio of

internal edges to the sum of internal and external edges (p. 8). This metric uses coupling dependencies to relate the internal cohesion of a microservice to its external coupling, and while tightly interwoven, differs from both internal cohesion and external coupling themselves. Eight studies apply Structural modular quality, which is "associated with both the cohesion and coupling criteria [...] and combines them into a single metric" (Sellami, Saied, et al., 2022, p. 211) to "measure the modularity quality of partitions based on structural cohesiveness and coupling" (Kalia et al., 2021, p. 1218). Another approach to modularity quality is the quantification of conceptual modular quality. It is applied in three studies and hinges on common textual terms between classes (Sellami, Saied, et al., 2022, p. 211). It evaluates "how focused the contexts represented by the microservices are" (Sellami, Saied, et al., 2022, p. 211). Seven studies measure the ratio of boundary-spanning calls as inter-call percentage, combining internal cohesion and external coupling (Kalia et al., 2021, p. 1218). Many modularity metrics, such as the silhouette coefficient, also relate the cohesiveness of a service to its separation from other services by comparing the similarities of their constituent entities (Santos & Paula, 2021, p. 55). In an MSA, with MS candidates "composed of methods belonging to several features" (Assunção et al., 2022, p. 51), feature modularization divides the number of occurrences of the predominant feature of each MS by the sum total of its features occurrences.

Reusability

Reusability is directly measured in different facets by multiple studies: Khoshnevis (2023) measures the average in-space reusability degree of microservices, as "the number of configurations a microservice is used in, to the total number of configurations" (p. 9). They also determine an adapted commonality degree of MS candidates as the number of microservices containing at least one complete mandatory business activity (p. 7). Reusability of a microservice is indicated by its extant reuse in the MSA in three studies. Among them, Assunção et al. (2022) aim to capture if microservice candidates are reused within the system (p. 51).

Granularity

This metric category is partially expressed through size measurements, with its use corresponding "to the size of each microservice and the size of the application" (Vera-Rivera et al., 2021, p. 117185). Santos and Paula (2021) measure granularity of an MS through the "number of services grouped in the same microservice" (p. 57). Sellami, Ouni, et al. (2022) determine granularity in relation to the legacy system as the "ratio of the number of classes in the original legacy system by the number of candidate microservices" (p. 6). Size

measurements are also related to more complex approaches to granularity quantifications: Non-extreme distribution "measures the distribution of individual cluster sizes" (Qian et al., 2023, p. 6) to avoid uneven distribution and centralization of classes.

Complexity

Two studies make use of user stories and map their complexity to the services realizing them in order to infer the complexity of their realization and speed of the service development process, with higher service complexity increasing the effort of its development (Vera-Rivera et al., 2021, p. 117186; Vera-Rivera et al., 2020, p. 89). Vera-Rivera et al. (2021) additionally base a metric of cognitive complexity based on operation-weighted service interfaces, coupling, and user story points (p. 117186).

Network Overhead

"Network overhead is a criterion designed specifically for microservice architecture" (Assunção et al., 2022, p. 8). The network overhead is defined as the sum of the sizes of the network traffic data to each microservice (Carvalho et al., 2020, p. 573). In the same vein, communication overhead is "the amount of time a system would spend on performing actions not directly addressing the user needs" (Carvalho et al., 2019, p. 23). As such, it concerns "the negative impact of extracting microservices on time spent along future microservice communication, which was originally performed locally (e.g., via function calls)" (Carvalho et al., 2019, p. 23). Carvalho et al. (2019) further emphasize the difference to coupling as "the degree between different elements, while overhead is time consumed in network communication" (p. 23). Assunção et al. (2022) create a heuristic based on dynamic analysis data to predict network overhead (p. 11). They compute network overhead directly as parameter size of each operation of each interface, obtained during the execution of the legacy system and consider additional "network overhead caused by the adopted protocol to communicate with the future extracted microservices" (p. 11). Run-time performance metrics offer direct measurements of latency and throughput, indicating resource requirements through communication overhead based on document sizes. Cojocaru et al. (2019) compute performance at design time based on network overhead "as the longest synchronous call or the average size of message, for asynchronous calls" (p. 88). The greenfield approach of Vera-Rivera et al. (2021) makes use of calls and requests as communication relationships between microservices to measure performance at design time (p. 117185). They state that "estimating the performance of an application at design time is difficult and imprecise" (p. 117185) while assuming, that calls and requests between

microservices increase "communication, latency, and response time of the application" (p. 117185), directly affecting its performance. Communication overhead metrics are also applied as a means of estimating cost: J. Li et al. (2022) compute communication overhead of inter-service method calls based on the distances between assigned deployment locations of different microservice candidates to infer its incurred expenses (p. 423).

Cost

Staffa et al. (2021) infer costs by a coupling metric enriched by a differentiation between read and write operations to predict costs more precisely (p. 835). Additionally, they measure replication cost as the amount of redundant replications of data entities (p. 835). Gouigoux and Tamzalit (2017) repeatedly test microservice candidates for deployment and quality assurance costs to iteratively optimize the partitioning (pp. 63–64).

Performance

The meaning of performance depends on the context any several studies apply performance metrics that go beyond communication overhead. performance in microservices "is usually regarded as a combination of response time and throughput" (Cojocaru et al., 2019, p. 88). In this way, Nitin et al. (2022) consider latency and throughput as performance metrics obtained through dynamic analysis of the monolithic legacy system during MS identification (pp. 7–8).

Organization

Direct measurements of organizational effects during microservice identification are rare. Z. Li et al. (2022) estimate the rate at which a reduction of team size is to be expected (p. 9). They focus on the team reduction of each extracted microservice relative to the monolithic system and calculate the average candidate microservice team reduction rate (pp. 9–10). The higher the value of the team size reduction rate, the more likely the division of microservices is to improve a development team's quality (p. 10). J. Li et al. (2022) consider the business criticality of classes to prioritize the quality and availability of computing resources (p. 428).

Functional Requirements

Two studies apply functional completeness metrics. Functional completeness of the MSA in regard to the entities that are supposed to be partitioned into microservices is measured by Sellami, Saied, et al. (2022) as the "percentage of classes from the monolith included in the decomposition" (p. 212). Functional

correctness of produced partitions, meaning their ability to correctly function is subject to exclusionary resource conflicts, as by J. Li et al. (2022), who also consider pre-existing functional constraints (pp. 427, 429). Khoshnevis (2023) also takes functional correctness into account as the rate of consistent valid architectures produced through their approach (p. 9).

Reliability

Reliability is exclusively considered by J. Li et al. (2022, p. 427). They consider availability constraints to encapsulate availability-critical entities together in microservices endowed with additional resources, to increase system reliability (p. 427). They integrate data consistency criticality and storage types in their approach, increasing reliability through data consistency (p. 427).

Security

Finally, security requirements and constraints, including security criticality and information security levels of classes, affect partitioning in one study (J. Li et al., 2022, pp. 426–427).

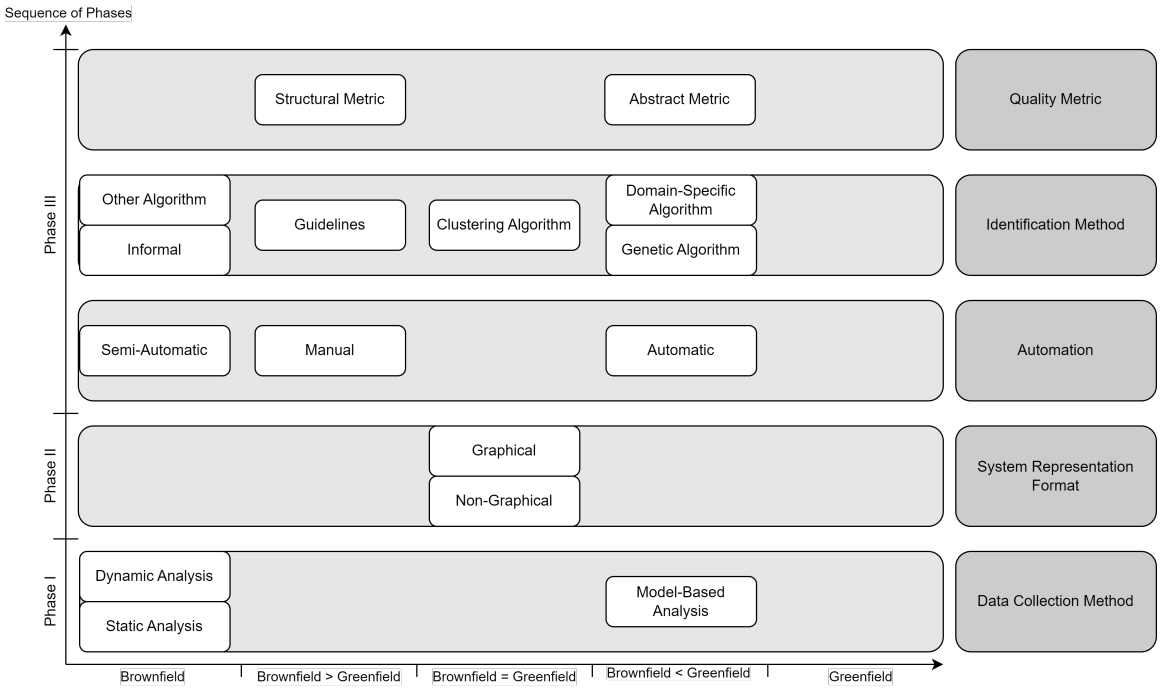


Figure 7.1: Microservice Architecture Design Framework (MSADF)

To address RQ 5.2, this study introduces MSADF, which organizes all taxonomical concepts according to their relevance and compatibility with the two primary starting environments in microservice architecture development. Figure 7.1 depicts MSADF, organizing taxonomical concepts according to their applicability to, and their closeness of affiliation with initial environments of the architecture development process culminating in architectural design. The organization of concepts is complicated by the frequent intersections among them, which is therefore clarified through an in-depth analysis of these overlaps in the following. Notably, the framework employs an ordinal scale to represent the applicability and affiliation of concepts, based on their occurrence relative to a respective chosen starting point of approaches, in which it is observed. Additionally, concepts positioned towards the left of the taxonomy are more commonly associated with other concepts techniques favoring that side, indicating a bias towards brownfield approaches. Conversely, concepts and techniques leaning towards the right show a predisposition towards greenfield methodologies. This nuanced approach mitigates potential oversimplifications from treating all concepts as variables, that depend solely on their initial environment. By identifying interdependencies among concepts, the framework accommodates the indirect affiliation of concepts linked to an initial environment through their association with other relevant concepts, increasing its precision.

Data Collection Method

Static analysis is the leading data collection method for brownfield projects, being uniquely suited to these environments. Dynamic analysis follows, a method also exclusive to brownfield contexts. Model-based analysis, although pivotal in greenfield projects as the only method used, is less emphasized in brownfield projects.

The applicability of static and dynamic analysis methods makes legacy application migration more flexible in terms of data collection methods, which partly explains why brownfield approaches remain the primary research focus in the field of microservice identification approaches. Specifically, monolithic legacy systems are commonly the initial focus of migration efforts. In contrast, greenfield development demands well-established documentation practices and the availability of abstract models from the outset, a challenge further compounded by inflexible refactoring, when initial architectural design deprecates and needs to be overhauled and refactored in light of better information. Dynamic analysis, despite its advantages, is rarely used as an isolated strategy. The reason for this is that the data it generates is often incomplete, making it less effective on its own. This limitation underscores its primary role as a supplementary technique, complementing other data collection methods with runtime information. Conversely, model-based analysis stands out as sufficiently comprehensive for use on its own, even in brownfield projects, where it adeptly navigates the complexities of legacy systems. This stark contrast demonstrates the independence of model-based analysis, as compared to dynamic analysis, further validating the merit of greenfield approaches, which can achieve comprehensive insights without analyzing legacy systems. In environments with less complete brownfield settings, where existing software systems are accessible for monitoring but lack available source code, dynamic analysis becomes particularly valuable. This method approximates and reverse engineers aspects typically explored through static analysis, akin to reverse engineering software architecture. It demonstrates dynamic analysis' capacity to incorporate critical insights from legacy systems in cases where static analysis is inapplicable. Furthermore, introducing additional data into a legacy system facilitates a model-based analysis approach. This method effectively expands the information space to include data typically associated with greenfield projects, creating a hybrid approach that leverages the strengths of both brownfield and greenfield methodologies. Such integration not only expands the information space but also adds complexity to the system representation engineering process. Despite this added complexity, there is a significant upside: it has the potential to

improve the quality of the microservice architecture by providing a more comprehensive view of the system and enabling the application of additional quality metrics. Therefore, incorporating model-based data encourages a deeper understanding and optimization of the system's architecture in legacy systems migration.

Dynamic analysis plays a pivotal role in assessing runtime quality metrics of legacy systems, enabling the integration of these insights into system representation. This approach unlocks the potential for evaluating system partitions in ways not possible during the design phase, by providing a live view of the system's operation and performance. However, when it comes to weaving various requirements into system representations, model-based analysis proves to be particularly effective. At the heart of design-time evaluations are metrics of granularity or modularity, such as coupling and cohesion, which consider various dimensions of the system's architecture. The incorporation of non-functional requirements and runtime data into the data subjected to these evaluations, significantly enhances its analysis. By incorporating these elements, otherwise basic assessments gain depth, offering a more nuanced understanding of the system's architecture and behavior. Model-based analysis benefits from a wider array of applied metrics thanks to its ability to draw on a more extensive and diverse dataset. This breadth of data provides the opportunity to analyze a broader spectrum of relevant criteria, making the insights derived from model-based analysis particularly comprehensive. The integration of both dynamic and model-based analyses, therefore, enhances the overall evaluation process, combining the strengths of operational insights with the thoroughness of structured, model-based evaluation.

System Representation Structure

The encoding of collected data in graphical system representations lends itself to the integration of a complex information space into one comprehensive data structure. This trend is supported by the shift of methods towards complex data structure, that integrate multiple data collection techniques in response to the growing complexity of data. As data complexity escalates, the adoption of sophisticated graphical data structures to simplify data analysis becomes crucial, allowing for a more comprehensive and efficient approach to system representation, manipulation and understanding. This makes graphical data collection methods closely related to brownfield approaches. However, greenfield approaches closely approximate brownfield approaches in terms of the frequency in which graphical system representations are used. Of the brownfield approaches that only use a singular data collection method, studies applying

model-based analysis for data collection most often produce graphical system representations. Thus, both the combination of data collection methods and usage of model-based analysis by itself leads to graphical system representations, making them equally closely related to both initial environments.

Automation

Although greenfield approaches automate microservice identification more often than brownfield approaches, the majority of microservice identification processes starting from brownfield are automated. The strong correlation of graphical system representations and fully automated microservice identification highlights the suitability of structured system representations, when aiming for automatic microservice identification methods. As graphical system representations are increasingly applied with rising dimensionality of collected data (in respect to the data collection methods producing them), combination of data collection methods can be seen as a predictor of automatic microservice identification methods. Brownfield approaches utilizing graphical system representations invariably employ full automation in microservice identification, contrasting with those brownfield approaches that do not use graphical representations and thus never fully automate microservice identification, relying solely on manual or semi-automatic methods. Semi-automatic identification methods are exclusive to monolith decomposition but also their least used method. This, in turn, makes brownfield approaches combining multiple data collection methods more aligned with automatic microservice identification, less so to semi-automatic microservice identification and lastly to manual identification. Such combinations are exclusive to brownfield settings, positioning them as particularly adept for identification processes that utilize graphical system representations, considering this factor alone. However, the same relationships hold for brownfield approaches applying only model-based analysis, which makes them closely akin to greenfield approaches and closely related to automatic microservice identification. Given that model-based analysis forms the cornerstone of all greenfield approaches, this similarity further reinforces their connection. Lastly, static analysis is less frequently the precursor to automatic microservice identification compared to model-based analysis, yet it precedes automatic identification more commonly than dynamic analysis does. Nonetheless, graphical representations of systems are predominantly analyzed through automatic microservice identification techniques and brownfield approaches use manual identification methods more frequently than greenfield approaches, which use them only in fringe cases.

Microservice Identification Method

While all microservice identification methods are observed in brownfield approaches, informal methods are exclusively used in brownfield approaches; they are, in principle, still applicable in greenfield development. This is also the case for all non-genetic and non-clustering algorithmic methods. Brownfield approaches also apply guidelines more frequently than greenfield, whereas genetic algorithms are applied at a significantly higher rate in greenfield approaches than in brownfield. The former also apply domain-specific algorithms more regularly. Lastly, clustering algorithms are used equally often across both environments. The higher complexity of the information space in brownfield is reflected by the increased variability of microservice identification methods. This is underscored by taking the sub-concepts of clustering algorithms into account: Clustering algorithms are common to both starting points, but only brownfield approaches employ all identified sub-concepts of clustering, as opposed to greenfield methods which resort to hierarchical clustering only. This reflects the greater flexibility in microservice identification within brownfield contexts compared to the rather focused solution space in greenfield development. The application of algorithmic identification methods rules out manual identification processes. The lack of fully manual identification processes in greenfield approaches can thus be explained by the application of algorithmic methods across all examined studies, starting from a clean slate. Furthermore, the prevalence of genetic algorithms in greenfield approaches is the underlying factor of the higher cohesion between greenfield environments as starting points and automatic identification, given their common use in fully automated processes. Clustering algorithms are also majorly applied in fully automatic microservice identification. Underscoring a general trend towards automation, studies show that informal processes are often supplemented with automatically generated visualizations rather than being entirely manual. A notable pattern emerges from the research: It is either the reliance on model-based analysis or a blend of various data collection methods, that culminates in the creation of graphical system representations. These, in turn, facilitate the automatic identification of microservice candidates, predominantly through clustering or genetic algorithms, marking a common pathway through the varied landscape of approaches to microservice architectural design.

Evaluation

Greenfield approaches are characterized by a more frequent and diverse application of metrics compared to brownfield approaches, with this diversity primarily driven by the use of abstract metrics. This is partially explained by the

increased amount of quality assessments in model-based analysis as compared to static and dynamic analysis. Even when normalizing for this fact, greenfield approaches apply a wider array of measurements at the design stage. Similarly, the degree of automation predicts the variability of applied metrics, with all quantitatively evaluated greenfield approaches utilizing automatic microservice identification. Finally, an archetypical greenfield approach lending itself to quality assessments, is identified: all quality-driven greenfield approaches perform microservice identification automatically, using genetic algorithms or hierarchical clustering. Therefore, this study highlights that for practitioners aiming to ensure quality in greenfield microservice architectural designs, genetic and clustering algorithms stand out as proven methods for microservice identification.

8 Conclusion

This study has applied a systematic literature review methodology and conducted two literature reviews to identify a total of 1,748 studies, which were systematically screened and filtered to obtain a final set of 128 studies. The studies were analyzed in depth to answer research questions regarding the comparative assessment of architectural styles and studied approaches to the development of architectural designs for microservices. 539 unique excerpts from the 72 selected studies from the first literature review were organized along the architectural styles they assess and a multiplicity of quality criteria, which have been synthesized into a final set of 20 quality criteria. The concept matrix of quality criteria provides a differentiated overview of criteria relevant to the assessment, comparison, and selection between the included architectural styles. It organizes the selected studies binarily across the quality criteria, creating a total of 1,404 data points. The taxonomy of microservice identification approaches, produced from the second literature review, provides a holistic view of the processes observed in the literature and contains six categories. For the 56 incorporated studies, this creates 336 data points. The augmented taxonomy, presented as the literature synthesis matrix, contains 12 fields, encompassing a total of 672 data points. Finally, MSADF refines the preceding taxonomy and maps the literature according to five categories in respect to their suitability and applicability to different initial environments, based on 280 distinct data points. This framework considers the specific requirements of organizations and seeks to provide a practical evaluation of these methodologies, paving the way for more informed and effective architectural decisions.

Answering RQ1, cost, deployment, development, governance, granularity, Interoperability, integration, analyzability, maintainability, modifiability, modularity, technology heterogeneity, organization, it business alignment, performance, portability, reusability, reliability, scalability, and security have been identified as the primary quality criteria related to service-oriented and microservice architecture, as presented in the literature.

Answering RQ1.1, the frequency count of quality criteria discussed in studies that specifically pertain to service-oriented architecture has been analyzed to identify Interoperability, reusability, it business alignment, modularity, finally followed by organization, integration, and development as the most frequently emphasized SOA quality criteria.

Less commonly cited but still significant are cost, granularity, maintainability, security, technology heterogeneity, modifiability, governance, and scalability.

Answering RQ1.2, modularity, followed by reliability, Interoperability, maintainability, and development stand out as the most significant quality criteria associated with microservices.

Criteria such as performance, deployment, scalability, security, portability, analyzability, granularity, and reusability, while not being among the most emphasized quality criteria, are still identified to be significant in the context of microservice architecture.

Answering RQ2, service-oriented architecture is primarily assessed in isolation, whereas microservices are majorly studied in comparison with other architectural styles. Both Service-oriented architecture and microservice architecture are frequently assessed in comparison to monoliths. Among the comparative studies, monolithic architecture functions as the predominant baseline and main artifact used for the comparison. Consequently, when service-oriented and microservice architectures are compared, this is usually done in addition to comparisons of either with monolithic architecture and only seldom exclusively between microservices and SOA.

Answering RQ3, the quality criteria, that differentiate Microservices most significantly from SOA, and are predominantly observed in microservice research are performance, reliability, deployment, scalability, portability, analyzability. The differentiating criteria, significantly more frequently observed in SOA research, are reusability, it business alignment, integration, organization, governance.

Finally and building on the results of RQ1, RQ2, and RQ3, relationships between the quality criteria presented in the literature and the architectural styles discussed in their context, have been compared to answer RQ4 in-depth. The comparative analysis was conducted in Section 4.4.3 and discussed in Section 5.

One literature review was dedicated to answering the fifth research question, and an overview of the state-of-the-art in microservice identification approaches to support the development of an architectural design of microservice architecture was presented. To facilitate this, relevant studies were identified, evaluated, and synthesized.

Answering RQ5.1, a taxonomy of microservice identification approaches was constructed to systematically classify relevant research across several dimensions of the architectural design process. It was applied to the incorporated studies and augmented by descriptors, facilitating the classification and

description of examined approaches to microservice architecture design in the primary phases of the process.

The primary phases of the microservice architecture design process are data collection, system representation engineering, and microservice identification. The major constituent elements of the data collection phase are the initial environment, the data collection method, and the collected data. The major constituent elements of system representation engineering are data transformation and system representation. Lastly, the major constituent elements of the microservice identification phase are the identification method, the identification process, and quality metrics.

Answering RQ5.1.1, greenfield and brownfield have been identified as the two primary initial environments to which microservice identification approaches are applied and which classify their pre-existing assets, with monolithic legacy systems as the predominant brownfield asset.

Answering RQ5.1.2, the methods used to analyze and collect data from initial environments are threefold, with static analysis, dynamic analysis and model-based analysis being used in the production of initial data artifacts.

Answering RQ5.1.3, data obtained from initial environments are then transformed into graphical and non-graphical system representations as the two main classes of inputs to microservice identification methods.

Answering RQ5.1.4, several categories of microservice identification methods have been identified. Clustering algorithms, genetic algorithms, domain-specific algorithms, and other algorithms, which encompass mathematical solvers, label propagation algorithms and topic modelling techniques, have been identified as the algorithmic classes of identification methods, which are always at least semi-automatic. Further, the application of semi-formal guidelines and informal identification methods have been observed as manual and semi-automatic approaches to partition system representations in the design of microservice architectures.

Answering RQ 5.1.5, the quality metrics during microservice identification have been categorized to identify a host of directly measured quality criteria. Quality-driven approaches to the design of microservice architectures proposed in the literature evaluate microservice candidates in respect to four structural criteria: coupling, cohesion, modularity, and granularity. Nine additional abstract criteria have been identified: reusability, complexity, network overhead, cost, performance, organizational impact, functional requirements, reliability, and security.

Answering RQ5.2, this research has produced MSADF, a framework and guideline to microservice architecture design for greenfield and brownfield development. It relates all previously defined concepts of microservice identification approaches to these initial environments in respect to their suitability and frequency of application to them.

8.1 Threats to Validity

To conduct this study effectively, a broad range of literature needed to be covered. The research methodology, derived from the framework by vom Brocke et al. (2009) was designed to rigorously and systematically explore software architecture literature, thereby enhancing the rigor, particularly in the "reliability and validity of the search process" (p. 5). Reliability was addressed through meticulous documentation of the research process, enabling replicability, while validity was supported by a systematic approach to the selection and evaluation of sources. Despite these precautions, there remain inherent limitations that might threaten the validity of this study's outcomes. The potential exclusion of relevant studies might limit the ability to generalize the findings, posing a threat to external validity. This was addressed by using a diverse selection of databases and incorporating high-ranking publications by means of a journal search, which predicated the choice of scientific databases and helped enhance the quality and relevance of the included studies. Broad keyword searches across multiple databases and stringent selection criteria were employed to balance the review's depth and breadth, although this approach may inadvertently exclude pertinent studies. The entire selection process was conducted by a single researcher, introducing potential bias, mitigated by adhering to a transparent and systematic review protocol. However, the limitations imposed by a single researcher's perspective and the exclusion of results from forward-backward searches mean that some biases remain unavoidable. Additionally, the findings have not been empirically validated in practice, indicating that practical implications are based primarily on theoretical evaluations from the literature. This underscores the need for cautious application of the conclusions drawn. The methodological framework adopted here—including concept matrices, a comprehensive summary table, and a literature synthesis matrix—provides a detailed and accessible presentation of the findings. The integration of summary tables offers a synthesized, yet detailed, overview of the research terrain, enhancing the clarity of the literature review. The concept matrix provides a thematic overview across papers, while the summary tables offer detailed insights into each paper, making the findings comprehensive and accessible. The literature synthesis and analysis of the second review shape a comprehensive

overview of the current research landscape, enabling a nuanced analysis that goes beyond the confines of standardized taxonomies. It aims to deepen the understanding of microservice identification approaches for practitioners, allowing for focused analysis in relation to the research questions. The explication of phases in both the taxonomy of microservice identification approaches and the MSADF scopes the analysis of incorporated research by partitioning the concepts of the process of obtaining candidate microservices, employing an ordinal rather than cardinal scale to account for validity threats. This integrated approach ensures that the presentation of findings is both comprehensive and detailed, thereby facilitating reliability and transparency while enhancing the overall validity and reliability of the research.

8.2 Research Agenda

The subsequent chapter illuminates significant research gaps across various domains related to enhancements of decision-making processes in software system development, indicating a ripe field for future exploration.

For future research within the context of the selection between architectural styles, a focused evaluation of identified quality criteria through case studies is proposed. These studies would serve to compare the theoretical insights from this thesis with practical implementations, validating and possibly refining the developed framework. To this purpose, case studies that detail the application of quality attributes, metrics, and selection criteria within various architectural styles should be identified. Their comparative analysis can reveal practical adherence to, or deviations from, the theoretical expectations. Such empirical evidence could fine-tune the framework of quality criteria relevant to architectural style selection.

In the exploration of architectural style selection, it's imperative to consider both the possibilities of architecture development and the strategic use of migration and architectural patterns. Establishing guidelines for available migration patterns is vital for managing transitions between architectures, providing clarity on the feasibility of implementing chosen styles. An in-depth examination of how migration patterns are utilized offers a new dimension to the taxonomy of microservice identification approaches, enhancing the understanding of the feasibility of various approaches and their individual steps. Architectural styles, being architectural patterns themselves, are essentially composites of more specific architectural patterns. They are not strictly separate; there is significant overlap and intersection at the pattern level. Mapping these patterns onto architectural styles lays the groundwork for creating detailed guidelines

for selecting subtypes within a particular style. Furthermore, a comparative analysis of the use and combination of architectural patterns adds another layer to the comparison of architectural styles. This method facilitates the development of hybrid architectural styles by merging desired patterns from various origins, leading to more nuanced decision-making in their hybridization. This approach involves discerning compatible patterns and understanding the constraints certain combinations, such as integrating an ESB while maintaining separate databases for each service, may impose. Such hybridization acknowledges diverse technical solutions, enabling the customization of architecture to meet specific project needs. Furthermore, this perspective outlines future directions for hybrid architectural styles, advocating a strategic blend of elements from different styles.

In the domain of microservice identification approaches, model-based analysis is a pivotal data collection method, that should be more closely examined. It encompasses all data collection from greenfield environments and consequently produces all data available to greenfield microservice architecture development. The data collected by model-based analysis is varied and the complexity of data produced using this method has been shown to effect subsequent steps. Future research should be conducted to account for the high degree of variability in the data collected in greenfield approaches, and model-based analysis in general. Specifically, the classification of model-based analysis should be expanded upon to refine this aspect of the framework. Initially collected data, especially when produced by model-based analysis, should be standardized, and elevated from an augmentative descriptor to a full-fledged taxonomical category. This enhancement aims to deepen the understanding of different data foundations for identification approaches, enabling a more precise distinction between greenfield and brownfield scenarios. Concurrently, it's crucial to align the metrics used during the architecture development with the collected data they are based on, allowing practitioners to discern, which metrics are applicable to the extant assets, that characterize their initial environment. Such an alignment will benefit brownfield environments, where it can forge a stronger linkage between data foundations and the measurement of quality criteria, but is particularly essential in greenfield contexts. The insights gained thereby are expected to yield superior guidelines for quality-driven microservice architecture development and allow for more precise evaluations of the suitability of legacy assets for their migration.

It is also essential to focus on quality metrics applied prior to the identification phase in microservice architecture development and the evaluation of legacy

assets concerning their quality attributes or specific requirements in this context. In this thesis, only metrics applied during the phase of microservice identification are considered. For future work, a more detailed distinction between metric-based quality evaluations during other phases of the development process is necessary. Although this research touches on relationship between the choice of inputs, as categorized by data collection methods, and applicable metrics during microservice identification, it is crucial to recognize that already collected data might be embedded with requirements and quality metrics, thus influencing further development. During data modeling, metrics or requirements are often integrated into the system model, affecting the architectural decisions made downstream. Future research should therefore explore, how early-stage metrics and requirements are interwoven into system representations used for microservice identification and how they impact the architectural design process. The consideration of these preliminary influences can aid in the establishment of more comprehensive guidelines on quality-driven architecture development.

The framework presented in this thesis centers on quality-driven identification methods and their applicability to initial environments. However, these methods can be further refined by categorizing them along additional dimensions, such as the explainability of their results or the granularity of partitioning they facilitate. Furthermore, identification methods can be segmented into various sub-phases of microservice identification, from the initial generation of partitioning to their optimization and subsequent evaluation. Such differentiations are usually found in approaches applying different identification methods in sequence. This calls for an extension of the framework to consider sequential combinations of identification methods, producing outputs further refined by other methods. Such combinations should be focused on in future research on suitable combinations of identification methods. By conducting quality assessments at each sub-phase, a more thorough analysis can be performed, enriching the potential for developing nuanced metrics and enhancing quality evaluations. This refined categorization provides a structured pathway for future research to build upon the work established in this thesis.

The current framework focuses on the initial phases of microservice-based system development and does not cover later stages. Future work should include these later steps into the framework and expand the existing taxonomy to reflect subsequent stages of the development lifecycle. This will ensure a comprehensive approach to microservice development from inception to deployment. By incorporating successive stages, the quality of microservice-based systems

created through various identification methods can be thoroughly assessed and their differences can then be traced back to extant components of the framework. This holistic evaluation will enable a deeper understanding of the strengths and weaknesses of various identification methods, leading to improved guidelines for their selection.

The current landscape reveals a significant variance in the application of metrics. The lack of an established standard leads to discrepancies, which impede accurate comparisons of approaches' quality. Although structural quality metrics are frequently observed, their use is inconsistent across studies. To address these irregularities, future research endeavors could focus on creating a standardized set of metrics tailored to microservice identification. The development of a comprehensive body of standardized comparative metrics or benchmarks would further enhance the evaluation and comparison of different approaches, contributing to a more precise and reliable assessment of architectural quality. This standardization would facilitate a uniform basis for architectural analyses and allow for more accurate and meaningful conclusions to be drawn from their comparison. Careful analysis of such metrics can inform guidelines for selecting microservice identification methods and approaches. Evaluations of identification methods and other components through these metrics, once mapped onto quality attributes, can guide the selection of approaches based on the criteria are most critical to users of the framework. Such guidelines can enable architects to make more informed decisions, aligning the selection between approaches to architectural design with their specific needs and quality expectations.

One of the objectives of this work was the establishment of a framework to guide quality-driven microservice architecture development. The initial literature review can thus be viewed as a comprehensive compilation of criteria predominantly relevant to microservices, providing context for what to expect in the development of microservice-based systems. However, microservices are no silver bullet, and dogmatically focusing on any particular architectural style could be problematic. The first literature review thus served the dual purpose of facilitating a deeper understanding of microservices by contrasting them with other architectural styles, while it also aimed to broaden the perspective on the selection between alternatives to potentially more suitable alternatives, addressing a gap often found in the literature. To fully incorporate the nuances of identification methods into the comparison, similar analyses of other architectural styles are necessary. As a continuation of this endeavor, it is crucial to delineate microservice identification approaches from those used in SOA.

Given SOA's similarities and its broader, more mature research regarding service identification methods, these methods carry the potential to be directly applicable to microservice identification, making their examination increasingly relevant. By expanding the scope to service identification of service-oriented architecture, approaches to architectural design in its context can be analyzed and compared, aiding in the decision-making process for selecting an architectural style. Identifying and categorizing metrics for architecture development approaches pertaining to alternative architectural styles also enables a more thorough comparison. The incorporation of metric-based quality assessments can then enhance predictions of the suitability of an architectural style in respect to its realization by means of a particular approach. This comprehensive analysis ensures informed choices are made, especially when specific criteria are crucial for the architecture's success.

In summary, a comprehensive guideline would consider alternative styles and their architectural patterns. It would also incorporate approaches to the implementation of a desired architectural style and include suggestions of suitable migration patterns to facilitate the development process. Still, considerations of extant assets and preconditions need to be incorporated as well. As outlined above, the differentiation of pre-existing assets, data collection methods and initial data artifacts needs to be made more precise to achieve this. Combinations of identification methods, their comparative assessment and later stages in the development process further augment the guidelines. Future research should focus on quantifying the comparison between architectural styles to determine the degrees, to which various quality criteria are accommodated by different architectural styles. Semantic analysis of statements in the literature could be performed to classify sentiments about quality criteria in the context of architectural styles. This can serve to quantify the suitability of an architectural style to the optimization of certain quality criteria. Furthermore, the relationships between quality criteria and effects of changes to individual criteria on others should be measured to determine criteria that are interdependent and can be indirectly measured through other quality criteria. This study methodically quantifies the relevance of quality criteria to architectural styles, but the analysis of effect of criteria on other criteria can be represented as a network of criteria represented by nodes and their effects on others as vertices. Such a network could be clustered to identify cohesive groups of criteria, that need to be considered in combination. This can uncover further criteria that practitioners need to be mindful of, when selecting between architectural styles based on their individually desired qualities. It can highlight interdependent quality criteria, that also become significant factors for the selection due to their overall

correlation with a certain subset of criteria, prioritized in the selection process. The framework presented in this study has mapped metrics to quality criteria, that are immediately measured and estimated by them. A methodical evaluation of effects of changes to the quality level of certain criteria on others is still pending and would serve to include indirectly measured quality criteria to allow for quality-driven validation of desired criteria at the design stage. Quantifying, how architectural styles score in respect to certain quality criteria can serve practitioners by showing them, what qualities they can expect from software systems developed in a particular style. Future studies should place a greater emphasis on the pre-existing organizational requirements that significantly influence the selection of architectural styles. These prerequisites could include assessments of the maturity of organizational processes and documentation practices. They should also encompass factors evaluations, such as the size and geographical distribution of organizations, as well as the types and quality of available data. The systematic categorization of these factors is expected to serve as a foundation for the identification of suitable architectural styles and approaches to their architectural design that align with the distinct characteristics of the organization. Understanding these prerequisites can also aid architects in the prioritization of desired qualities and drivers for the transition to a different architectural style, including its realization. A questionnaire can serve as a standardized form of these prerequisites to facilitate their mapping to suitable architectural patterns and development approaches. It can also include individually prioritized quality criteria. The design of said questionnaire can be used as part of a survey form to validate its comprehensiveness and usefulness. The prioritization of quality criteria can be used to refine the quality scores of architectural styles. Aggregating the results from assessments of individual criteria into quality scores can then facilitate the weighted scoring of directly prioritized criteria quality. Considering the interrelations of quality criteria, this weighted score for quality criteria should incorporate quality criteria, that are not prioritized but strongly affected by prioritized criteria, depending on the degree of their correlation. Finally, an aggregated quality score can be computed based on weighted quality scores, to obtain a unified suitability score, by which to compare the appropriateness of alternative architectural styles at a glance. The individual scores and the aggregate score represent a measure of the individual suitability of architectural styles for an organization, facilitating a data-driven and quality-driven approach to architectural decision-making. The resulting framework can guide practitioners along the process of development, including the assessment of relevant preconditions, functional and non-functional requirements, desired quality criteria and development approaches, including

architectural design. The realization of such a decision guide on architectural style and design approach selection concludes the research agenda.

8.3 Implications

This thesis targets practitioners steering their organizations towards complex software architecture. It contributes to the existing body of knowledge by offering a comprehensive exploration into the intricacies of software architecture, bridging theoretical knowledge with practical application, and providing critical insights into significant factors affecting their selection and adoption. In particular, it highlights alternatives to microservices, addressing a pressing challenge in contemporary software engineering and underscoring the importance of choosing architectural styles that are not only technically sound but also strategically advantageous. One can glean an improved understanding of the implications inherent in choosing between microservice architecture and service-oriented architecture, the effort needed to implement microservices, and how to build proper microservices from different preconditions. This research integrates and contrasts the disparate literature on monolithic, service-oriented, and microservice architectures, delivering a comparative analysis that assists in selecting the most appropriate architectural style. It organizes the spectrum of microservice identification approaches, presenting an overview of the state-of-the-art approaches to microservice identification and aiding practitioners in understanding the implications of transitioning to a microservice architecture. Complex architectural styles require and lend themselves to quality-driven approaches. This study helps improve service quality by offering guidelines for creating services from existing assets and applicable metrics to enhance microservice architectural design concerning a multitude of quality criteria. By outlining various options for practitioners and aligning them with the specific conditions of their environments, strategically guiding them through the landscape of microservice identification approaches with a keen eye on the distinct conditions of greenfield and brownfield development projects by means of a taxonomy and a dedicated framework. These equip both researchers and practitioners with a deeper comprehension of architectural development tasks, supporting more informed decision-making regarding such transformations. Typically, business requirements guide the initial selection of components to be exposed as services, but this selection can be further refined by evaluating various quality criteria. This is particularly vital as the operational focus of microservices often emphasizes their technical independence over their alignment with business domains in their architectural design. This thesis is therefore instrumental in enhancing the quality of microservice architectural design, guided by the

identified metrics and guidelines. It empowers software architects to choose the most suitable architectural approach based on predefined criteria, bridging the gap between common bottom-up approaches, often mired in technical constraints, and a strategic top-down view. This broader perspective fosters a holistic understanding that complements the detailed, constraint-driven approach prevalent in the industry, thereby facilitating more informed and effective architectural decisions. Finally, this seminal work lays the groundwork for the research community to further develop and expand upon the research agenda set forth, contributing to the field's progression and the practical application of its findings in software architecture design and implementation.

Appendix

List of Appendix Figures

List of Appendix Tables

A Related Works

A.1 Related Literature Reviews

Primary Topic	Study	Research Questions
Service-oriented Architecture	(Niknejad et al., 2020)	RQ1: What are the primary studies on SOA in the Information Systems domain and which research themes have been addressed in the literature? RQ2: What theoretical frameworks and models are emphasized in current SOA research? RQ3: What are the most influential factors affecting SOA adoption/implementation in organizations? RQ4: What are the gaps, limitations, and future work recommendations in current SOA research?
Service-oriented Architecture	(Joachim, 2011)	RQ1: How should SOA be conceptualized in empirical research? RQ2: What are the factors influencing SOA adoption? RQ3: What is the business value of SOA? RQ4: Which SOA governance mechanisms are important in order to implement an effective SOA? RQ4: Which SOA governance mechanisms are important in order to implement an effective SOA?
Service-oriented Architecture	(Mahdavi-Hezavehi et al., 2013)	RQ1: What quality attributes do existing methods for variability in quality attributes of service-based systems handle? RQ2: What software development activities are addressed by existing methods for handling variability in quality attributes of service-based systems? RQ3: What solution types are used by methods to handle variability in quality attributes of service-based systems? RQ4: What evidence is available to adopt proposed methods for handling variability in quality attributes of service-based systems? RQ5: Are methods only applicable to variability of design-time or run-time quality attributes? RQ6: Is there support for practitioners concerning how to use current methods?
Service-oriented Architecture & Microservice Architecture & Quality Criteria	(Bogner et al., 2017)	RQ: What is a feasible maintainability QM for SBSs and µSBSs with a focus on automatic measurements, simplicity, and practical Applicability?
Microservice Architecture	(Di Francesco et al., 2017)	RQ1: What are the publication trends of research studies about architecting microservices? RQ2: What is the focus of research on architecting microservices? RQ3: What is the potential for industrial adoption of existing research on architecting microservices?
Microservice Architecture	(Di Francesco et al., 2019)	RQ1: What are the publication trends of research studies about architecting with microservices? RQ2: What is the focus of research on architecting with microservices?

Primary Topic	Study	Research Questions
		RQ3: What is the potential for industrial adoption of existing research on architecting with microservices?
Microservice Architecture	(Pahl & Jamshidi, 2016)	RQ1: What are the main practical motivations behind using microservices? RQ: What are the different types of microservice architectures involved? RQ3: What are the existing methods, techniques, and tool support to enable microservice architecture development and operation? RQ4: What are the existing research issues and what should be the future research agenda?
Microservice Architecture & Quality Criteria	(Alshuqayran et al., 2016)	RQ1: What are the architectural challenges that microservices systems face? RQ2: What architectural diagrams/views are used to represent microservices architectures? RQ3: What quality attributes related to microservices are presented in the literature?
Microservice Architecture & Quality Criteria	(S. Li et al., 2021)	RQ1: What are the most concerned QAs for MSA? RQ2: What tactics have been proposed or discussed to improve the most concerned QAs of MSA?

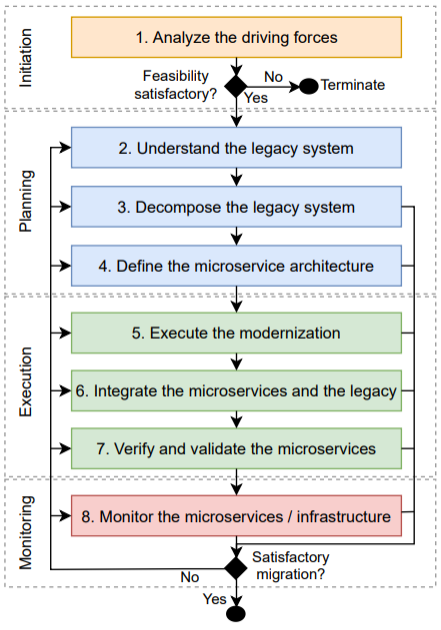
Appendix Table A.1: Literature reviews on SOA, MSA, and quality criteria in their context

Primary Topic	Study	Research Questions
Service Identification in Service-oriented Architecture	(Abdellatif et al., 2021)	RQ1: What are the inputs used by SIAs? RQ2: What are the processes followed by SIAs? RQ3: What are the outputs of SIAs? RQ4: What is the usability of SIAs?
Service Identification in Service-oriented Architecture	(Abdellatif et al., 2018)	RQ1: What kind of systems are being migrated to SOA? RQ2: Why are such systems being migrated? RQ3: What approaches are being used for application migration, in general, and SI in particular?
Microservice Identification	(Velepucha & Flores, 2021)	RQ: What problems and challenges are there for the migration process of monolithic applications to microservices?
Microservice Identification	(Fritzsch et al., 2019)	RQ: What are existing architectural refactoring approaches in the context of decomposing a monolithic application architecture into Microservices and how can they be classified with regard to the techniques and strategies used?
Microservice Identification	(Wolfart et al., 2021)	RQ: Why and how are monolithic legacy systems migrated to microservices?
Microservice Identification	(Oumoussa & Saidi, 2024)	RQ1: How has the area of study on microservices identification evolved? RQ2: What is the current state-of-the-art in microservices identification research? RQ3: What are the current and potential challenges associated with microservices identification?
Microservice Identification	(Ponce et al., 2019)	RQ1: What are the migration techniques proposed in the literature? RQ2: In what types of systems have the proposed techniques been applied? RQ3: What type of validation do the authors of the techniques use? RQ4: Are there challenges associated with migration from monolith to microservices?
Microservice Identification	(Abgaz et al., 2023)	RQ1: What are the primary phases of monolith-to-microservices decomposition and the major constituent elements of those phases?

Primary Topic	Study	Research Questions
		RQ2: What are the existing approaches, tools and methods observed in the decomposition of monolith applications into microservices? RQ3: What are the metrics, datasets, and benchmarks used for evaluating and validating monolith decomposition into microservices? RQ4: What research gaps can be identified in the current literature?
Microservice Identification & Quality Criteria	(Schröer et al., 2020)	RQ: Which approaches do exist to support the identification of microservices along the software development process considering several criteria? RQ1: Which starting points can be used for microservices identification approaches during requirements analysis? RQ2: Which microservices identification approaches do exist at design phase? RQ3: How can identified microservices be evaluated?
Microservice Identification & Quality Criteria	(Capuano & Muccini, 2022)	RQ1: Which studies implement a quality-driven approach to migrate to microservices? RQ2: Which are the quality attributes analyzed in the migration phases? RQ3: In which migration phase the quality-driven process is implemented?
Microservice Identification & Quality Criteria	(Cojocaru et al., 2019)	None

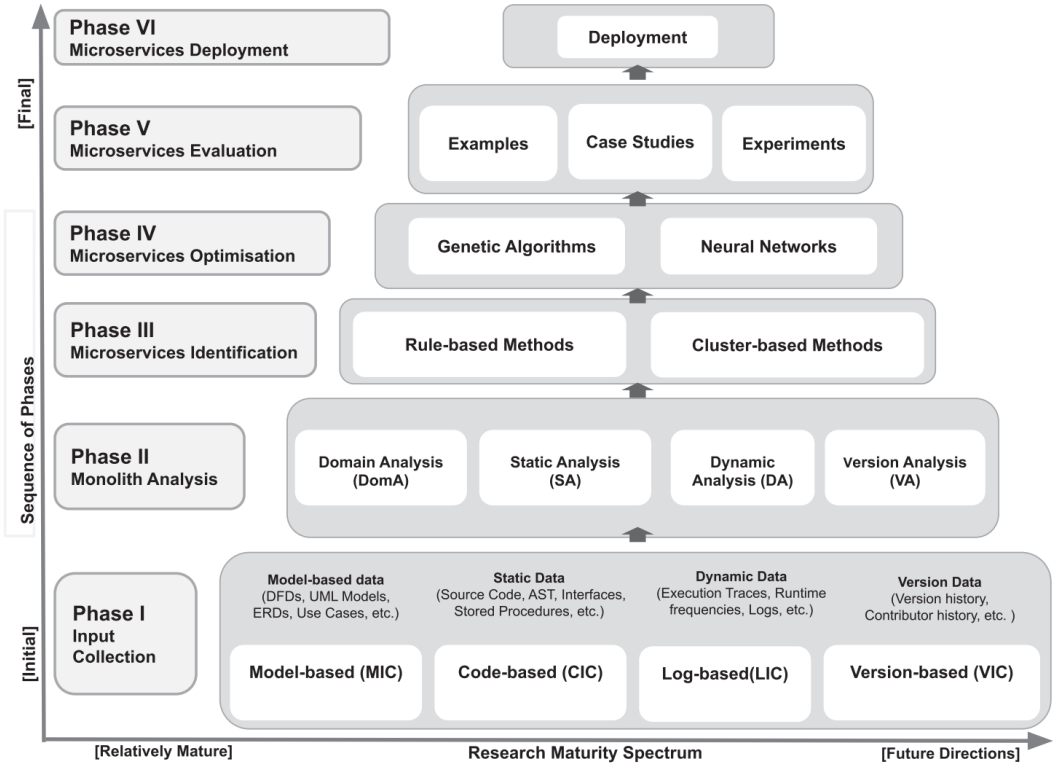
Appendix Table A.2: Literature reviews on microservice identification

A.2 Related Roadmap and Framework



(Wolfart et al., 2021, p. 151)

Appendix Figure A.1: Roadmap for modernizing legacy systems with microservices

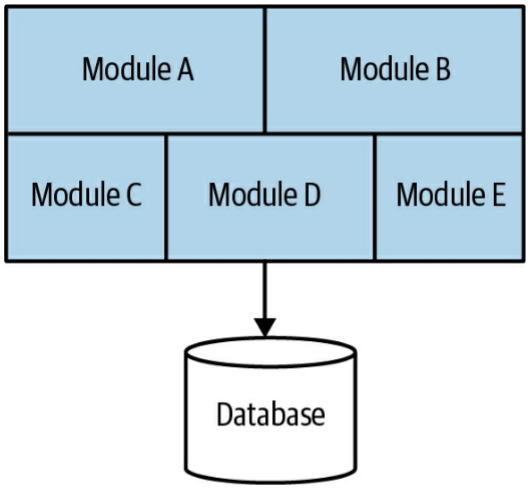


(Abgaz et al., 2023, p. 4230)

Appendix Figure A.2: Monolith to microservices decomposition framework

B Theoretical Background

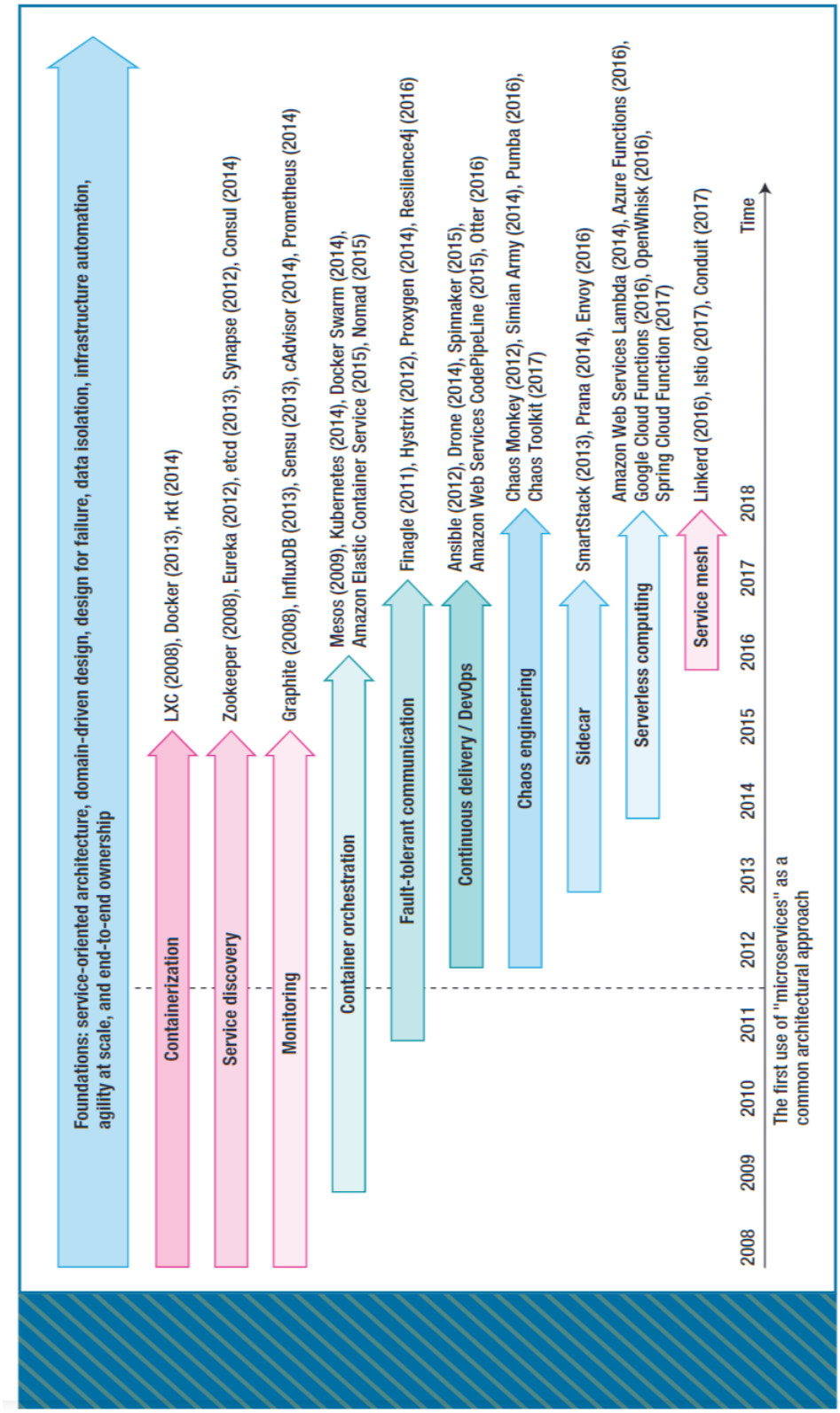
B.1 Monolithic Architecture



(Newman, 2021, pp. 16–17)

Appendix Figure B.1: Modular monolithic architecture

B.2 Microservice Architecture

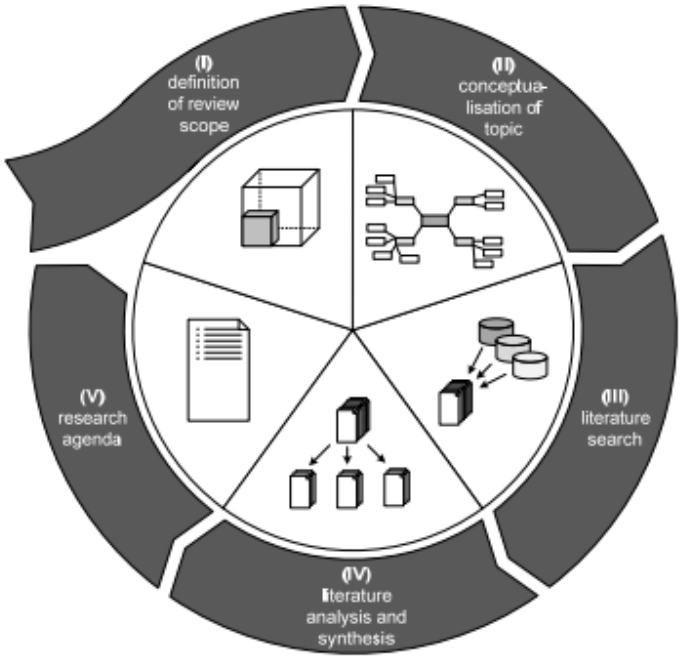


(Jamshidi et al., 2018, p. 26)

Appendix Figure B.2: A microservice technologies timeline

C Research Approach

C.1 Methodology



(vom Brocke et al., 2009, p. 8)

Appendix Figure C.1: Framework for literature reviewing

C.2 Literature Review Taxonomy

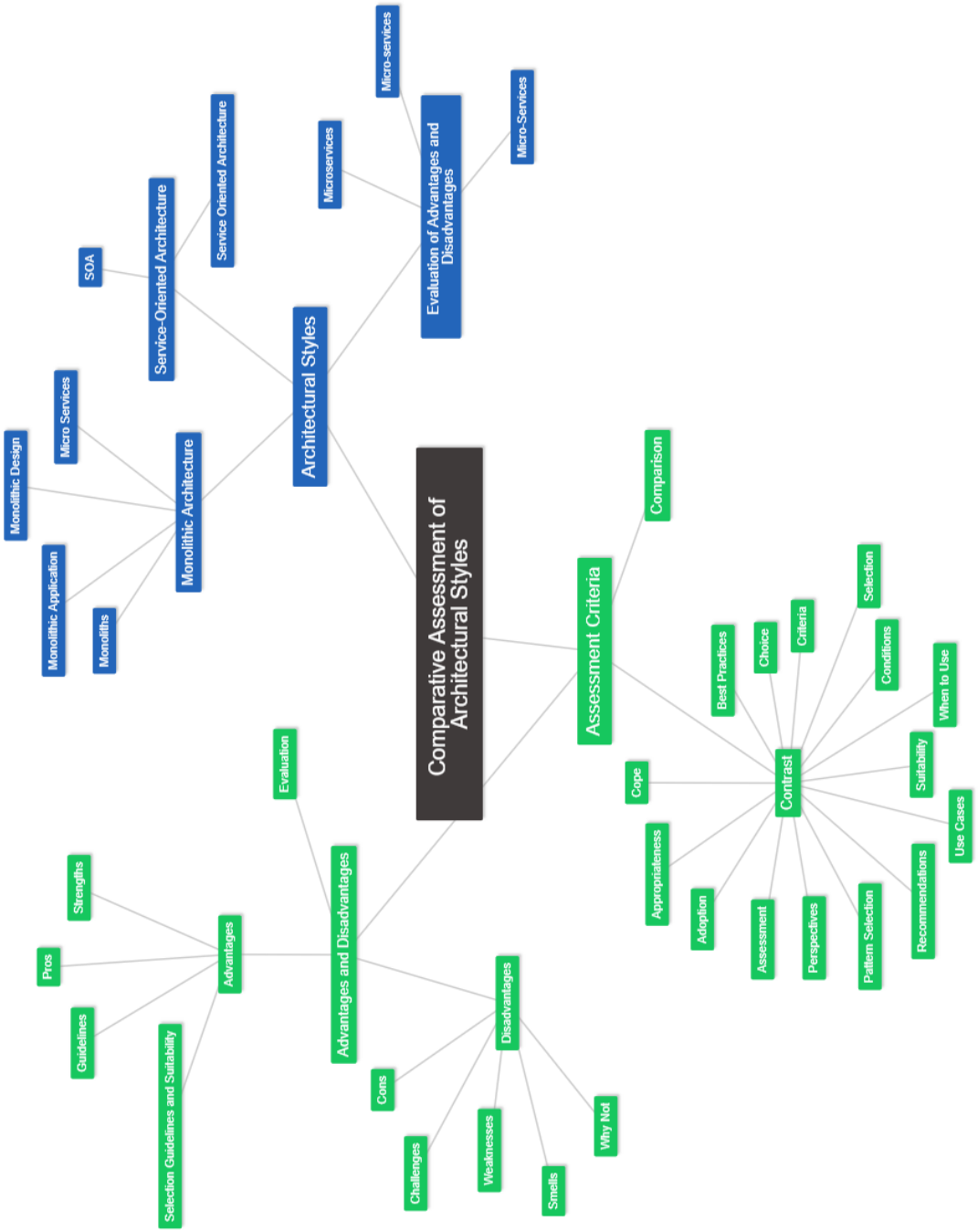
Table 1
A Taxonomy of Literature Reviews

Characteristic	Categories
Focus	Research Outcomes Research Methods Theories Practices or Applications
Goal	Integration a) Generalization b) Conflict Resolution c) Linguistic Bridge-building Criticism Identification of Central Issues
Perspective	Neutral Representation Espousal of Position
Coverage	Exhaustive Exhaustive with Selective Citation Representative Central or Pivotal
Organization	Historical Conceptual Methodological
Audience	Specialized Scholars General Scholars Practitioners or Policy Makers General Public

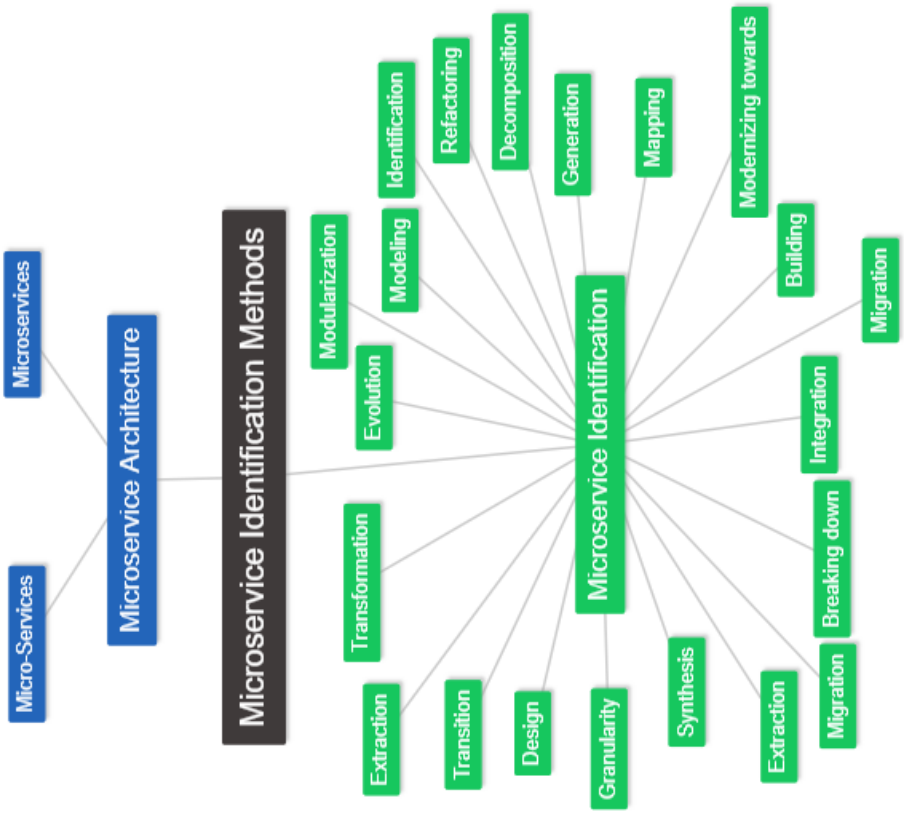
(Cooper, 1988, p. 109)

Appendix Figure C.2: Cooper's taxonomy of literature reviews

C.3 Concept Maps

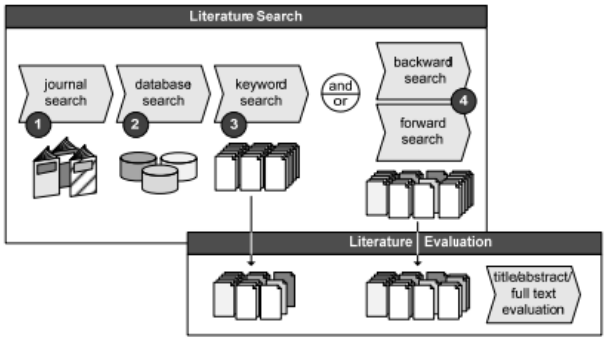


Appendix Figure C.3: Concept map for the literature review on architectural styles



Appendix Figure C.4: Concept map for the literature review on microservice identification

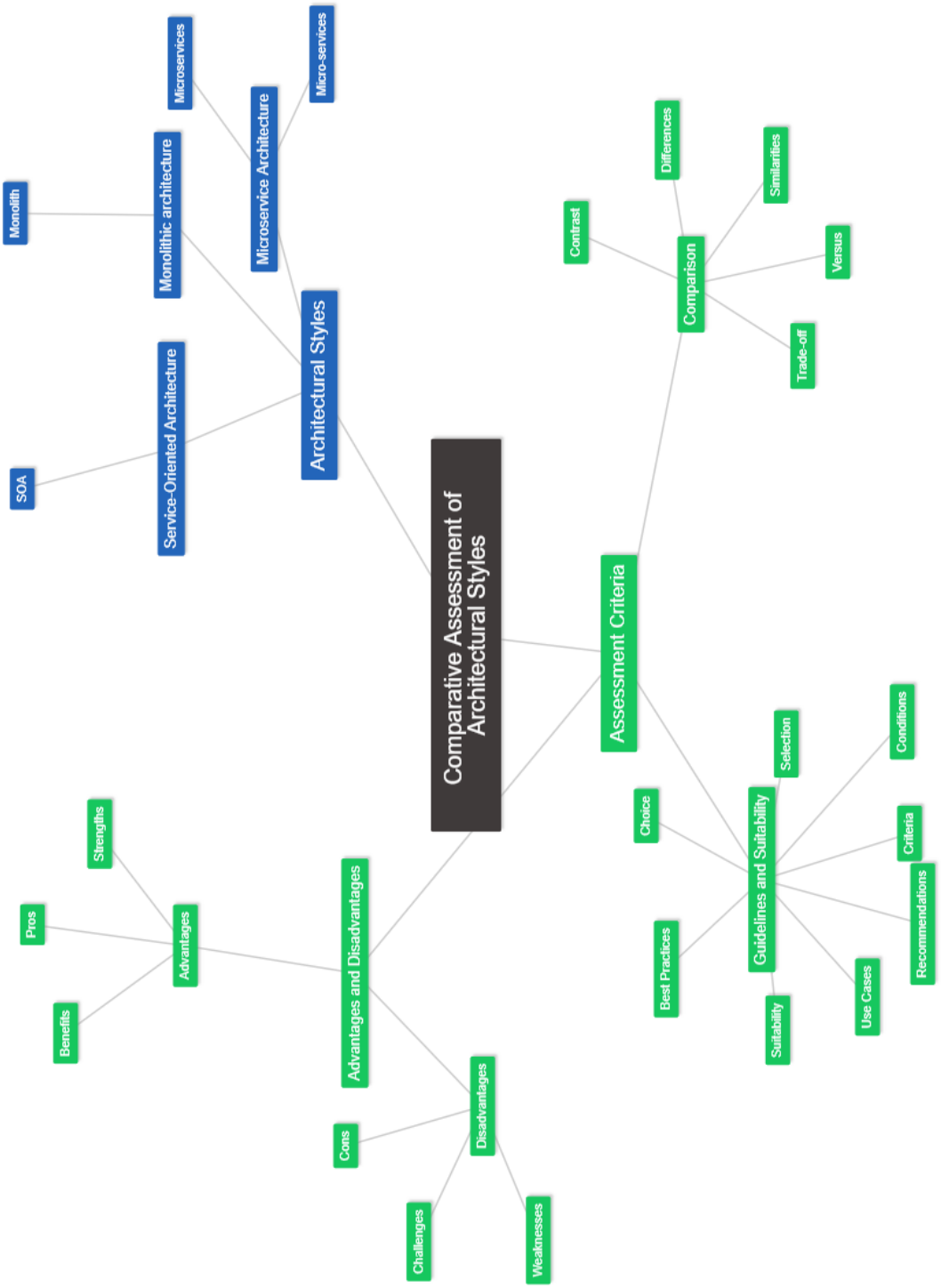
C.4 Literature Search Process



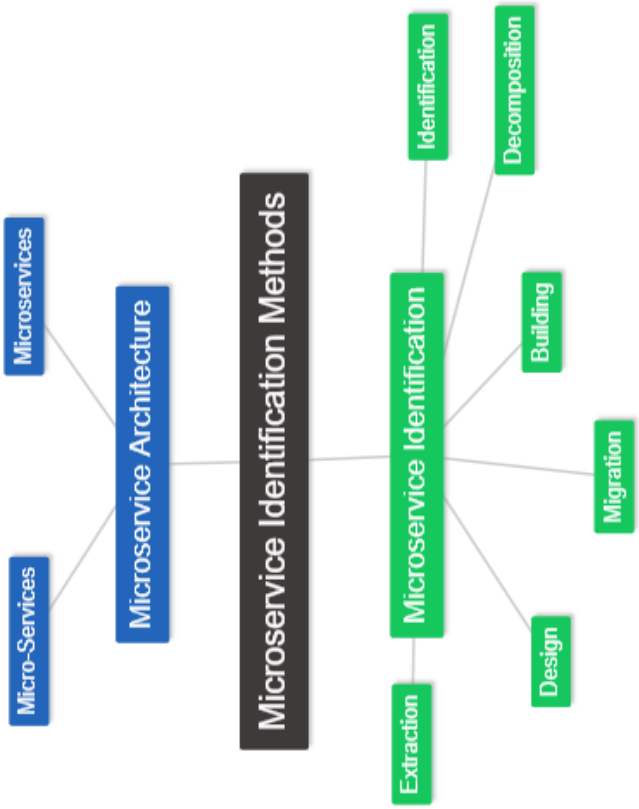
(vom Brocke et al., 2009, p. 10)

Appendix Figure C.5: Literature search process

C.5 Search Strings



Appendix Figure C.6: Concept map of search strings for the literature review on architectural styles



Appendix Figure C.7: Concept map of search strings for the microservice identification review

C.6 Concept Matrix

Table 3. Concept Matrix Augmented with Units of Analysis

Articles	Concepts														
	A			B			C			D			…		
Unit of analysis	O	G	I	O	G	I	O	G	I	O	G	I	O	G	I
1					✖				✖						✖
2	✖				✖	✖		✖							
…								✖	✖			✖			

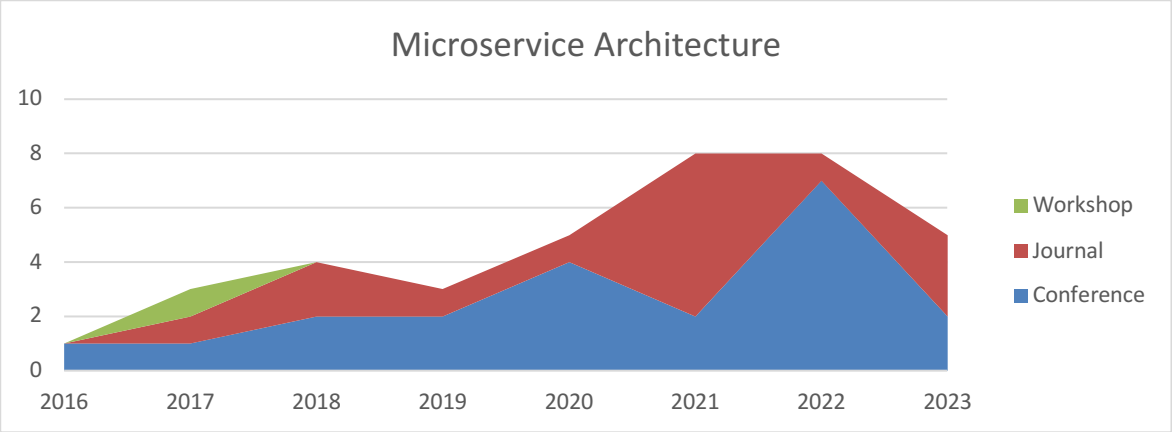
Legend: O (organizational), G (group), I (individual)

(Webster & Watson, 2002, xvii)

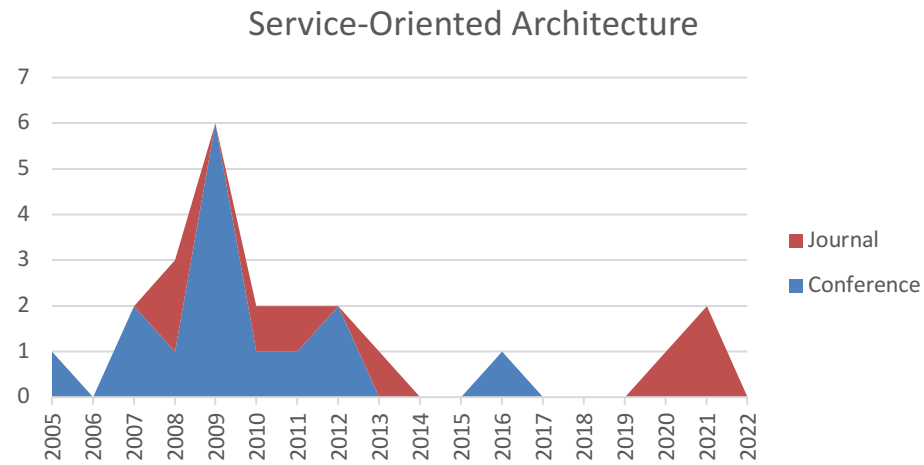
Appendix Figure C.8: Concept matrix

D Architectural Styles

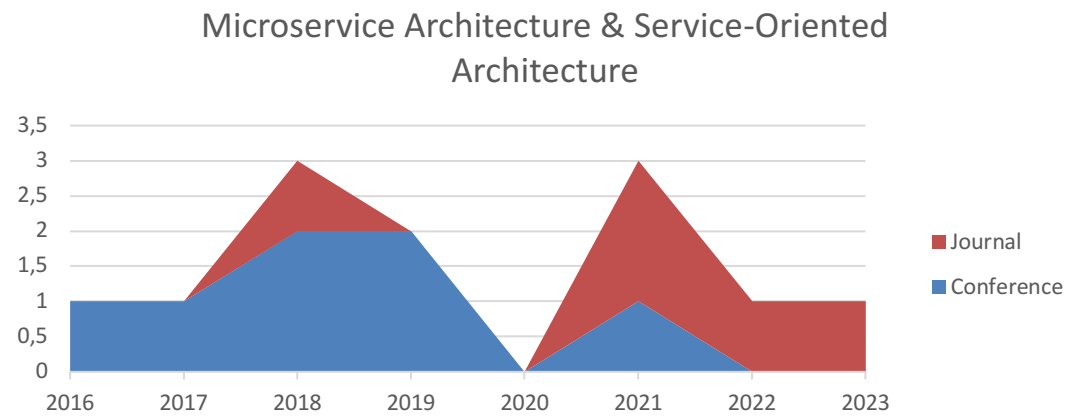
D.1 Overview



Appendix Figure D.1: Temporal distribution and publication venues of MSA



Appendix Figure D.2: Temporal distribution and publication venues of SOA



Appendix Figure D.3: Temporal distribution and publication venues of SOA & MSA

D.2 Concept Matrix

	Archi-tectural Style		Qualitative Criteria																				
Study	Microservices	SOA	Monolith	Cost	Deployment	Development	Governance	Granularity	Interoperability	Integration	Analyzability	Maintainability	Modifiability	Modularity	Technology Hetero-geneity	Organization	IT Business Alignment	Performance	Portability	Reusability	Reliability	Scalability	Security
(Ilk et al., 2010)		x		x		x	x					x		x			x			x			
(Pulparambil et al., 2021)		x						x								x							
(Zhang & Tan-niru, 2005)		x						x	x					x					x	x		x	
(Hustad & Olsen, 2021)		x		x	x	x	x		x	x			x	x	x	x	x		x	x		x	
(Beimborn et al., 2008)		x				x		x	x	x				x	x	x	x			x			
(Boh & Yellin, 2010)		x		x		x			x	x				x		x	x						
(Reddy et al., 2009)		x						x	x					x			x		x	x		x	
(Joachim et al., 2009)		x		x		x		x								x	x			x			x
(Aggelopoulou & Pramatari, 2009)		x		x		x			x	x		x			x	x		x		x			
(MacLennan & van Belle, 2012)		x			x		x		x	x		x				x				x			x
(Antikainen & Pekkola, 2008)		x					x										x			x			
(Schramm & Daneva, 2016)		x				x		x					x	x	x		x	x				x	x
(Simanta et al., 2009)		x							x				x	x									x
(Beydoun et al., 2013)		x		x					x	x	x						x			x			
(White et al., 2012)		x				x			x			x											
(Niknejad et al., 2020)		x		x					x	x	x			x		x	x			x			
(Becker et al., 2011)		x		x		x		x	x	x	x	x	x	x	x	x	x			x			
(Y.-H. Wang & Liao, 2009)		x				x			x	x		x		x		x	x			x			
(Phan, 2007)		x					x		x	x			x	x									x
(Xin Zhou et al., 2022)	x			x		x					x			x							x		
(Camilli et al., 2023)	x							x				x		x					x		x	x	
(Christoforou et al., 2022)	x				x	x			x			x		x				x	x	x	x	x	x

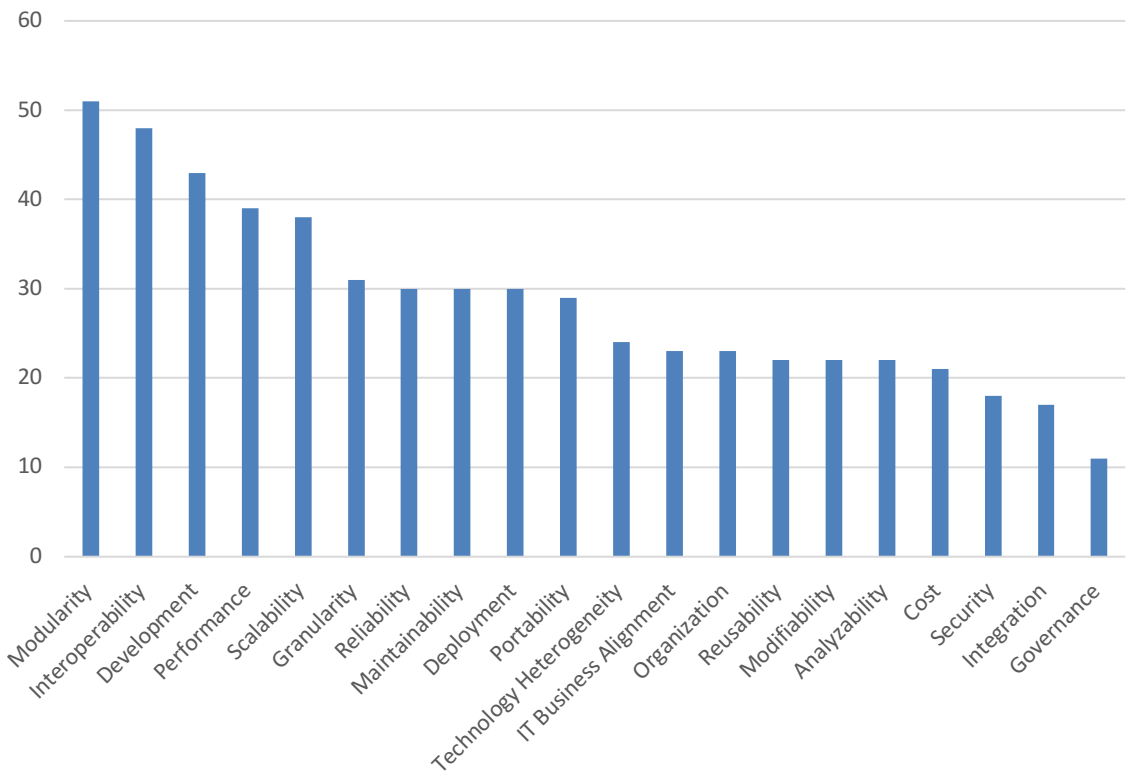
	Archi-tectural Style		Qualitative Criteria																					
Study	Microservices	SOA	Monolith	Cost	Deployment	Development	Governance	Granularity	Interoperability	Integration	Analyzability	Maintainability	Modifiability	Modularity	Technology Hetero-geneity	Organization	IT Business Alignment	Performance	Portability	Reusability	Reliability	Scalability	Security
(Abdelfattah & Cerny, 2022)	x								x														x
(Salii et al., 2023)	x				x	x						x		x		x		x			x	x	
(Di Francesco et al., 2017)	x							x			x	x											
(Billawa et al., 2022)	x				x	x			x		x	x		x									x
(Aksakalli et al., 2021)	x				x	x		x	x	x				x				x	x		x		
(Di Francesco et al., 2019)	x								x			x		x							x		x
(Lira et al., 2023)	x				x				x				x					x	x	x	x	x	
(Waseem et al., 2020)	x					x			x					x				x					x
(Xin Zhou et al., 2023)	x				x	x	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x	
(Al-Debagy & Martinek, 2018)	x	x			x	x		x	x			x		x	x		x	x			x	x	
(Kleftakis et al., 2022)	x	x				x		x	x			x	x	x				x	x			x	
(Oliveira Rosa et al., 2020)	x	x				x		x	x					x	x	x		x		x	x	x	x
(Niño-Martínez et al., 2022)	x	x			x	x						x		x	x	x		x	x		x	x	
(Papakonstan-tinou et al., 2020)	x	x	x						x									x			x	x	
(Gan et al., 2019)	x	x			x	x		x			x		x	x	x			x	x		x	x	
(Götz et al., 2018)	x	x				x		x	x					x					x			x	
(Benavente et al., 2022)	x	x	x	x	x			x	x	x	x	x	x	x	x			x			x	x	x
(Villamizar et al., 2017)	x	x	x	x	x											x		x	x			x	
(Laigner et al., 2021)	x	x			x	x							x	x	x						x		
(Trihinas et al., 2018)	x	x			x		x	x	x	x								x	x			x	x
(Jatkiewicz & Okrój, 2023)	x	x																x				x	
(Xiang Zhou et al., 2021)	x	x							x		x							x			x		
(Ding & Zhang, 2022)	x	x				x			x					x	x		x		x				
(Villamizar et al., 2016)	x	x	x	x														x	x			x	

	Archi-tectural Style			Qualitative Criteria																				
Study	Microservices	SOA	Monolith	Cost	Deployment	Development	Governance	Granularity	Interoperability	Integration	Analyzability	Maintainability	Modifiability	Modularity	Technology Hetero-geneity	Organization	IT Business Alignment	Performance	Portability	Reusability	Reliability	Scalability	Security
(Taibi et al., 2017)	x		x						x			x	x	x		x		x						
(G. Liu et al., 2020)	x		x	x	x	x			x		x			x				x			x			
(Jagiełło et al., 2019)	x		x			x												x			x	x		
(Y. Wang et al., 2021)	x		x	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x		x	x	x	x
(Raharjo et al., 2022)	x		x															x			x			
(Gos & Zabierowski, 2020)	x		x		x			x			x	x		x				x	x		x	x		
(Gravanis et al., 2021)	x		x		x	x					x	x	x	x	x			x	x		x	x		
(Auer et al., 2021)	x		x	x	x	x		x	x		x	x		x	x	x		x	x	x	x	x		
(Mendonça et al., 2021)	x		x	x	x	x			x			x		x	x	x		x	x			x		
(Waseem et al., 2021)	x		x		x	x			x		x	x		x	x		x	x	x		x	x	x	x
(Raj & Sadam, 2021)	x	x												x				x		x		x		
(Xiao et al., 2016)	x	x				x	x	x	x					x	x				x	x				
(Raj & Ravichandra, 2018)	x	x			x	x		x	x	x	x		x	x			x		x	x		x		
(Joachim, 2011)		x	x	x		x	x	x	x	x		x				x	x			x		x		
(Leonard, 2009)		x	x											x										
(Erickson & Siau, 2008)		x	x					x	x					x										
(Mahmood, 2007)		x	x			x	x		x				x	x			x	x			x			x
(Andriyanto et al., 2019)	x	x	x		x	x		x	x			x	x	x		x		x	x		x	x		
(Sorgalla et al., 2021)	x	x	x	x		x		x	x		x	x	x	x	x	x		x	x		x	x		
(Baškarada et al., 2020)	x	x	x		x	x		x	x	x	x	x	x	x	x	x	x	x	x		x	x	x	x
(Cerny et al., 2017)	x	x	x		x				x				x	x	x		x							
(Mangwani et al., 2023)	x	x	x	x	x	x					x		x	x				x			x	x		
(Munaf et al., 2019)	x	x	x		x	x	x	x	x			x		x	x							x		
(Blinowski et al., 2022)	x	x	x			x		x	x		x		x	x	x		x	x	x		x	x		

	Archi-tectural Style			Qualitative Criteria																				
Study	Microservices	SOA	Monolith	Cost	Deployment	Development	Governance	Granularity	Interoperability	Integration	Analyzability	Maintainability	Modifiability	Modularity	Technology Hetero-geneity	Organization	IT Business Alignment	Performance	Portability	Reusability	Reliability	Scalability	Security
(Yarygina & Bagge, 2018)	x	x	x					x			x	x						x	x				x
(Weerasinghe & Perera, 2021)	x	x	x	x	x													x				x	

Appendix Table D.1: Concept matrix of architectural styles and quality criteria

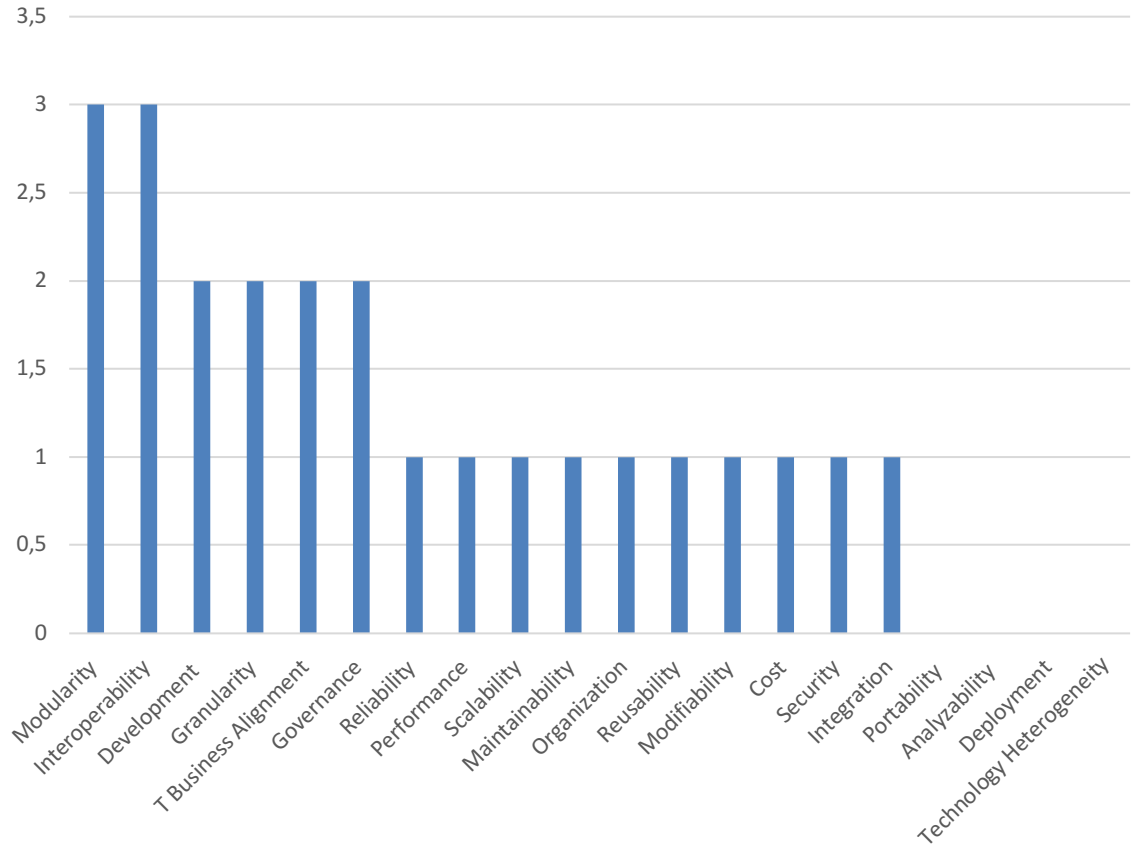
D.3 Quality Criteria



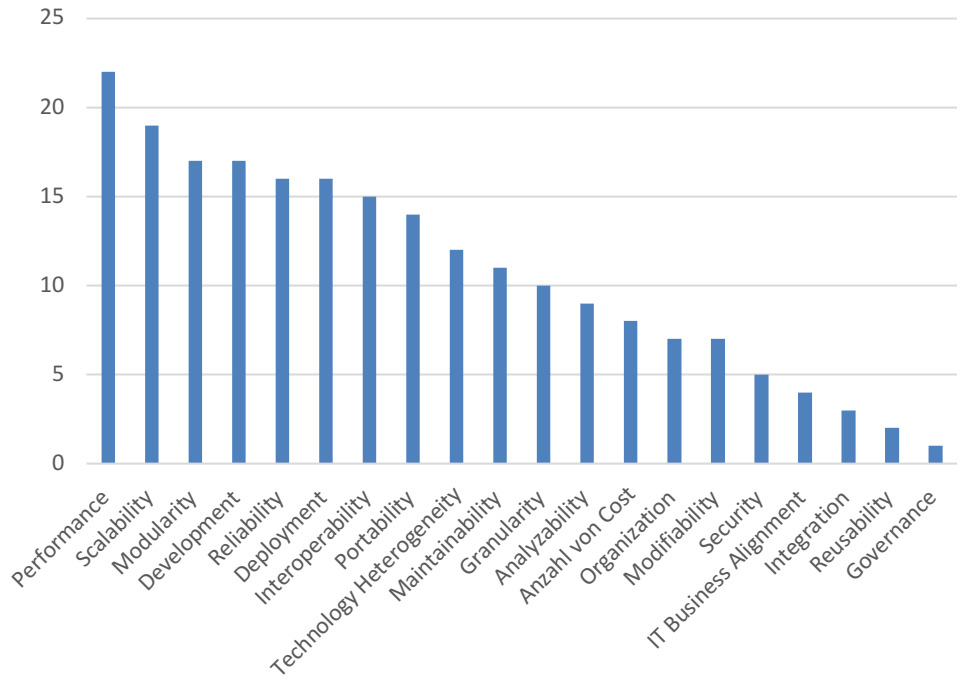
Appendix Figure D.4: Observation frequency of quality criteria across all incorporated studies

In the context of 72 total studies, the chart in Appendix Figure D.4 indicates that modularity, Interoperability, development, performance, and Salability are discussed in more than 50% of the studies, placing them in the set of most frequently concerned quality criteria. granularity, reliability, maintainability, deployment, portability, technology heterogeneity, it business alignment, organization, reusability, modifiability, analyzability, cost, security, and

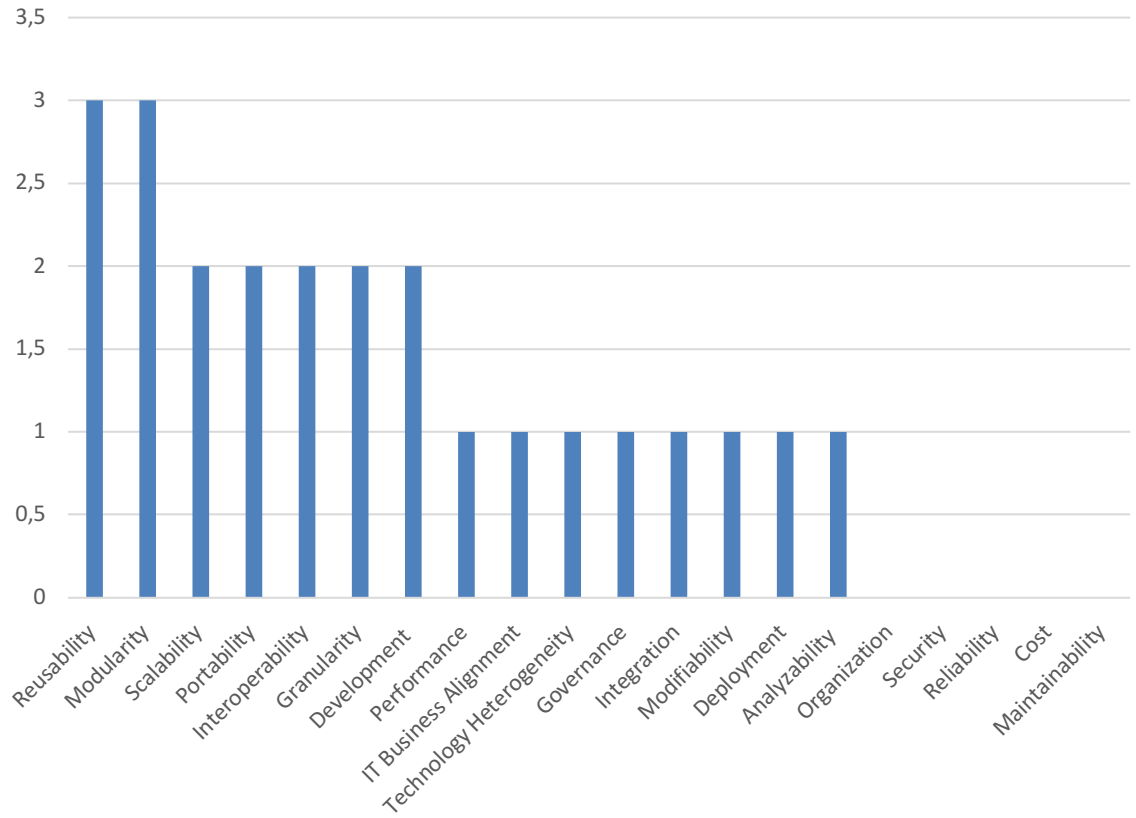
integration are also significant, appearing in more than 20% but less than 50% of the studies. Conversely, the governance criterion is mentioned in fewer than 20% of the studies, suggesting a lower emphasis in the surveyed research.



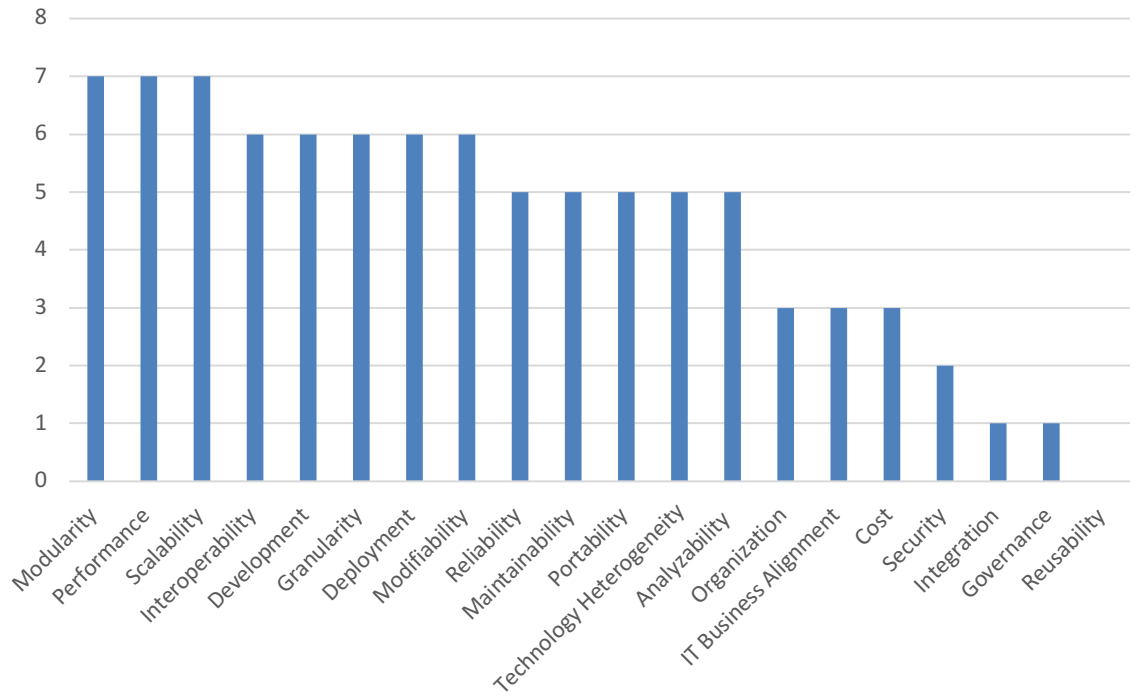
Appendix Figure D.5: Observation frequency of quality criteria in the domain SOA, monolith



Appendix Figure D.6: Observation frequency of quality criteria in the domain MSA, monolith

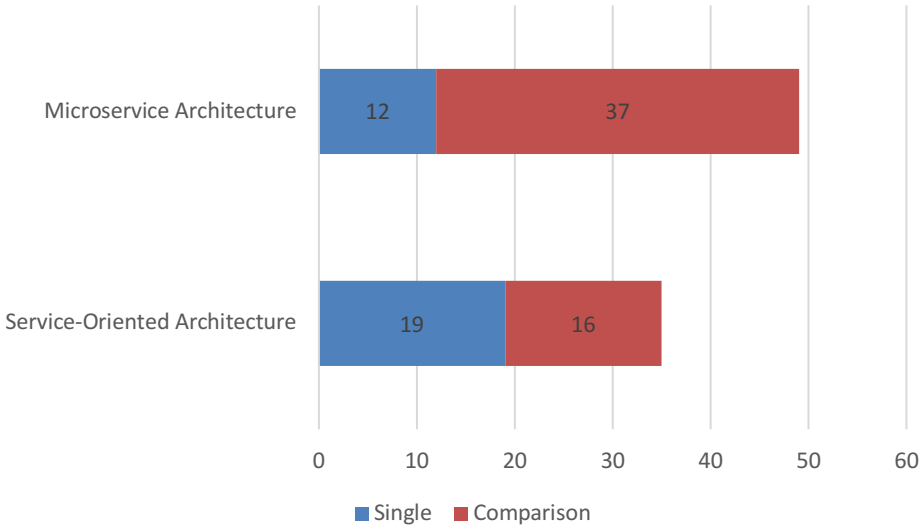


Appendix Figure D.7: Observation frequency of quality criteria in the MSA, SOA domain



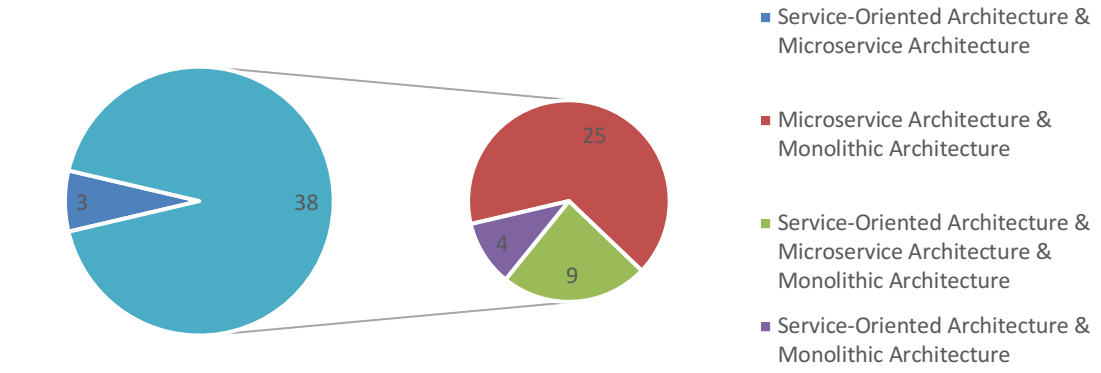
Appendix Figure D.8: Observation frequency of quality criteria in the MSA, SOA, monolith domain

D.4 Comparison by Architectural Styles



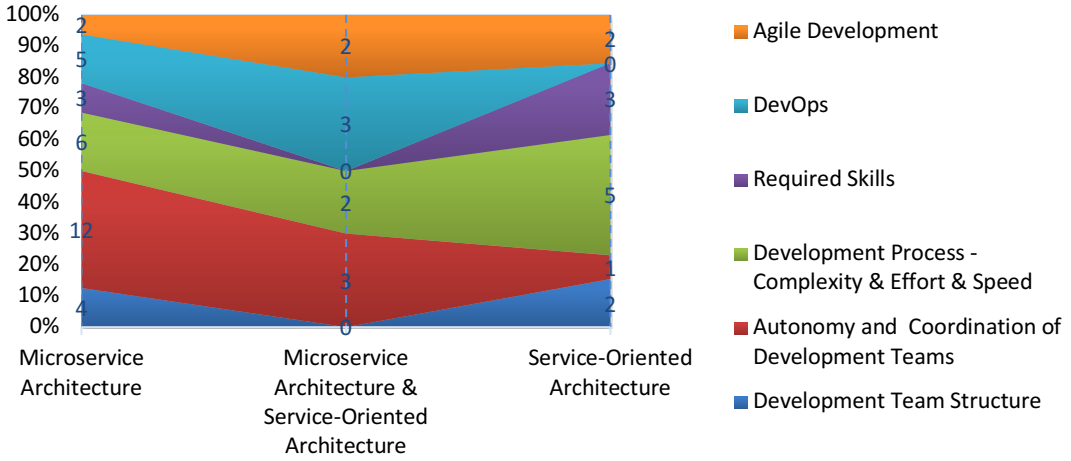
Appendix Figure D.9: Distribution of studies on SOA and MSA in isolation versus comparison

Appendix Figure D.9 presents the distribution of studies on MSA and SOA, comparing between studies, that discuss each architectural style in isolation of others, versus those comparing it with the other architectural style and/or with monolithic architecture.

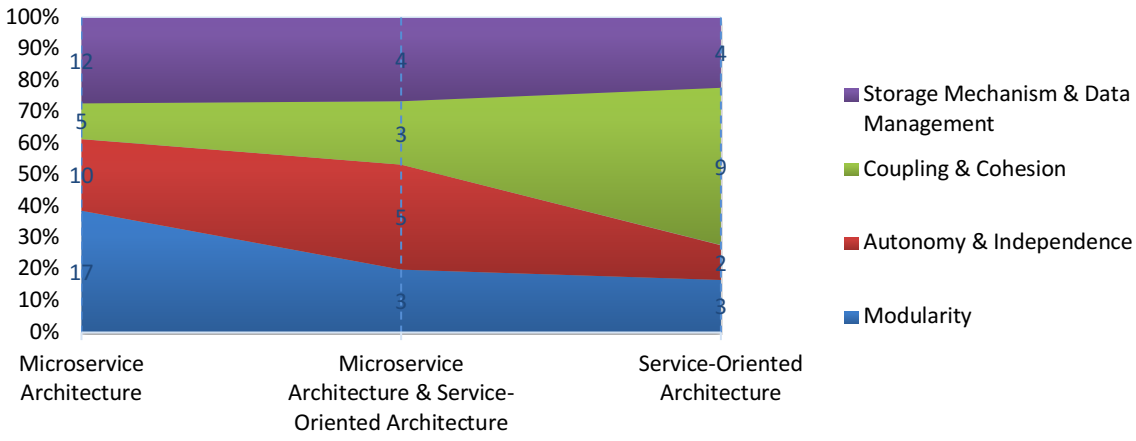


Appendix Figure D.10: Distribution of comparative studies involving monolithic architecture

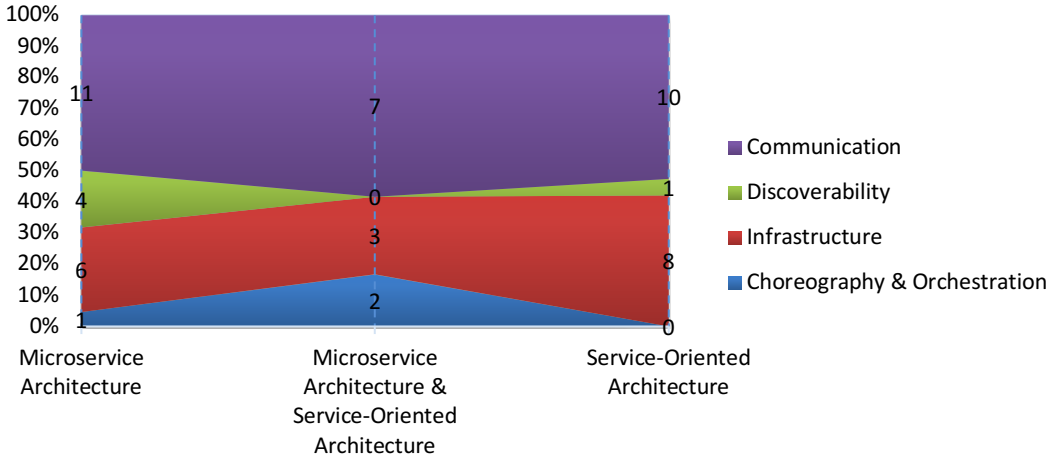
D.5 Comparison by Quality Criteria



Appendix Figure D.11: Distribution of development sub-criteria



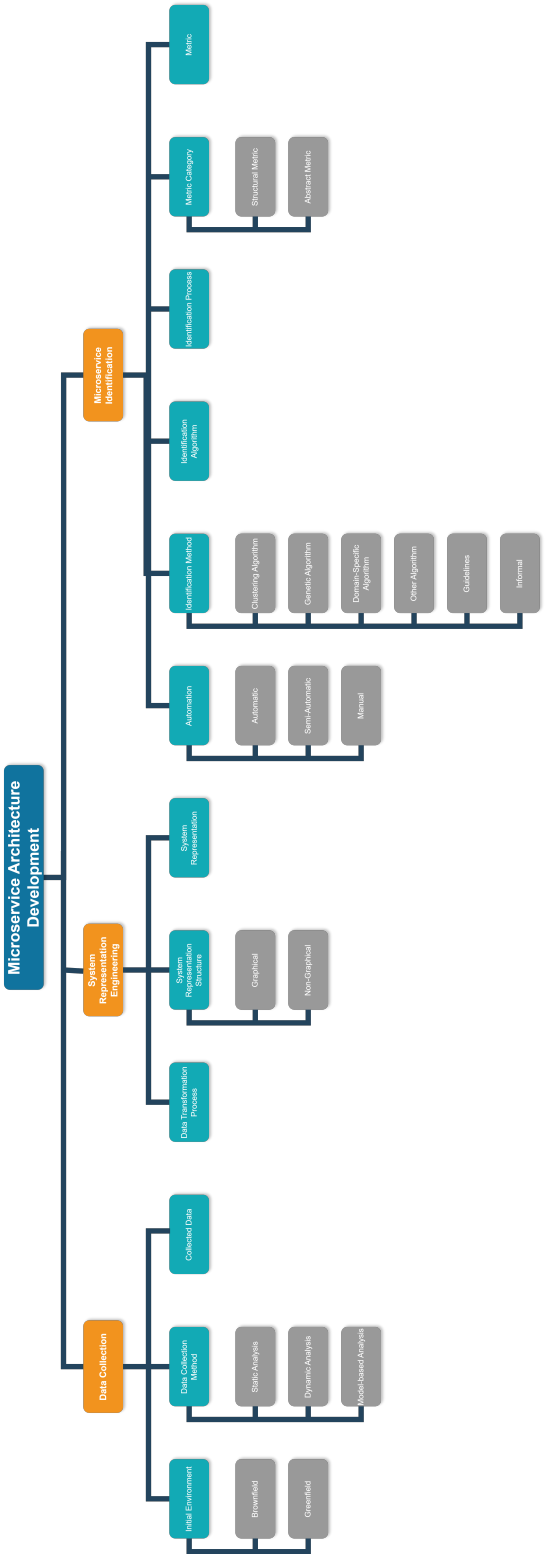
Appendix Figure D.12: Distribution of modularity sub-criteria



Appendix Figure D.13: Distribution of interoperability sub-criteria

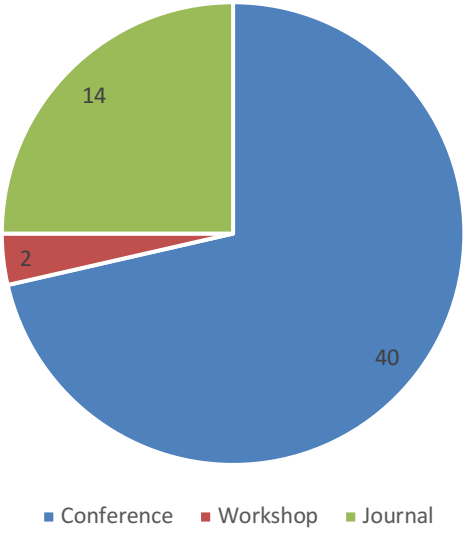
E Microservice Identification

E.1 Augmented Taxonomy



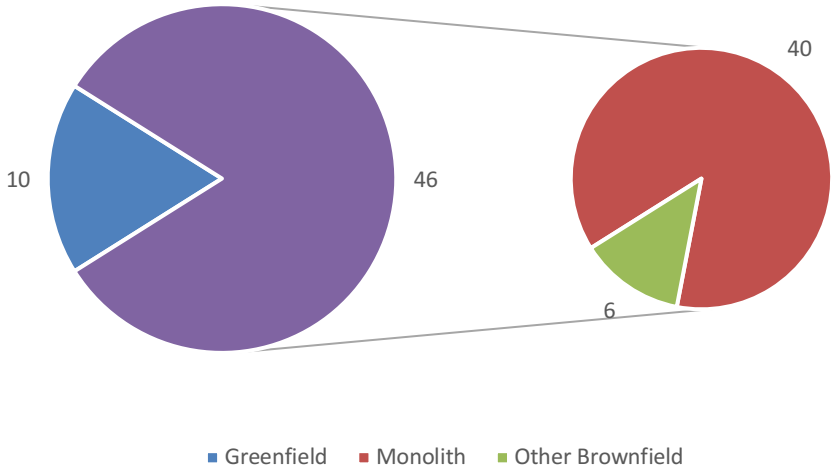
Appendix Figure E.1: Augmented taxonomy of microservice identification approaches

E.2 Overview

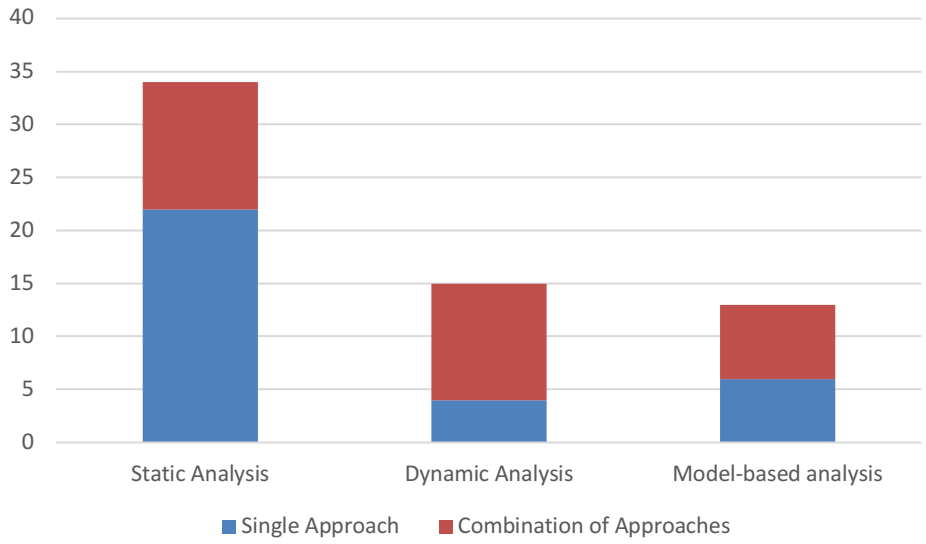


Appendix Figure E.2: Distribution of publication venues

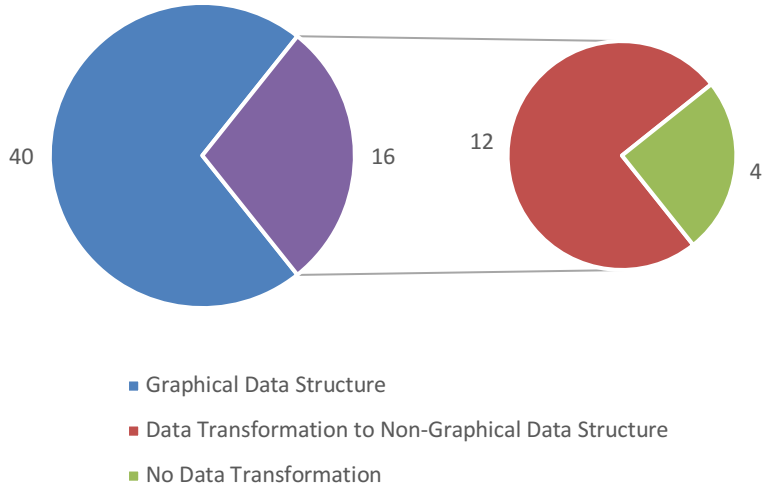
E.3 Analysis



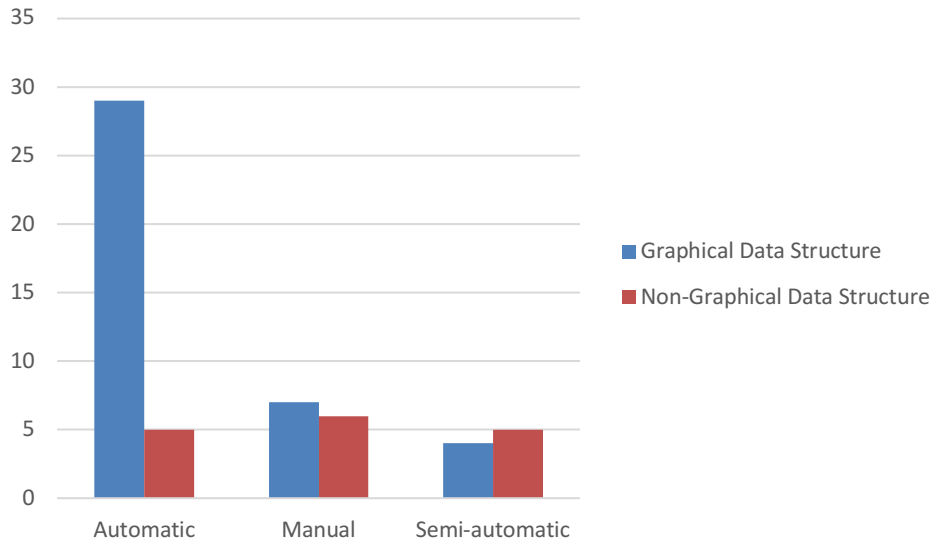
Appendix Figure E.3: Initial environments of microservice identification approaches



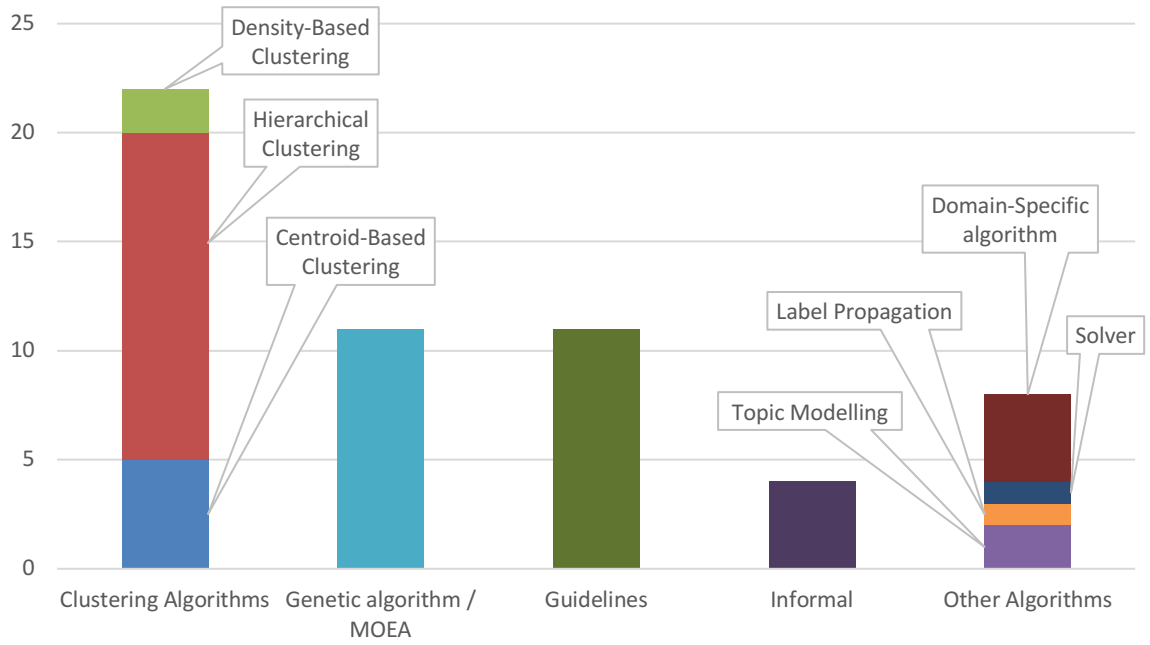
Appendix Figure E.4: Application of brownfield data collection methods as a standalone approach



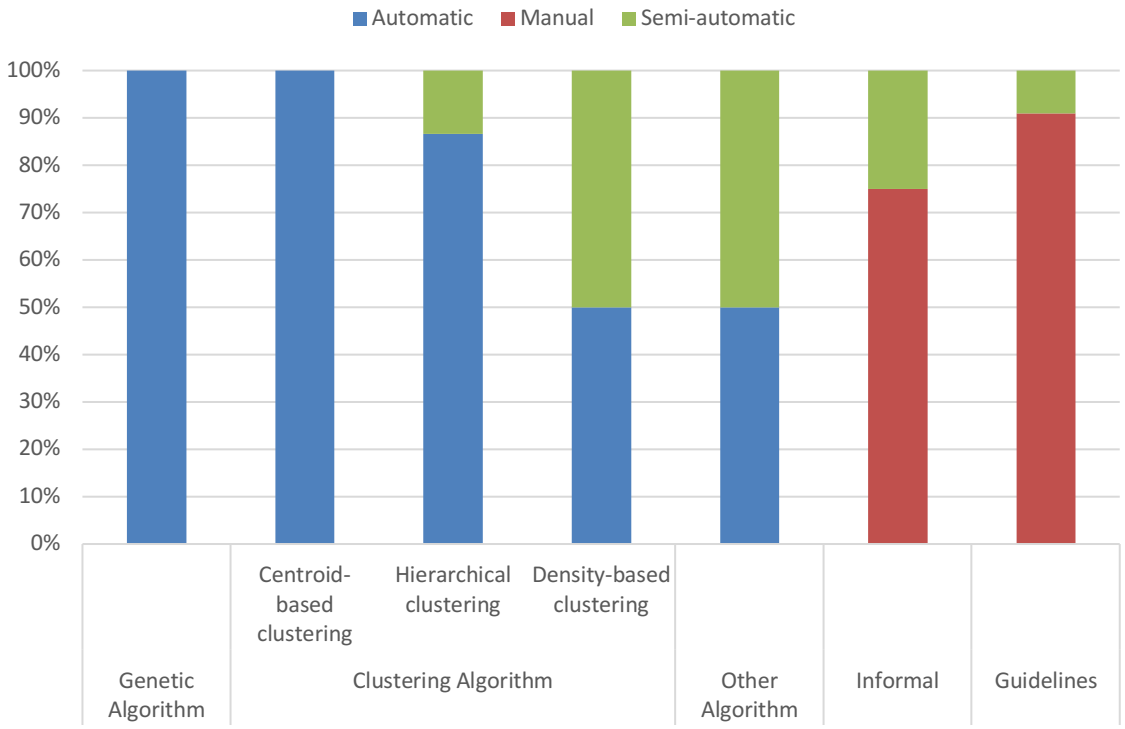
Appendix Figure E.5: Prevalence of data transformation to graphical data structures



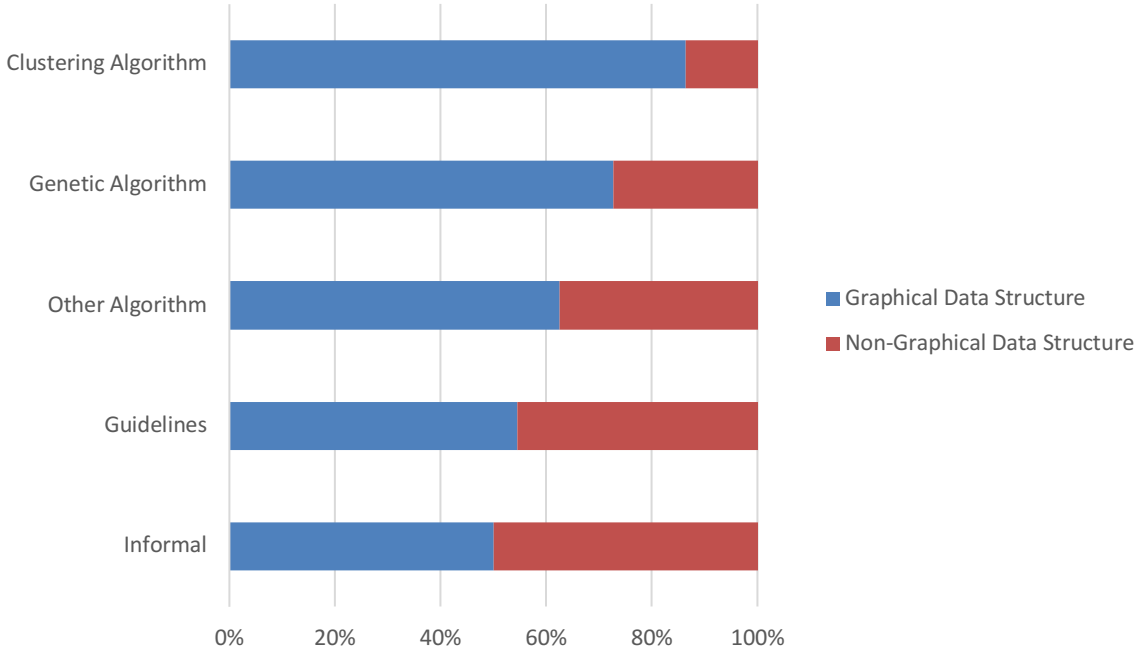
Appendix Figure E.6: Use of graphical system representations across different degrees of automation



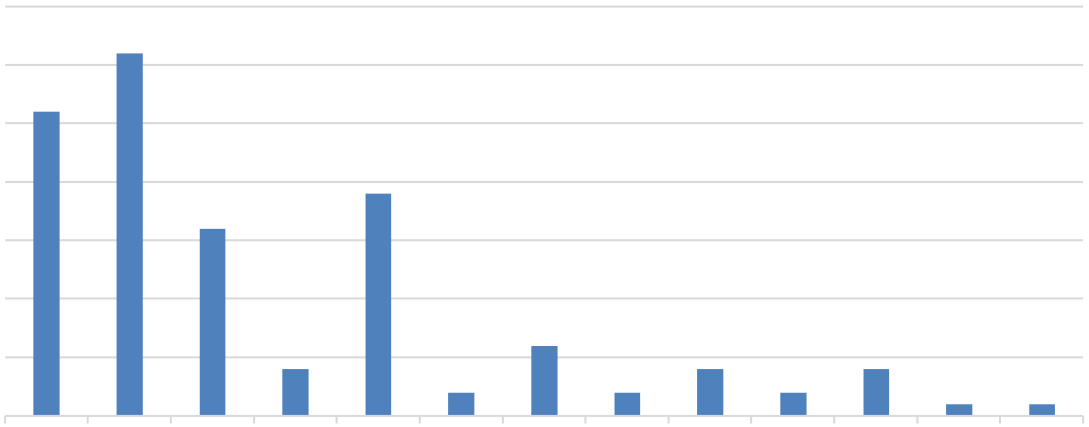
Appendix Figure E.7: Overall prevalence of identified microservice identification methods



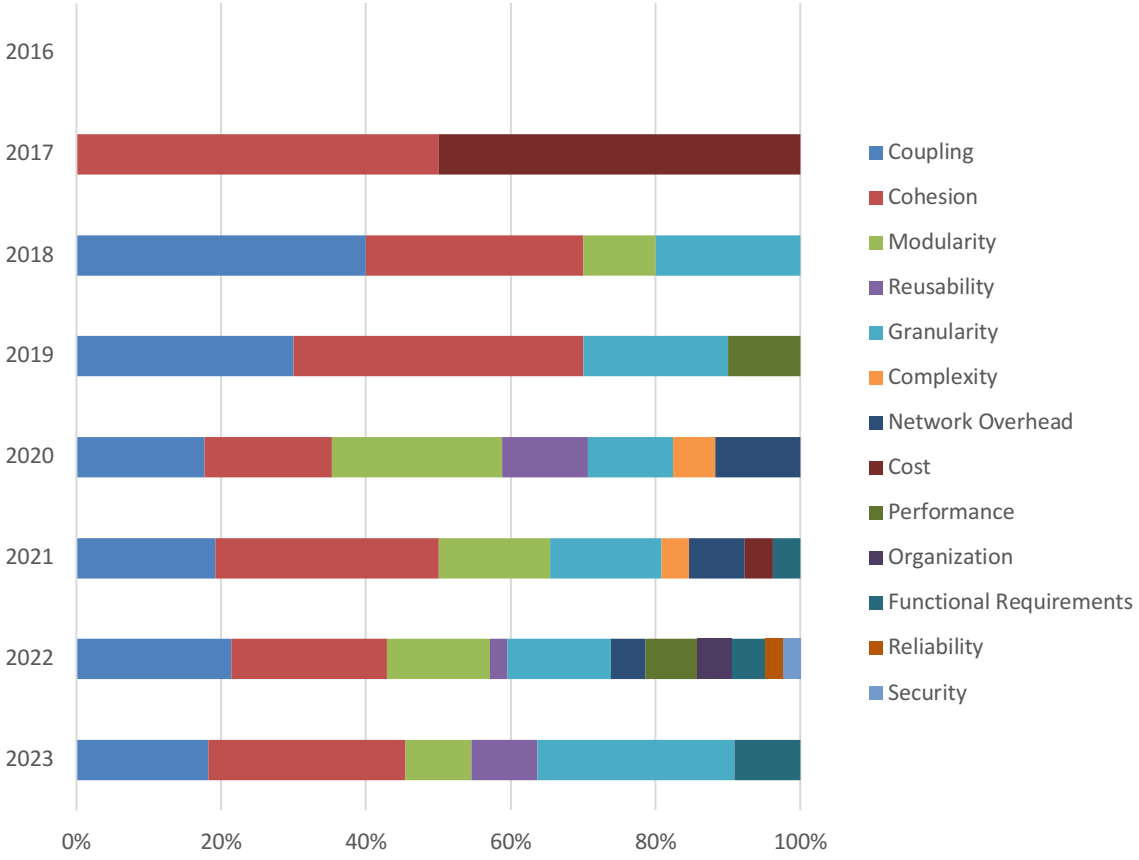
Appendix Figure E.8: Identification methods and the automation degrees of identification processes



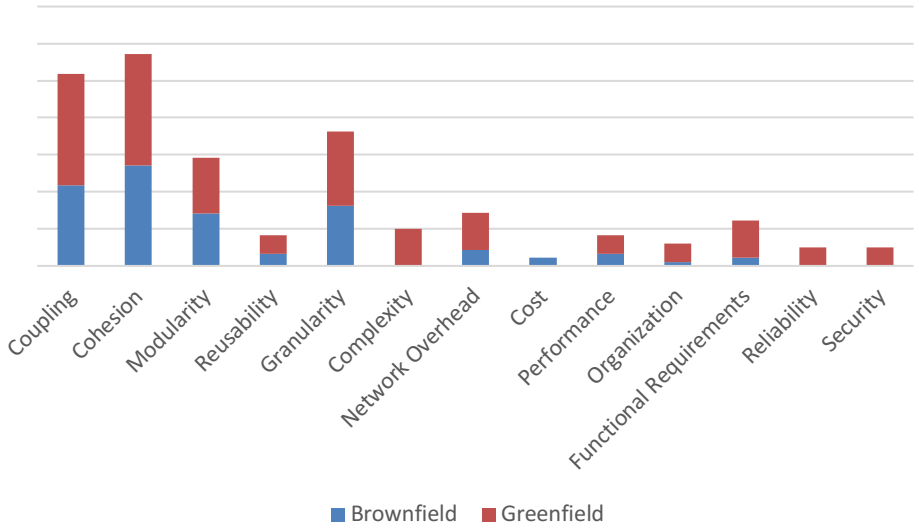
Appendix Figure E.9: Identification methods and the structure of system representations



Appendix Figure E.10: Overview of quality metric categories, with the frequency of their observation

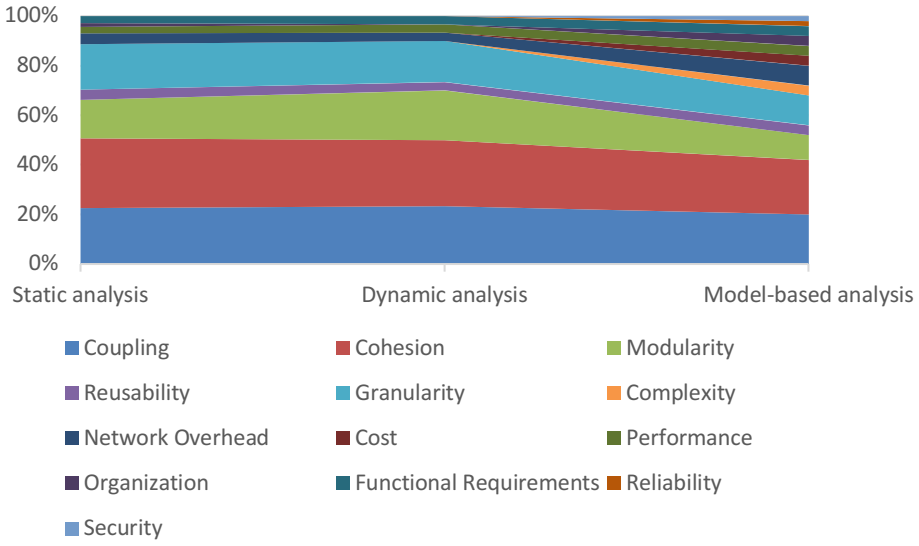


Appendix Figure E.11: Quality metrics organized by the recency of their utilization

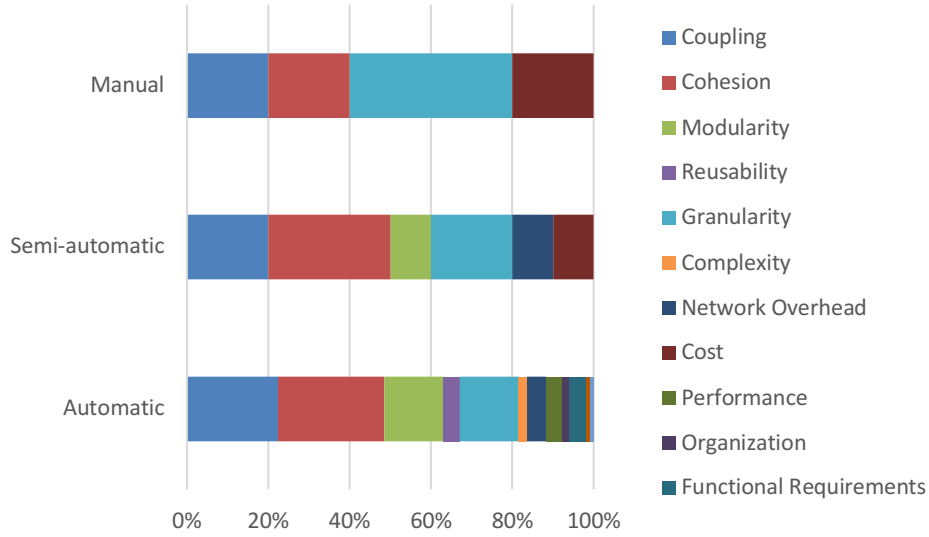


Appendix Figure E.12: Quality criteria with the initial environments of approaches applying them.

The observation frequencies in Appendix Figure E.12 are normalized to account for the higher number of brownfield studies included in the analysis.



Appendix Figure E.13: Quality metric categories, organized by data collection methods



Appendix Figure E.14: Relationships between automation and metric categories

F TxtToCSV

```
import pandas as pd


# Name of the AISeL Bibliography Export .txt file to be converted to .csv
txt_file = "AISeL Raw Bibliography Export.txt"


# Name of the .csv file containing the converted AISeL Bibliography Export
csv_file = "AISeL Transformed Bibliography Export.csv"


currentRecord = {"title": "", "author": "", "abstract": "", "type": "",
"year": "", "publication": "", "link": ""}
df = pd.DataFrame(columns=['title', 'author', 'abstract', 'type', 'year',
'publication', 'link'])


# Open and read the txt file with error handling
try:
    with open(txt_file, "r", encoding='utf-8') as file:
        # Remove leading and trailing whitespaces, then store in a list
        lines = [line.strip() for line in file.readlines()]
except FileNotFoundError:
    print("File does not exist")
    exit()
```

```
# Scan the list of lines and write relevant lines to a dictionary
for line in lines:
    if line.startswith("%T "):
        currentRecord["title"] = line[3:]
    elif line.startswith("%A "):
        currentRecord["author"] += line[3:] + ";"
    elif line.startswith("%X "):
        currentRecord["abstract"] = line[3:]
    elif line.startswith("%0 "):
        currentRecord["type"] = line[3:]
    elif line.startswith("%D "):
        currentRecord["year"] = line[3:]
    elif line.startswith("%B "):
        currentRecord["publication"] = line[3:]
    elif line.startswith("%U "):
        currentRecord["link"] = line[3:]
    elif line == "":
        if currentRecord["title"] != "":
            df = df.append(currentRecord, ignore_index=True)
            # Empty the dictionary
            currentRecord = {key: "" for key in currentRecord}

# Export data to a csv file using pandas
df.to_csv(csv_file, index=False)
```

G Digital Appendix

The following files are stored on the enclosed/attached USB flash drive:

G.1 Digital Appendix 1: Basket of Eight

The AIS Senior Scholars' List of Premier Journals and their coverage by selected databases

File path: Digital Appendix/AIS Senior Scholars' List of Premier Journals.xlsx

G.2 Digital Appendix 2: JOURQUAL3

File path: Digital Appendix/VHB-JOURQUAL3.xlsx

G.3 Digital Appendix 3: Journal and Database Search

This is a deduplicated list, containing the unique journals selected for the literature reviews.

The journals were selected based on renowned rankings.

Scientific databases were chosen to cover all selected journals, as is required by the applied methodology. These databases are listed with the chosen journals covered by them.

File path: Digital Appendix/Chosen Journals and Databases.xlsx

G.4 Digital Appendix 4: Full Search Strings

The sheet "Architectural Styles" contains the comprehensive list of search strings applied in the first literature review.

The sheet "Microservice Identification" contains the comprehensive list of search strings applied in the second literature review.

File path: Digital Appendix/Full Search Strings.xlsx

G.5 Digital Appendix 5: Forward Backward Searches

The sheet "Architectural Styles" contains the results of the first literature review.

The sheet "Microservice Identification" contains the results of the second literature review

File path: Digital Appendix/Forward Backward Searches.xlsx

G.6 Digital Appendix 6: Architectural Styles Literature Summary Table

File path: Digital Appendix/Architectural Styles Literature Summary Table.xlsx

G.7 Digital Appendix 7: Microservice Identification Literature Synthesis Matrix

File path: Digital Appendix/Literature Synthesis Matrix of Microservice Identification Approaches.xlsx

G.8 Digital Appendix 8: Concept Matrix of Quality Metrics

File path: Digital Appendix/Concept Matrix of Quality Metrics.xlsx

G.9 Digital Appendix 9: PDF Version of this Thesis

File path: Digital Appendix/Digital Version of the Thesis.pdf

References

Abdelfattah A, Cerny T (2022). Microservices Security Challenges and Approaches. In: Proc 30th International Conference on Information Systems Development (ISD), pp. 1–8, Cluj-Napoca, Romania.

Abdellatif M, Hecht G, Mili H, Elboussaidi G, Moha N, Shatnawi A, Privat J, Guéhéneuc Y-G. (2018). State of the Practice in Service Identification for SOA Migration in Industry. In: C. Pahl, M. Vukovic, J. Yin, & Q. Yu (Eds.), Lecture Notes in Computer Science. Service-Oriented Computing, pp. 634–650. Springer International Publishing. doi: 10.1007/978-3-030-03596-9_46.

Abdellatif M, Shatnawi A, Mili H, Moha N, Boussaidi GE, Hecht G, Privat J, Guéhéneuc Y-G (2021). A taxonomy of service identification approaches for legacy software systems modernization. Journal of Systems and Software, 173, pp. 1–22. doi: 10.1016/j.jss.2020.110868.

Abgaz Y, McCarren A, Elger P, Solan D, Lapuz N, Bivol M, Jackson G, Yilmaz M, Buckley J, Clarke P (2023). Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review. IEEE Transactions on Software Engineering, 49(8), pp. 4213–4242. doi: 10.1109/TSE.2023.3287297.

Aggelopoulou E, Pramatari K (2009). Factors Affecting The Adoption Of Service Oriented Architectures. In: Proc 4th Mediterranean Conference on Information Systems (MCIS), pp. 435–446, Athens, Greece.

Aksakalli IK, Çelik T, Can AB, Teki̇nerdoğan B (2021). Deployment and communication patterns in microservice architectures: A systematic literature review. Journal of Systems and Software, 180, pp. 1–25. doi: 10.1016/j.jss.2021.111014.

Al-Debagy O, Martinek P (2018). A Comparative Review of Microservices and Monolithic Architectures. In: Proc 18th International Symposium on Computational Intelligence and Informatics (CINTI), pp. 149–154. IEEE, Budapest, Hungary. doi: 10.1109/CINTI.2018.8928192.

Alshuqayran N, Ali N, Evans R (2016). A Systematic Mapping Study in Microservice Architecture. In: Proc 9th International Conference on Service-Oriented Computing and Applications (SOCA), pp. 44–51. IEEE, Macau, China. doi: 10.1109/SOCA.2016.15.

Andriyanto A, Doss R, Yustianto P (2019). Adopting SOA and Microservices for Inter-enterprise Architecture in SME Communities. In: Proc 6th International Conference on Electrical, Electronics and Information Engineering (ICEEIE), pp. 282–287. IEEE, Denpasar, Bali, Indonesia. doi: 10.1109/ICEEIE47180.2019.8981437.

Antikainen J, Pekkola S (2008). Factors influencing the alignment of SOA development with business objectives. In: Proc 16th European Conference on Information Systems (ECIS), pp. 1–13, Galway, Ireland.

Association for Information Systems (Ed.). (2023, February 17) Senior Scholars' List of Premier Journals. https://aisnet.org/page/SeniorScholarListofPremierJournals.

Assunção WKG, Colanzi TE, Carvalho L, Garcia A, Pereira JA, Lima MJ de, Lucena C (2022). Analysis of a many-objective optimization approach for identifying microservices from legacy systems. Empirical Software Engineering, 27(2), pp. 1–31. doi: 10.1007/s10664-021-10049-7.

Auer F, Lenarduzzi V, Felderer M, Taibi D (2021). From monolithic systems to Microservices: An assessment framework. Information and Software Technology, 137, pp. 1–12. doi: 10.1016/j.infsof.2021.106600.

Bajaj D, Goel A, Gupta SC (2022). GreenMicro: Identifying Microservices From Use Cases in Greenfield Development. IEEE Access, 10, pp. 67008–67018. doi: 10.1109/ACCESS.2022.3182495.

Baškarada S, Nguyen V, Koronios A (2020). Architecting Microservices: Practical Opportunities and Challenges. Journal of Computer Information Systems, 60(5), pp. 1–9. doi: 10.1080/08874417.2018.1520056.

Becker A, Widjaja T, Buxmann P (2011). Value Potentials and Challenges of Service-Oriented Architectures. Business & Information Systems Engineering, 3(4), pp. 199–210. doi: 10.1007/s12599-011-0167-3.

Beimborn D, Joachim N, Schlosser F, Streicher B (2009). The Role of IT/Business Alignment for Achieving SOA Business Value - Proposing a Research Model. In: Proc 15th Americas Conference on Information Systems 2009 (AMCIS), pp. 1–8, San Francisco, USA.

Beimborn D, Joachim N, Weitzel T (2008). Drivers and Inhibitors of SOA Business Value - Conceptualizing a Research Model. In: Proc 14th Americas Conference on Information Systems (AMCIS), pp. 1–10, Toronto, Canada.

Benavente V, Yantas L, Moscol I, Rodriguez C, Inquilla R, Pomachagua Y (2022). Comparative Analysis of Microservices and Monolithic Architecture. In: Proc 14th International Conference on Computational Intelligence and Communication Networks (CICN), pp. 177–184. IEEE, Al-Khobar, Saudi Arabia. doi: 10.1109/CICN56167.2022.10008275.

Beydoun G, Xu D, Sugumaran V (2013). Service Oriented Architectures (SOA) Adoption Challenges. International Journal of Intelligent Information Technologies, 9(2), pp. 1–6. doi: 10.4018/jiit.2013040101.

Billawa P, Bambhore Tukaram A, Díaz Ferreyra NE, Steghöfer J-P, Scandariato R, Simhandl G (2022). SoK: Security of Microservice Applications: A Practitioners' Perspective on Challenges and Best Practices. In: Proc 17th International Conference on Availability, Reliability and Security (ARES), pp. 1–10. ACM, Vienna, Austria. doi: 10.1145/3538969.3538986.

Blinowski G, Ojdowska A, Przybylek A (2022). Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. IEEE Access, 10, pp. 20357–20374. doi: 10.1109/ACCESS.2022.3152803.

Bogner J, Wagner S, Zimmermann A (2017). Towards a practical maintainability quality model for service-and microservice-based systems. In: R. de Lemos (Ed.), Proceedings 11th European Conference on Software Architecture: Companion Proceedings (ECSA), pp. 195–198. ACM, Canterbury, United Kingdom. doi: 10.1145/3129790.3129816.

Boh WF, Yellin DM (2010). Enablers and benefits of implementing Service-Oriented Architecture: an empirical investigation. International Journal of Information Technology and Management, 9(1), pp. 3–29. doi: 10.1504/IJITM.2010.029432.

Brito M, Cunha J, Saraiva J (2021). Identification of microservices from monolithic applications through topic modelling. In: C.-C. Hung, J. Hong, A. Bechini, & E. Song (Eds.), Proc 36th Annual ACM Symposium on Applied Computing (SAC), pp. 1409–1418. ACM. doi: 10.1145/3412841.3442016.

Camilli M, Colarusso C, Russo B, Zimeo E (2023). Actor-Driven Decomposition of Microservices through Multi-level Scalability Assessment. ACM Transactions on Software Engineering and Methodology, 32(5), pp. 1–46. doi: 10.1145/3583563.

Capuano R, Muccini H (2022). A Systematic Literature Review on Migration to Microservices: a Quality Attributes perspective. In: Proc 19th International Conference on Software Architecture Companion (ICSA-C), pp. 120–123. IEEE, Honolulu, HI, USA. doi: 10.1109/ICSA-C54293.2022.00030.

Carvalho L, Garcia A, Colanzi TE, Assuncao WKG, Pereira JA, Fonseca B, Ribeiro M, Lima MJ de, Lucena C (2020). On the Performance and Adoption of Search-Based Microservice Identification with toMicroservices. In: Proc 36th International Conference on Software Maintenance and Evolution (ICSME), pp. 569–580. IEEE, Adelaide, Australia. doi: 10.1109/ICSME46990.2020.00060.

Carvalho L, Garcia A, K. G. Assuncao W, Mello R de, Julia de Lima M (2019). Analysis of the Criteria Adopted in Industry to Extract Microservices. In: Proc Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice

(SER&IP), pp. 22–29. IEEE/ACM, Montreal, QC, Canada. doi: 10.1109/CESSER-IP.2019.00012.

Cerny T, Donahoo MJ, Pechanec J (2017). Disambiguation and Comparison of SOA, Microservices and Self-Contained Systems. In: K. Grygiel, T. Cerny, S.-R. Kim, & W. Wang (Eds.), Proc International Conference on Research in Adaptive and Convergent Systems (RACS), pp. 228–235. ACM, Krakow, Poland. doi: 10.1145/3129676.3129682.

Christoforou A, Andreou AS, Garriga M, Baresi L (2022). Adopting microservice architecture: A decision support model based on genetically evolved multi-layer FCM. Applied Soft Computing, 114, pp. 1–17. doi: 10.1016/j.asoc.2021.108066.

Cojocaru M-D, Uta A, Oprescu A-M (2019). Attributes Assessing the Quality of Microservices Automatically Decomposed from Monolithic Applications. In: Proc 18th International Symposium on Parallel and Distributed Computing (ISPDC), pp. 84–93. IEEE, Amsterdam, Netherlands. doi: 10.1109/ISPDC.2019.00021.

Cooper HM (1988). Organizing knowledge syntheses: A taxonomy of literature reviews. Knowledge in Society, 1(1), pp. 104–126.

Di Francesco P, Lago P, Malavolta I (2019). Architecting with microservices: A systematic mapping study. Journal of Systems and Software, 150, pp. 77–97. doi: 10.1016/j.jss.2019.01.001.

Di Francesco P, Malavolta I, Lago P (2017). Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption. In: Proc International Conference on Software Architecture (ICSA), pp. 21–30. IEEE, Gothenburg, Sweden. doi: 10.1109/ICSA.2017.24.

Ding X, Zhang C (2022). How Can We Cope with the Impact of Microservice Architecture Smells? In: Proc 11th International Conference on Software and Computer Applications (ICSCA), pp. 8–14. ACM, Melaka, Malaysia. doi: 10.1145/3524304.3524306.

Erickson J, Siau K (2008). Critical Success Factors in SOA Implementation. In: Proc 14th Americas Conference on Information Systems (AMCIS), pp. 1–9, Toronto, Canada.

Fritzsch J, Bogner J, Zimmermann A, Wagner S. (2019). From Monolith to Microservices: A Classification of Refactoring Approaches. In: J.-M. Bruel, M. Mazzara, & B. Meyer (Eds.), Lecture Notes in Computer Science. Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment, Vol. 11350, pp. 128–141. Springer International Publishing. doi: 10.1007/978-3-030-06019-0_10.

Gan Y, Zhang Y [Yanqi], Cheng D, Shetty A, Rathi P, Katarki N, Bruno A, Hu J, Ritchken B, Jackson B, Hu K, Pancholi M, He Y, Clancy B, Colen C, Wen F, Leung C, Wang S, Zaruvinsky L, . . . Delimitrou C (2019). An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In: I. Bahar, M. Herlihy, E. Witchel, & A. Lebeck (Eds.), Proc 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 3–18. ACM, Providence, RI, USA. doi: 10.1145/3297858.3304013.

Goncalves N, Faustino D, Silva AR, Portela M (2021). Monolith Modularization Towards Microservices: Refactoring and Performance Trade-offs. In: Proc 18th International Conference on Software Architecture Companion (ICSA-C), pp. 1–8. IEEE, Stuttgart, Germany. doi: 10.1109/ICSA-C52384.2021.00015.

Gos K, Zabierowski W (2020). The Comparison of Microservice and Monolithic Architecture. In: Proc 26th International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH), pp. 150–153. IEEE, Lviv, Ukraine. doi: 10.1109/MEMSTECH49584.2020.9109514.

Götz B, Schel D, Bauer D, Henkel C, Einberger P, Bauernhansl T (2018). Challenges of Production Microservices. Procedia CIRP, 67, pp. 167–172. doi: 10.1016/j.procir.2017.12.194.

Gouigoux J-P, Tamzalit D (2017). From Monolith to Microservices: Lessons Learned on an Industrial Migration to a Web Oriented Architecture. In: Proc International Conference on Software Architecture Workshops (ICSAW), pp. 62–65. IEEE, Gothenburg, Sweden. doi: 10.1109/ICSAW.2017.35.

Gravanis D, Kakarontzas G, Gerogiannis V (2021). You don't need a Microservices Architecture (yet). In: Proc 2nd European Symposium on Software Engineering (ESSE), pp. 39–44. ACM, Larissa, Greece. doi: 10.1145/3501774.3501780.

Haselböck S, Weinreich R, Buchgeher G (2017). Decision guidance models for microservices. In: O. Rysavy, V. Vranić, & G. A. Papadopoulos (Eds.), Proc 5th European Conference on the Engineering of Computer-Based Systems (ECBS), pp. 1–10. ACM, Larnaca, Cyprus. doi: 10.1145/3123779.3123804.

Hennig-Thurau, T., & Sattler, H. (Eds.) VHB-JOURQUAL3: Business & Information Systems Engineering: Bpm 2012 International Workshops, Tallinn, Estonia, September 3, 2012, Revised Papers. Verband der Hochschullehrer für Betriebswirtschaft e.V.

Hustad E, Olsen DH (2021). Creating a sustainable digital infrastructure: The role of service-oriented architecture. Procedia Computer Science, 181, pp. 597–604. doi: 10.1016/j.procs.2021.01.210.

Ilk N, Goes P, Zhao J (2010). A framework to support service-oriented architecture investment decision. In: Proc 31st International Conference on Information Systems (ICIS), pp. 1–15. AIS Electronic Library, Saint Louis, USA.

Jagiełło M, Rusek M, Karwowski W. (2019). Performance and Resilience to Failures of an Cloud-Based Application: Monolithic and Microservices-Based Architectures Compared. In: K. Saeed, R. Chaki, & V. Janev (Eds.), Lecture Notes in Computer Science. Computer Information Systems and Industrial Management, Vol. 11703, pp. 445–456. Springer International Publishing. doi: 10.1007/978-3-030-28957-7_37.

Jamshidi P, Pahl C, Mendonca NC, Lewis J, Tilkov S (2018). Microservices: The Journey So Far and Challenges Ahead. IEEE Software, 35(3), pp. 24–35. doi: 10.1109/MS.2018.2141039.

Jatkiewicz P, Okrój S (2023). Differences in performance, scalability, and cost of using microservice and monolithic architecture. In: J. Hong, M. Lanperne, J. W. Park, T. Cerny, & H. Shahriar (Eds.), Proc 38th ACM/SIGAPP Symposium on Applied Computing (SAC), pp. 1038–1041. ACM, Tallinn, Estonia. doi: 10.1145/3555776.3578725.

Jewargi K (2023). Public Cloud to Cloud Repatriation Trend. Scholars Journal of Engineering and Technology, 11(1), pp. 1–3. doi: 10.36347/sjet.2023.v11i01.001.

Joachim N (2011). A Literature Review of Research on Service-Oriented Architectures (SOA): Characteristics, Adoption Determinants, Governance Mechanisms, and Business Impact. In: AIS electronic library (Ed.), Proc 17th Americas Conference on Information Systems (AMCIS), pp. 1–11, Detroit, Michigan, USA.

Joachim N, Beimborn D, Hoberg P, Schlosser F (2009). Examining the Organizational Decision to Adopt Service-Oriented Architecture (SOA) - Development of a Research Model. In: Proc 10th Workshop Digital Broadcasting (DIGIT), pp. 1–20.

Kalia AK, Xiao J, Krishna R, Sinha S, Vukovic M, Banerjee D (2021). Mono2Micro: a practical and effective tool for decomposing monolithic Java applications to microservices. In: D. Spinellis, G. Gousios, M. Chechik, & M. Di Penta (Eds.), Proc 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), pp. 1214–1224. ACM, Athens, Greece. doi: 10.1145/3468264.3473915.

Khoshnevis S (2023). A search-based identification of variable microservices for enterprise SaaS. Frontiers of Computer Science, 17(3), pp. 1–16. doi: 10.1007/s11704-022-1390-4.

Kleftakis S, Mavrogiorgou A, Zafeiropoulos N, Mavrogiorgos K, Kiourtis A, Kyriazis D (2022). A Comparative Study of Monolithic and Microservices Architectures in Machine Learning Scenarios. In: Proc 2nd International Conference on Computing

(ICOCO), pp. 352–357. IEEE, Kota Kinabalu, Malaysia. doi: 10.1109/ICOCO56118.2022.10031648.

Klose K, Knackstedt R, Beverungen D (2007). Identification of services-a stakeholder-based approach to SOA development and its application in the area of production planning. In: Proc 15th European Conference on Information Systems (ECIS), pp. 1802–1814, St. Gallen, Switzerland.

Laigner R, Zhou Y, Salles MAV, Liu Y, Kalinowski M (2021). Data management in microservices. Proceedings of the VLDB Endowment, 14(13), pp. 3348–3361. doi: 10.14778/3484224.3484232.

Leonard J (2009). All Part of the Service? Addressing Data Longevity in Service Oriented Architectures. In: Proc 20th Australasian Conference on Information Systems (ACIS), pp. 1013–1023, Melbourne, Australia.

Lewis J, Fowler M. (2014). *Microservices: A Definition of This New Architectural Term*. https://www.martinfowler.com/articles/microservices.html

Li J [Jianan], Xu H, Xu X, Wang Z (2022). A Novel Method for Identifying Microservices by Considering Quality Expectations and Deployment Constraints. International Journal of Software Engineering and Knowledge Engineering, 32(03), pp. 417–437. doi: 10.1142/S021819402250019X.

Li S, Zhang H, Jia Z, Li Z [Zheng], Zhang C, Li J [Jiaqi], Gao Q, Ge J, Shan Z (2019). A dataflow-driven approach to identifying microservices from monolithic applications. Journal of Systems and Software, 157, pp. 1–16. doi: 10.1016/j.jss.2019.07.008.

Li S, Zhang H, Jia Z, Zhong C [Chenxing], Zhang C, Shan Z, Shen J, Babar MA (2021). Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. Information and Software Technology, 131, pp. 1–23. doi: 10.1016/j.infsof.2020.106449.

Li Z [Zhiding], Shang C, Wu J, Li Y (2022). Microservice extraction based on knowledge graph from monolithic applications. Information and Software Technology, 150, pp. 1–20. doi: 10.1016/j.infsof.2022.106992.

Lira C, Batista E, Delicato FC, Prazeres C (2023). Architecture for IoT applications based on reactive microservices: A performance evaluation. Future Generation Computer Systems, 145, pp. 223–238. doi: 10.1016/j.future.2023.03.026.

Liu B, Xiong J, Ren Q, Tyszberowicz S, Yang Z (2022). Log2MS: a framework for automated refactoring monolith into microservices using execution logs. In: Proc International Conference on Web Services (ICWS), pp. 391–396. IEEE, Barcelona, Spain. doi: 10.1109/ICWS55610.2022.00065.

Liu G, Huang B, Liang Z, Qin M, Zhou H, Li Z [Zhang] (2020). Microservices: architecture, container, and challenges. In: Proc 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), pp. 629–635. IEEE, Macau, China. doi: 10.1109/QRS-C51114.2020.00107.

MacLennan E, van Belle J-P (2012). Factors Affecting The Organizational Adoption Of Service-Oriented Architecture (Soa). In: Proc 16th Pacific Asia Conference on Information Systems (PACIS), pp. 1–14, Ho Chi Minh City, Vietnam.

Mahdavi-Hezavehi S, Galster M, Avgeriou P (2013). Variability in quality attributes of service-based software systems: A systematic literature review. Information and Software Technology, 55(2), pp. 320–343. doi: 10.1016/j.infsof.2012.08.010.

Mahmood Z (2007). Service Oriented Architecture: Potential Benefits and Challenges. In: Proc of the 11th WSEAS International Conference on Computers (ICCOMP), pp. 497–501. World Scientific and Engineering Academy and Society, Agios Nikolaos, Greece.

Mangwani P, Mangwani N, Motwani S (2023). Evaluation of a Multitenant SaaS Using Monolithic and Microservice Architectures. SN Computer Science, 4(2), pp. 1–8. doi: 10.1007/s42979-022-01610-2.

Mendonça NC, Box C, Manolache C, Ryan L (2021). The Monolith Strikes Back: Why Istio Migrated From Microservices to a Monolithic Architecture. IEEE Software, 38(5), pp. 17–22. doi: 10.1109/MS.2021.3080335.

Munaf RM, Ahmed J, Khakwani F, Rana T (2019). Microservices Architecture: Challenges and Proposed Conceptual Design. In: Proc 2nd International Conference on Communication Technologies (ComTech), pp. 82–87. IEEE, Rawalpindi, Pakistan. doi: 10.1109/COMTECH.2019.8737831.

Newman S. (2021) (Second edition) Building microservices: Designing fine-grained systems. O'Reilly Media, Beijing, Boston, Farnham, Sebastopol, Tokyo.

Niknejad N, Ismail W, Ghani I, Nazari B, Bahari M, Hussin ARBC (2020). Understanding Service-Oriented Architecture (SOA): A systematic literature review and directions for further investigation. Information Systems, 91, pp. 1–27. doi: 10.1016/j.is.2020.101491.

Niño-Martínez VM, Ocharán-Hernández JO, Limón X, Pérez-Arriaga JC (2022). A Microservice Deployment Guide. Programming and Computer Software, 48(8), pp. 632–645. doi: 10.1134/S0361768822080151.

Nitin V, Asthana S, Ray B, Krishna R (2022). CARGO: AI-Guided Dependency Analysis for Migrating Monolithic Applications to Microservices Architecture. In: Proc 37th

IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 1–12. ACM, Rochester, MI, USA. doi: 10.1145/3551349.3556960.

O'Brien L, Merson P, Bass L (2007). Quality Attributes for Service-Oriented Architectures. In: Proc International Workshop on Systems Development in SOA Environments (SDSOA: ICSE Workshops), pp. 1–30. IEEE, Minneapolis, MN, USA. doi: 10.1109/SDSOA.2007.10.

Oliveira Rosa T de, Daniel JFL, Guerra EM, Goldman A (2020). A Method for Architectural Trade-off Analysis Based on Patterns. In: Proc 25th European Conference on Pattern Languages of Programs (EuroPLoP), pp. 1–8. ACM, Virtual Event, Germany. doi: 10.1145/3424771.3424809.

Oumoussa I, Saidi R (2024). Evolution of Microservices Identification in Monolith Decomposition: A Systematic Review. IEEE Access, 12, pp. 23389–23405. doi: 10.1109/ACCESS.2024.3365079.

Pahl C, Jamshidi P (2016). Microservices: A Systematic Mapping Study. In: Proc 6th International Conference on Cloud Computing and Services Science (CLOSER), pp. 137–146. SCITEPRESS - Science and Technology Publications, Rome, Italy. doi: 10.5220/0005785501370146.

Papakonstantinou I, Kalafatidis S, Mamatas L (2020). A Techno-Economic Assessment of Microservices. In: Proc 16th International Conference on Network and Service Management (CNSM), pp. 1–5. IEEE, Izmir, Turkey. doi: 10.23919/CNSM50824.2020.9269114.

Paperpile. (2023, July 13). *Online BibTeX to CSV converter*. https://www.bibtex.com/c/bibtex-to-csv-converter/

Phan C (2007). Service Oriented Architecture (SOA) - Security Challenges and Mitigation Strategies. In: Proc IEEE Military Communications Conference (MILCOM), pp. 1–7. IEEE, Orlando, FL, USA. doi: 10.1109/MILCOM.2007.4455012.

Ponce F, Marquez G, Astudillo H [Hernan] (2019). Migrating from monolithic architecture to microservices: A Rapid Review. In: Proc 38th International Conference of the Chilean Computer Science Society (SCCC), pp. 1–7. IEEE, Concepcion, Chile. doi: 10.1109/SCCC49216.2019.8966423.

Pulparambil S, Baghdadi Y, Salinesi C (2021). A methodical framework for service oriented architecture adoption: Guidelines, building blocks, and method fragments. Information and Software Technology, 132, pp. 1–12. doi: 10.1016/j.infsof.2020.106487.

Qian L, Li J [Jing], He X, Gu R, Shao J, Lu Y (2023). Microservice extraction using graph deep clustering based on dual view fusion. Information and Software Technology, 158, pp. 1–11. doi: 10.1016/j.infsof.2023.107171.

Raharjo AB, Andyartha PK, Wijaya WH, Purwananto Y, Purwitasari D, Juniarta N (2022). Reliability Evaluation of Microservices and Monolithic Architectures. In: Proc International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM), pp. 1–7. IEEE, Surabaya, Indonesia. doi: 10.1109/CENIM56801.2022.10037281.

Raj V, Ravichandra S (2018). Microservices: A perfect SOA based solution for Enterprise Applications compared to Web Services. In: Proc 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), pp. 1531–1536. IEEE, Bangalore, India. doi: 10.1109/RTEICT42901.2018.9012140.

Raj V, Sadam R (2021). Evaluation of SOA-Based Web Services and Microservices Architecture Using Complexity Metrics. SN Computer Science, 2(5), pp. 1–10. doi: 10.1007/s42979-021-00767-6.

Reddy VK, Dubey A, Lakshmanan S, Sukumaran S, Sisodia R (2009). Evaluating legacy assets in the context of migration to SOA. Software Quality Journal, 17(1), pp. 51–63. doi: 10.1007/s11219-008-9055-6.

Saidani I, Ouni A, Mkaouer MW, Saied A. (2019). Towards Automated Microservices Extraction Using Muti-objective Evolutionary Search. In: S. Yangui, I. Bouassida Rodriguez, K. Drira, & Z. Tari (Eds.), Lecture Notes in Computer Science. Service-Oriented Computing, Vol. 11895, pp. 58–63. Springer International Publishing. doi: 10.1007/978-3-030-33702-5_5.

Salii S, Ajdari J, Zenuni X (2023). Migrating to a microservice architecture: benefits and challenges. In: Proc 46th MIPRO ICT and Electronics Convention (MIPRO), pp. 1670–1677. IEEE, Opatija, Croatia. doi: 10.23919/MIPRO57284.2023.10159894.

Santos A, Paula H (2021). Microservice decomposition and evaluation using dependency graph and silhouette coefficient. In: C. Vasconcellos, K. Roggia, P. Bousfield, V. Collere, & R. Bonifácio (Eds.), Proc 15th Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS), pp. 51–60. ACM, Joinville, Brazil. doi: 10.1145/3483899.3483908.

Schramm M, Daneva M (2016). Implementations of service oriented architecture and agile software development: What works and what are the challenges? In: Proc 10th International Conference on Research Challenges in Information Science (RCIS), pp. 1–12. IEEE, Grenoble, France. doi: 10.1109/RCIS.2016.7549345.

Schröer C, Kruse F, Marx Gómez J. (2020). A Qualitative Literature Review on Microservices Identification Approaches. In: S. Dustdar (Ed.), Communications in

Computer and Information Science. Service-Oriented Computing, Vol. 1310, pp. 151–168. Springer International Publishing. doi: 10.1007/978-3-030-64846-6_9.

Sellami K, Ouni A, Saied MA, Bouktif S, Mkaouer MW (2022). Improving microservices extraction using evolutionary search. Information and Software Technology, 151, pp. 1–14. doi: 10.1016/j.infsof.2022.106996.

Sellami K, Saied MA, Ouni A, Abdalkareem R. (2022). Combining Static and Dynamic Analysis to Decompose Monolithic Application into Microservices. In: J. Troya, B. Medjahed, M. Piattini, L. Yao, P. Fernández, & A. Ruiz-Cortés (Eds.), Lecture Notes in Computer Science. Service-Oriented Computing, Vol. 13740, pp. 203–218. Springer Nature Switzerland. doi: 10.1007/978-3-031-20984-0_14.

Simanta S, Morris E, Balasubramaniam S, Davenport J, Smith DB (2009). Information assurance challenges and strategies for securing SOA environments and web services. In: Proc 3rd Annual IEEE Systems Conference, pp. 173–178. IEEE, Vancouver, BC, Canada. doi: 10.1109/SYSTEMS.2009.4815791.

Sorgalla J, Wizenty P, Rademacher F, Sachweh S, Zündorf A (2021). Applying Model-Driven Engineering to Stimulate the Adoption of DevOps Processes in Small and Medium-Sized Development Organizations. SN Computer Science, 2(6), pp. 1–25. doi: 10.1007/s42979-021-00825-z.

Staffa S, Quattrocchi G, Margara A, Cugola G. (2021). Pangaea: Semi-automated Monolith Decomposition into Microservices. In: H. Hacid, O. Kao, M. Mecella, N. Moha, & H. Paik (Eds.), Lecture Notes in Computer Science. Service-Oriented Computing, Vol. 13121, pp. 830–838. Springer International Publishing. doi: 10.1007/978-3-030-91431-8_60.

Taibi D, Lenarduzzi V, Pahl C, Janes A (2017). Microservices in agile software development. In: R. Tonelli (Ed.), Proc Scientific Workshops of XP2017 (XP), pp. 1–5. ACM, Cologne, Germany. doi: 10.1145/3120459.3120483.

Tranfield D, Denyer D, Smart P (2003). Towards a methodology for developing evidence-informed management knowledge by means of systematic review. British Journal of Management, 14(3), pp. 207–222.

Trihinas D, Tryfonos A, Dikaiakos MD, Pallis G (2018). DevOps as a Service: Pushing the Boundaries of Microservice Adoption. IEEE Internet Computing, 22(3), pp. 65–71. doi: 10.1109/MIC.2018.032501519.

Velepucha V, Flores P (2021). Monoliths to microservices - Migration Problems and Challenges: A SMS. In: Proc 2nd International Conference on Information Systems and Software Technologies (ICI2ST), pp. 135–142. IEEE, Quito, Ecuador. doi: 10.1109/ICI2ST51859.2021.00027.

Vera-Rivera FH, Puerto E, Astudillo H [Hernan], Gaona CM (2021). Microservices Backlog–A Genetic Programming Technique for Identification and Evaluation of Microservices From User Stories. IEEE Access, 9, pp. 117178–117203. doi: 10.1109/ACCESS.2021.3106342.

Vera-Rivera FH, Puerto-Cuadros EG, Astudillo H [Hernán], Gaona-Cuevas CM. (2020). Microservices Backlog - A Model of Granularity Specification and Microservice Identification. In: Q. Wang, Y. Xia, S. Seshadri, & L.-J. Zhang (Eds.), Lecture Notes in Computer Science. Services Computing – SCC 2020, Vol. 12409, pp. 85–102. Springer International Publishing. doi: 10.1007/978-3-030-59592-0_6.

Villamizar M, Garces O, Ochoa L, Castro H, Salamanca L, Verano M, Casallas R, Gil S, Valencia C, Zambrano A, Lang M (2016). Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures. In: Proc 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 179–182. IEEE, Cartagena, Colombia. doi: 10.1109/CCGrid.2016.37.

Villamizar M, Garcés O, Ochoa L, Castro H, Salamanca L, Verano M, Casallas R, Gil S, Valencia C, Zambrano A, Lang M (2017). Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures. Service Oriented Computing and Applications, 11(2), pp. 233–247. doi: 10.1007/s11761-017-0208-y.

vom Brocke J, Simons A, Niehaves B, Reimer K, Plattfaut R, Cleven A (2009). Reconstructing the giant: On the importance of rigour in documenting the literature search. In: Proc 7th European Conference on Information Systems (ECIS), pp. 1–14, Verona, Italy.

Wang Y-H, Liao JC (2009). Why or Why Not Service Oriented Architecture. In: Proc IITA International Conference on Services Science, Management and Engineering (SSME), pp. 65–68. IEEE, Zhangjiajie, China. doi: 10.1109/SSME.2009.126.

Wang Y, Kadiyala H, Rubin J (2021). Promises and challenges of microservices: an exploratory study. Empirical Software Engineering, 26(4), pp. 1–44. doi: 10.1007/s10664-020-09910-y.

Waseem M, Liang P, Shahin M (2020). A Systematic Mapping Study on Microservices Architecture in DevOps. Journal of Systems and Software, 170, pp. 1–30. doi: 10.1016/j.jss.2020.110798.

Waseem M, Liang P, Shahin M, Di Salle A, Márquez G (2021). Design, monitoring, and testing of microservices systems: The practitioners' perspective. Journal of Systems and Software, 182, pp. 1–44. doi: 10.1016/j.jss.2021.111061.

Webster J, Watson RT (2002). Analyzing the past to prepare for the future: Writing a literature review. MIS Quarterly, 26(2), xiii-xxiii.

Weerasinghe LDSB, Perera I (2021). An exploratory evaluation of replacing ESB with microservices in service-oriented architecture. In: Proc 4th International Research Conference on Smart Computing and Systems Engineering (SCSE), pp. 137–144. IEEE, Colombo, Sri Lanka. doi: 10.1109/SCSE53661.2021.9568289.

White L, Wilde N, Reichherzer T, El-Sheikh E, Goehring G, Baskin A, Hartmann B, Manea M (2012). Understanding Interoperable Systems: Challenges for the Maintenance of SOA Applications. In: Proc 45th Hawaii International Conference on System Sciences (HICSS), pp. 2199–2206. IEEE, Maui, HI, USA. doi: 10.1109/HICSS.2012.614.

Wolfart D, Assunção WKG, Da Silva IF, Domingos DCP, Schmeing E, Villaca GLD, Paza DdN (2021). Modernizing Legacy Systems with Microservices: A Roadmap. In: R. Chitchyan, J. Li, B. Weber, & T. Yue (Eds.), Evaluation and Assessment in Software Engineering, pp. 149–159. ACM, Trondheim, Norway. doi: 10.1145/3463274.3463334.

Wolff E. (2018) (2. Edition) Microservices: Grundlagen flexibler Softwarearchitekturen. dpunkt.verlag, Heidelberg.

Xiao Z, Wijegunaratne I, Qiang X (2016). Reflections on SOA and Microservices. In: Proc 4th International Conference on Enterprise Systems (ES), pp. 60–67. IEEE, Melbourne, Australia. doi: 10.1109/ES.2016.14.

Yarygina T, Bagge AH (2018). Overcoming Security Challenges in Microservice Architectures. In: Proc 12th IEEE Symposium on Service-Oriented System Engineering (SOSE), pp. 11–20. IEEE, Bamberg, Germany. doi: 10.1109/SOSE.2018.00011.

Younas A, Ali P (2021). Five tips for developing useful literature summary tables for writing review articles. Evidence-Based Nursing, 24(2), pp. 32–34.

Zhang Y [Yiwen], Tanniru M (2005). Business Flexibility and Operational Efficiency - Making Trade-Offs in Service Oriented Architecture. In: Proc 11th Americas Conference on Information Systems (AMCIS), pp. 2265–2270, Omaha, Nebraska, USA.

Zhou X [Xiang], Peng X, Xie T, Sun J, Ji C, Li W, Ding D (2021). Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study. IEEE Transactions on Software Engineering, 47(2), pp. 243–260. doi: 10.1109/TSE.2018.2887384.

Zhou X [Xin], Huang H, Zhang H, Huang X, Shao D, Zhong C [Chenxin] (2022). A cross-company ethnographic study on software teams for DevOps and microservices. In: M. Harman & H. Miller (Eds.), Proc 44th International Conference on Software

Engineering: (ICSE), pp. 1–10. ACM, Pittsburgh, Pennsylvania, USA. doi: 10.1145/3510457.3513054.

Zhou X [Xin], Li S, Cao L, Zhang H, Jia Z, Zhong C [Chenxing], Shan Z, Babar MA (2023). Revisiting the practices and pains of microservice architecture in reality: An industrial inquiry. Journal of Systems and Software, 195, pp. 1–20. doi: 10.1016/j.jss.2022.111521.

University of Bamberg Press

Designing enterprise software systems demands careful selection of architectural styles and patterns that align with technical and organizational requirements. Effective selection necessitates a nuanced evaluation of organizational compatibility, rooted in a comprehensive understanding of relevant quality criteria.

This study examines strategic considerations for adopting complex architectural patterns, to address contemporary architecture design challenges. It identifies critical decision-making criteria and quality attributes that inform architectural choices, integrating them into a framework that provides structured guidance for both pattern selection and the design process.

A taxonomy of architectural design strategies, with a focus on microservices, is introduced to systematically assess design approaches within selected architectural styles. It establishes a consistent design process framework, detailing each step, and ensuring organizational fit from data collection to quality-driven service identification. The taxonomy further embeds quality criteria into the design process through appropriate quality metrics, ensuring alignment with the guiding principles of pattern selection.

This work also presents the Microservice Architecture Design Framework, a tool for guiding organizations in selecting optimal design approaches. By applying this framework, it identifies archetypal patterns and establishes best practices for microservice architecture design. This structured, quality-centric approach supports informed architectural decisions that align technical needs with strategic business objectives.

As a consultative resource, this research offers actionable guidelines for software architects, bridging the gap between theoretical foundations and practical applications in complex enterprise environments.

ISBN: 978-3-98989-038-1
9 783989 890398
www.uni-bamberg.de/ubp