Operational Research in Engineering Sciences: Theory and Applications
Vol. 7, Issue 2, 2024, pp. 417-446
ISSN: 2620-1607
eISSN: 2620-1747
DOI: https://doi.org/10.31181/oresta/070220

ASSESSING THE QUALITY OF MICROSERVICE AND MONOLITHIC-BASED ARCHITECTURES: A SYSTEMATIC LITERATURE REVIEW

Saad Hussein[1*], Mariam Lahami[2], Mouna Torjmen[3]

[1]ENETCom SFAX, ReDCAD Laboratory, University of Sfax, B.P. 1173, 3038 Sfax, Tunisia, Computer Science and Information Technology, Al-Qadisiyah University, Iraq,
[2,3] ReDCAD Laboratory, National Engineering School of Sfax, University of Sfax, Soukra Road Km 4, 3038, Sfax, Tunisia.

Received: 13 January 2024
Accepted: 22 June 2024
First Online: 30 June 2024

Research Paper

Abstract: *Building a scalable system has been found to be an even greater challenge than developing software in general, due to the complexity and otherwise involved in its development. Whereas monolithic applications are made of big entities that are developed together, independent services sum up the arrays of a micro services-based architecture. This research work will therefore come up with the framework that would be used in supporting the migration of organizations and industries into micro services. This approach gives companies the evaluative methodology for assessing their adoption of micro services. This approach enables an enterprise to measure its capacity for the effective implementation of micro services using quality criteria. An SLR was conducted, as we selected 48 relevant research papers published during the last four years, 2020–2023. Findings on the quality characteristics of monolithic versus micro services-based systems were collated to demonstrate how suitable quality attribute metrics help evaluate these architectural approaches more effectively. Key indicators can thus help transition from monolithic architectures to a micro services architecture. The outcome of the literature review brings forth the key quality attributes in addition to their sub-characteristics as follows: performance, scalability, coupling, cohesion, deployment, security, development, complexity, maintainability, and availability. The results display that interest among researchers in quality-driven micro services migration is growing while an appreciable number of studies are centred on quality enhancement as the main objective of strategies of migration.*

Keywords: *Quality Attributes; Software Engineering; Monolithic-Based Architecture; Software Measurement; Microservice-Based Architecture; Quality Attribute Assessment.*

*Corresponding Author: saad.hussain@qu.edu.iq (S. Hussein)

1. Introduction

Due to advanced computer technology, monolithic applications confront bottlenecks and challenges in terms of availability and other problems such as lower speeds of development and scaling single units 0. As quality assurance becomes an important challenge in architecture during the process of migration while moving to or developing systems based upon microservices, QAs including maintainability, reusability, and scalability take importance in the process of migration (Arzo et al., 2024; Ziadeh & Al-Qora'n, 2024). Distributed systems become more trend-worthy due to the robust, scalable, reliable, secure, and fault-tolerant design and cannot be achievable with the conventional monolithic architectures in most situations due to the application demands of modern times (Indrasiri & Siriwardena., 2018). Several companies, such as Netflix, eBay, Amazon, and IBM, have adopted microservices to augment scalability, maintainability, and flexibility despite their economic and technical difficulties (Li et al., 2021; Selmadji et al., 2020). However, there is a need for evidence-based decision-making frameworks to ensure that the migrations are worthy since automation reduces cost and manages rapid software change (Taibi & Systä, 2020; Zhang et al., 2020). Microservices also provide various benefits including elastic scalability, load balancing, and easier deployment yet require proper consideration of the quality characteristics (Jatkiewicz & Okrój, 2023; Kalia et al., 2021; Mosleh et al., 2016). The present research works towards systematic evaluation of monolithic compared to microservices architectures, including comparison of quality attributes that aid developers to implement MSA and helps towards deeper understanding of the technological processes involved during the process of migration (Zhong et al., 2024; Hassan et al., 2020).

A literature review of several research papers compares the quality attributes of monolithic and microservice-based applications, with a detailed analysis of microservice-based architecture versus monolithic architecture, using the Kitchenham methodology. To evaluate and enhance software quality for data collection, analysis, visualization, and publication. This document uses ISO 25010 Estdale and Georgiadou (2018), a quality standard established for scientific research software. This paper describes how proper-quality attribute metrics may be selected to allow the estimation of both monolithic and microservice-based systems more accurately. The metrics that are being used provide useful measures that will allow for more accurate estimation of this transition from monolithic to microservice-based architectures. The focus of this research is on identifying quality metrics that could be used in estimating software transformed into microservices that is semantically equivalent.

This work supports organisations, especially software architects, in deciding whether or not to migrate monolithic systems to microservices. In this work, the relevant objective measurements that are considered relevant to systems of interest were assessed, and further discussions and analyses on possible advantages and disadvantages regarding migration and re-architecting processes can be entertained. Knowledge gained from characterisation and metrics considered prior to migration enables the establishment of comparisons over the practical utility of them. Our contribution to the related study aims to encourage the use of microservices by making comparisons between mono- lithic systems and microservice-based systems. We have identified the strengths and weaknesses of previous research in this area. We

found that microservices are more productive and effective and that improving quality attributes has an impact on microservice migration processes. In addition to filling any gaps found during these studies and developing a plan to guide and support the migration process according to quality standards.

Section 2 presents Background on monolithic vs microservice architectures with comparison, summary of the process for microservices migration. Section 3 Analysis of related previous research to set a frame for existing work in the topic. Section 4 description of the methodology and approach followed in this study through research questions. Section 5 Research questions with Key Findings. Discussion of threats to validity: Section 6 Conclusion: Section 7 Summary of findings, conclusions, limitations, and possible future research directions.

2. Background

2.1 Monolithic Architecture

A monolithic architecture is the traditional, unified approach to building a software program. The term Monolithic is something solid and unified. According to the Cambridge Dictionary, "monolithic is too large and unable to be changed". Monolithic architecture is a very old approach for software development that the big companies Amazon and eBay used in their old approaches. In monolithic architectures functionalities are encapsulated in a single application. This can simplify development, testing, deployment, and scalability for small systems with fewer functionalities. Expanding a monolithic system simply involves duplicating the entire content. However, as software increases its complexity, constraints tend to surface (Aggarwal & Singh, 2024). For example, increased complexity can reduce the reliability and limited scalability can constrain technological advancement. In the traditional monolithic setup, users relate with a front-end application that services requests through communication with a database. All the services are running on a platform with the same code base. Thus, modification to the code base has to ensure that all services operate smoothly. Adding more services to this set also increases complexity; it is difficult for a company to implement new features by using this. Furthermore, each new release forces to restart all the services, which adversely affects the user experience. A key weakness is the single point of failure-the system crashes if one of the services crashes, taking all of them down with it (Newman, 2015), as shown in Figure 1.

Monolithic systems generally start simple but eventually grow to meet business needs. Including new features may lead to problems such as an inability to scale particular parts of a system due to tight coupling, hard maintenance code in that hidden dependencies are silently created, and increased vulnerability to failure as testing becomes harder. These problems may hinder the ability to make progress toward future stability, especially when design documentation is outdated or non-existent, and original developers no longer participate (Al-Debagy & Martinek, 2018). Although the above bottlenecks apply, monolithic architecture is still possible for some applications, especially in the development of a proof-of-concept or even a minimum viable product. Monoliths are easier in the initial development as they use a single shared code base. It is easier to debug since only one process and one memory

space exists, and it is less complex to visualize (Ponce et al., 2019). Some features of monolithic applications often have several sections combined into a single large application which can, in turn display some features (Su et al., 2024).

a. Authorization is the act of giving a user sufficient permission to use a program. It is, in effect, a response to HTTP requests using data formats like JSON, XML, or HTML. Underlying business logic drives features and functionality of the application but usually implemented by the layer of the DBMS.

b. It envelops the database access objects that the application uses. Moreover, the binding with services or data sources is controlled and managed.

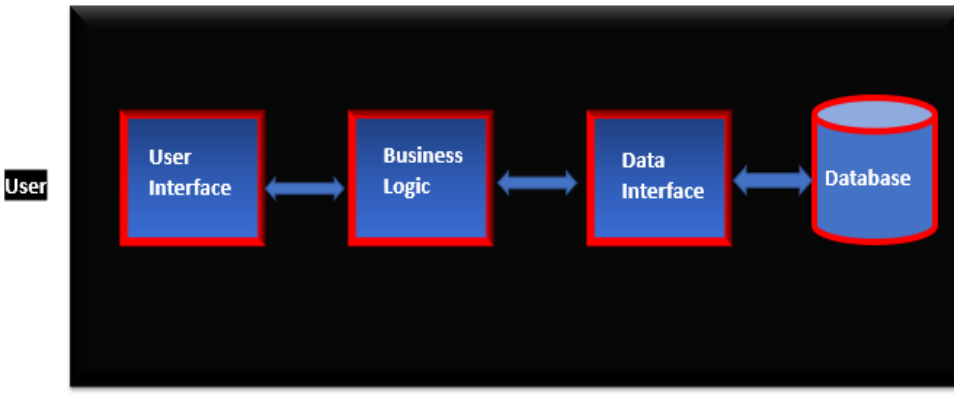

Figure 1: Monolithic Architecture Overview

As the application size increases, the gains of a monolithic architecture must be weighed against disadvantages. Large monoliths are normally difficult to create, debug, and maintain over time. Finally, disadvantages replace advantages of the monolithic design. In such scenarios, a microservices architecture may be the best choice for the application migration process. Contrary to the monolithic systems, microservices are mostly loosely coupled, decentralised units of execution (El Akhdar et al., 2024). Monolithic applications integrate a number of components into one large application and due to the size of the code base poses most of the challenges for management and long-term maintenance (Oumoussa & Saidi, 2024).

2.2 Microservice Architecture

It is an architectural style of software development using independent components, each focused on specific fine-grained business activities and communicating with each other through well-defined interfaces (Hassan et al., 2020). A microservice is also a self-contained small service using lightweight protocols for communication. The use of microservice architecture has gained much popularity as the new architectural approach for modern applications (Faustino et al., 2024; Schröer et al., 2021)abase. Each microservice can independently carry out update, testing, deployment, and scalability operations. Separating domain-specific concerns from core business functions does indeed achieve separation from other technologies-creating an independent code base for every service (Alshuqayran et al., 2016). As such, although microservices do not reduce complexity per se, they decompose activities into smaller, independently running processes that improve system visibility

and make complexity more manageable, as depicted in Figure 2. One of the advantages of MSA is fault tolerance-the application continues functioning even in case some microservices have failed. In addition, horizontal scaling guarantees that only the strained microservice needs to be strengthened. For example, each microservice can use different technologies based upon the business requirements, without any technological constraints. Microservices break up large applications into smaller independent services with the following attributes (Andrade et al., 2022):

a. They can be deployed and tested independently, loosely coupled with APIs, have different technology stacks, and thus are deployed independently.

b. Microservices are developed and operate according to business capabilities. This is the very reason they enjoy integration with a cloud environment.

c. Such aspects make it easier for developers to handle components with interfaces that are relatively easier to understand. Lesser numbers of components ease the coordination and testing, meaning quicker updates and scalable, available applications.

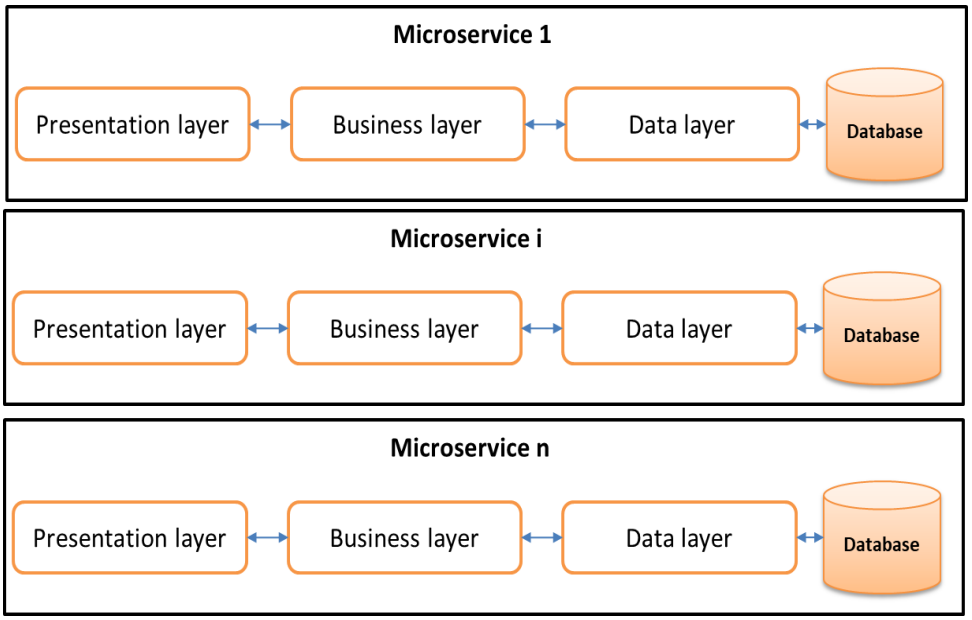

Figure 2: Microservice Architecture Overview

2.3 Comparison of Monolithic Vs Microservice

Figure 3 and Figure 4 represent the basic differences in monolithic and microservices architecture. One process represents monolithic architecture, whereas several processes are involved in the microservices architecture. Microservices developed in isolation; instead, a monolithic system is developed as a whole application with classes, functions, and namespaces (El Akhdar et al., 2024). On the other hand, in a microservice architecture, only the affected microservice needs to be updated and redeployed with no impact on others (Su et al., 2024).

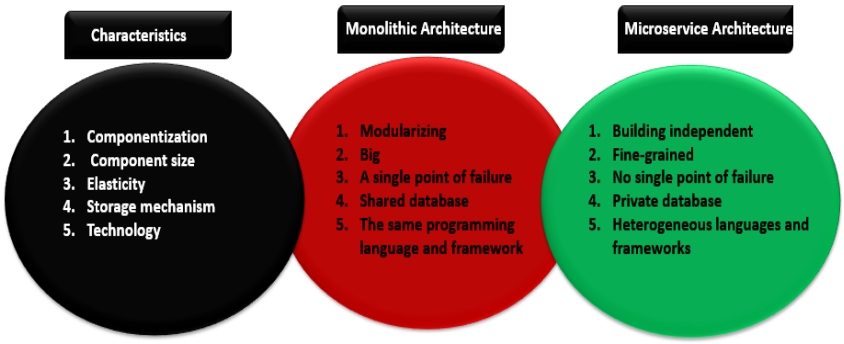

Figure 3: The Technical Comparison between Monolith and Microservice

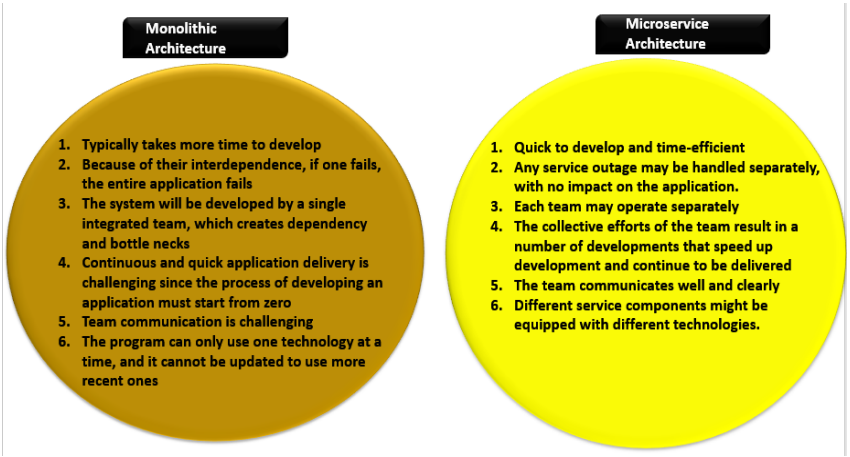

Figure 4: Approaches to Software Development: Monolithic and Microservices

2.4 Migration to Microservices

Migrating to microservices is a common strategy for organizations embracing change in the software architecture to enhance agility and efficiency in the development of software. Some studies have looked into the changes from monolithic to microservice architectures. There is evidence that input data such as source code, logs, execution traces, and use cases help in breaking down applications into microservices. These techniques are primarily useful when developing new systems or during a change from a monolithic to a microservices-based architecture (Gill et al., 2025). The process of migration into microservices is strategic and transformative for improvement in agility, scalability, and maintainability of the software system. The paradigm shift is the decomposition of a large, monolithic application into smaller pieces of loosely coupled services with each accountable for delivering business-specific functionalities (Indrasiri & Siriwardena., 2018). The assessment involves breaking the existing monolith to identify its important functionalities and dependencies. An architecture therefore about microservice is developed according to the needs of a specific organisation. This process transition also includes containerization and orchestration, as a modern development practice for simplifying

the deployment of microservices and management. Along with technical changes, the process also ensures cross-functional collaboration, continuous integration, and autonomous teams; however, the migration process itself is complex and doesn't have a structured framework (Hutcheson et al., 2024). To successfully migrate to microservices, a comprehensive understanding of the migration journey is necessary (Newman, 2015) . The migration process involves both technical changes and long-term systemic changes. It is important to consider both the business and technical aspects of the architecture. Setting up supporting artefacts and changing the software development paradigm are key components of the technical migration.

3. Related Work

Our research centres on the transition to microservices and the reasons that drive individuals to embrace this. We conduct an analysis of reviews to generalize features and metrics applied in a comparison of monolithic and microservice-based systems. The contribution of our work is therefore to confirm or refute these findings and thereby provide valuable insights for practitioners and academics alike. Table 1 Summarized key results which have been discussed in this section. A study in Tapia et al. (2020) compared microservices against monolithic designs for a web application and concluded that, in terms of hardware resource usage, cost minimization, and productivity, microservices surpass the monolithic designs. The study utilized computer-related metrics such as CPU, disk speeds, memory, and network reception to prove that microservices outperform monolithic architectures. However, this study addresses only one quality attribute and excludes the others, like security, and is not related to cyber-attacks against microservices.

In Li et al. (2021), a work is focussed towards underlining the fact that the microservices provide agile, reduced developmental time, scalability and flexibility in terms of choice of technology. However, the organizations must make their resources adaptable enough to utilize these strengths to the utmost. In Bushong et al. (2021), the authors categorized approaches and strategies for the analysis of microservice systems during 2018-2021. The research offers a guideline in making a decision concerning the analysis of cloud-native systems and discovers security, performance, and maintainability as significant quality aspects. For scalability and maintainability, the research demonstrates static code analysis, case studies, and dynamic analysis, yet it requires more extensive evaluations of decentralized systems.

In Aksakalli et al. (2021), the review of deployment and communication in microservice architectures was also done, identifying three approaches to the deployment approach, and seven communication patterns, a total of eight challenges during deployment, and six challenges during communication, pleading for further research in topics such as complexity management, monitoring, and security on microservices. Quality assurance of some of the identified issues of microservices entails service discovery, data consistency, performance prediction, testing, and security. Some of the suggested solutions proposed by this study include information-centric networking to perform service discovery, multi-agent architecture to manage distributed transaction coordination, as well as automated regression testing to predict performance.

In the study (Capuano & Muccini, 2022), it was attempted here to elaborate on how quality attribute improvement drives migration to microservices. After systematically reviewing 58 research papers, it has been realized that coupling, cohesion-related attributes, scalability, and performance are related to migration phases. However, the present study doesn't involve any discussion in relation to the kind of variations the said attributes cause in different types of migration or their measurability at the different migration stages. Finally, (Abdelfattah & Cerny, 2023) comprises the review of the system analysis approaches along with their relevance to automated or human-centred assessment in microservice systems. The research conducted introduces an intermediate system representation that decouples the phases of processes that lend to giving different perspectives in evaluating system quality like performance, security, and fault tolerance. These studies have informed our understanding of the microservices as well as informing the development of our evaluation metrics, as shown in Table 1.

Table 1: List of the Related Reviews

Ref.	Type	Year	Publisher	Focus	Common Challenges Papers with SLR	
(Li et al., 2021)	SLR	2020	Science Direct	Introducing the Quality Attributes of Microservice Architecture	21	yes
(Tapia et al., 2020)	Opinion Paper	2020	MDPI Applied Sciences	Comparing Microservices Vs Monolithic	8	yes
(Bushong et al., 2021)	Survey	2021	IEEE	Analysing Microservice-Based Systems	6	yes
(Aksakalli et al., 2021)	SLR	2021	Science Direct	Discussing the Deployment and Communication in MSA	22	yes
(Söylemez et al., 2024)	SLR	2022	MDPI Applied Sciences	Challenges of QA in MSA	40	yes
(Capuano & Muccini, 2022)	SLR	2022	IEEE	The QA Contributes to the Improvement of the Migration Process	8	yes
(Abdelfattah & Cerny, 2023)	Rapid Review	2023	MDPI	The QA Helps Improve the Migrating Process	35	yes

4. Methodology

This paper uses the three stages, planning, reviewing, and reporting, of the Kitchenham methodology for conducting a literature survey (Abdelfattah & Cerny, 2023; Kitchenham et al., 2009). The review protocol is developed under the planning phase, while the need for conducting a review is identified. The primary studies selection, review, data extraction, and synthesis are within the scope of the reviewing phase of the methodology as discussed in Section 4.2 and Section 4.3. The final report writing phase includes recording the review, which is made up of observation of the documents and reporting of the results, all of which will be described in Section 5. Figure 5 depicts the key steps of the SLR process. The main difference is that any SLR is distinguished from a traditional literature review in the fact that the search process will be more exhaustive. An SLR search consists of three phases: manual, automatic,

and snowballing for the sake of collecting as much relevant literature as possible (ElGheriani, 2022). Literature review is regarded as one of the basic building blocks of any kind of research study; all the necessary contexts, relevance, and backgrounds that relate to the given research problem being investigated.

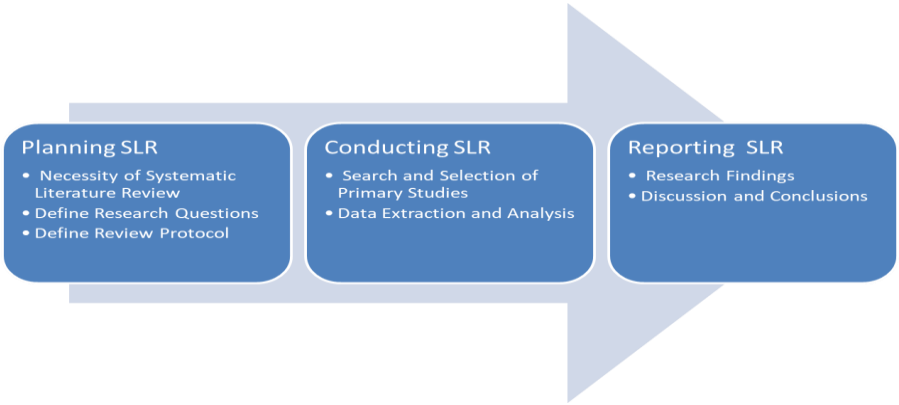

Figure 5: Systematic Literature Review (SLR) Process

4.1 Planning the Survey

Planning involved verification of the motivation of research, leading to four RQs as follows:

RQ1: Most critical quality attributes to be addressed during applications' migration from monolithic architecture to microservices. The question then focuses on identifying the most dominant quality attributes that lead the migration towards the microservice architecture; further information provides insight into the motivations for their transformation.

RQ2: What is the impact of migration on varying application quality attributes? Answer This question explains what impact the migration process has on different quality attributes to put a spotlight on various considerations when transitioning to a microservices design.

RQ3: Compare the monolithic and microservice quality attributes of such implementations. This question compares quality attributes of both monolithic and microservice architectures by delineating differences in design, development, and deployment that impact these attributes and induce the transition.

RQ4: What are the differences in metrics between monolithic and microservice quality attributes? This question identifies the differences in the quality attributes of the microservices as compared to the quality attributes of monolithic architecture, focusing on the different metrics and formulas used to determine the performance of each system.

4.2 Selection of Primary Studies

All full-text papers were downloaded to conduct the review process, and they were read and analysed comprehensively. The results are then classified with other appropriate criteria and discuss with regards to others' evaluation results. Any doubts and contradictions were cleared, and finally, the final results were presented. Multiple

quality criteria are utilized in the form of metrics with respect to monolithic and microservices implementation. Since multiple features exist, the most potential ones have been prioritized for application and excluded less important ones to maintain feasibility. Before undertaking a quantitative analysis of the quality criteria adopted for implementing measurement, we established a number of traits to be utilized in assessment and comparison. The period from 1 January 2020 to 20 July 2023, having been chosen for review, gave us a current evaluation of research status within the field. Reviews published within the last few years have highlighted the latest findings and innovations; therefore, data will still be very current and up-to-date. The manual search began in January 2020. For selecting reputable journals and conference papers, five major publishers' digital library portals were used: ACM Digital Library, IEEE Xplore, ScienceDirect, Wiley, and SpringerLink, as depicted in Table 2.

Table 2: URL with Databases used for Identifying Primary Sources

URL	Database
http://www.sciencedirect.com	Science Direct
http://dl.acm.org	ACM Digital Library
http://www.springerlink.com	SpringerLink
https://onlinelibrary.wiley.com	Wiley
http://ieeexplore.ieee.org	IEEE Xplore

4.2.1 Inclusion Criteria

- The journal or conference in which the paper was published is peer-reviewed.
- Paper is on monolithic and microservice architecture.
- Quality assessment in the paper focuses on primary studies.
- Studies justify impact on specific QAs of microservice and monolithic architectures.
- Studies are based on ISO/IEC 25010.
- In reality, according to studies, proof on the impact of microservices and monolithic designs on certain quality attributes is available.

4.2.2 Quality Evaluation

We examine each of the chosen studies on whether it meets the requirements and addresses one or more of the research questions. In case of contradiction, the results are shown, and a conclusion is drawn. In order to effectively obtain and store data, a form is applied as summarized in Table 3.

Table 3: Quality Assessment Checklist

No.	Assessment Question
Q1	Does the study clearly state its research objectives?
Q2	Any insights or recommendations of the study toward future directions or practical applications?
Q3	Have the research measurements used in the work been well defined and aligned with the objectives of the research?
Q4	Is the research methodology well-articulated and presented?
Q5	Were the principal findings of the study presented clearly in terms of their validity and reliability?

Moreover, each of the quality criteria responses was rated as No (0), Yes (1), or partially (0.5). Although the final selection of the primary studies was subjected to a cut set of the criteria, each was assessed and scored individually on how effectively they matched the stated research objectives as depicted in Table 4.

Table 4: Publication Sources Searched

Source	Studies Initially Retrieved	Studies After Applying Exclusion/Quality Criteria
Wiley	232	7
Springer	315	5
ACM	450	8
Science Direct	978	16
IEEE Xplore	255	12
Total	2230	48

This study focuses on analysing the performance of monolithic and microservice architectures along with different approaches towards quality. From already existing literature, we have identified quality attributes that need to be considered during the evaluation of an application based on a microservice. Thus, an architecture analysis follows that analyses microservices in more detail and how various methodologies impact their nature. Thus, we compare the value of each characteristic across the various implementations after evaluating the quality characteristics of various monolithic and microservice-based systems. The references list comprises selection of scholarly papers as selected in Figure 6.

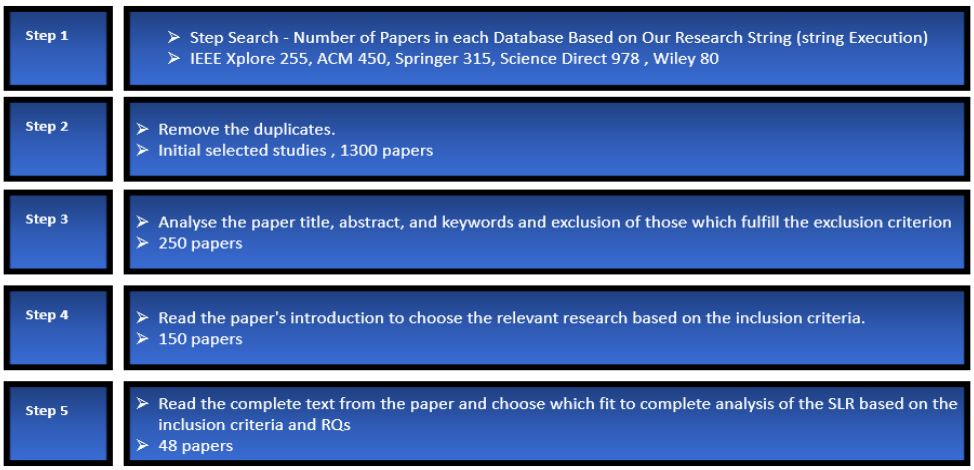

Figure 6: Study Identification Process

To answer the RQs presented in Section 4.1, we determined a set of data extraction criteria. Data items refer to specific information extracted from each primary study directly in line with the RQs. In order to answer our research questions, we applied classification criteria to each paper which include the metrics and quality attributes studied or analysed as highlighted in Table 5.

Table 5: Relevant Data Items Extracted from the Selected Primary Studies

Data item	Related RQ	Description
Publication's Year	Main Information, N/A	Temporal Information of Each Study
Study Title	Main Information, N/A	Full Title of Primary Study
Publication's Venue	RQ3	Name of the Journal, Conference, Workshop, Book, symposium, and Magazine. J = Journal, C = Conference, W = Workshop, WO = Wiley Online, SD = Science Direct
The Method used to Choose QA	RQ1,RQ2,RQ3	The Method When was the Quality Attributes Identified
Quality Attributes and Sub Attributes	RQ1,RQ2,RQ3	Identify Quality Attributes

4.3 Analysing and Synthesizing Data

We analysed the QA results by values, meaning how many times each QA is referenced in the literature, which is counted as a factor value for each paper. This helps in the determination of the factors' significance and the literature that focuses on QA (Ghayyur et al., 2018). Some observations on the data as a whole. A total of 2230 publications was found through database searches and backward and forward snowballing. Removing duplicates leaves a total of 1662 results of which 48 publications were selected according to inclusion and exclusion criteria, determined either by the title and abstract or by further content analysis. We closely read all of these 48 articles so that we could answer data extraction questions. Of the 48 publications, 36 were journal papers (Springer Link =5, ACM =8, IEEE =12, Science Direct =16, Wiley =7), and 12 were conference papers, as shown in Figure 7.

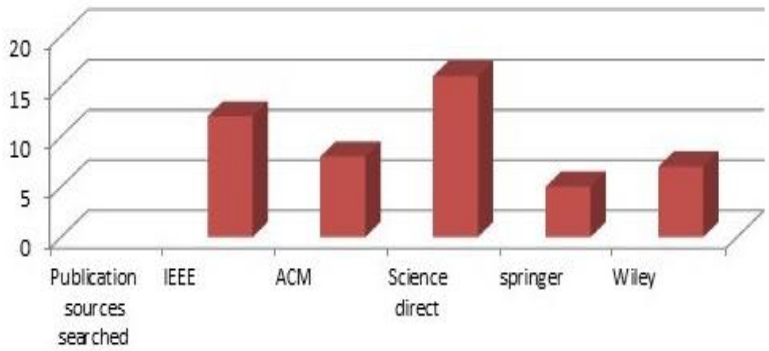

Figure 7: Classification of the Primary Studies in Accordance with the Publisher

Figure 8 Illustration of the list of primary studies found during the course of this systematic literature review; we illustrate the count of primary studies by year. Figure 6 Primary Studies by Year from the figure above, it seems that there is great interest in the subject since the same year MSA was actually proposed, namely 2020. The largest amount of primary studies was published in 2023, demonstrating the high degree of interest the research community has shown concerning this emerging topic-microservice architecture. This number becomes substantially higher when conference research is included in this count, which reflects the lags in the publication

of such studies. The results demonstrate that conferences (n = 12, 25) and journals (n = 36, 75) are the most common venues for publishing pertinent studies, as depicted in Figure 9.

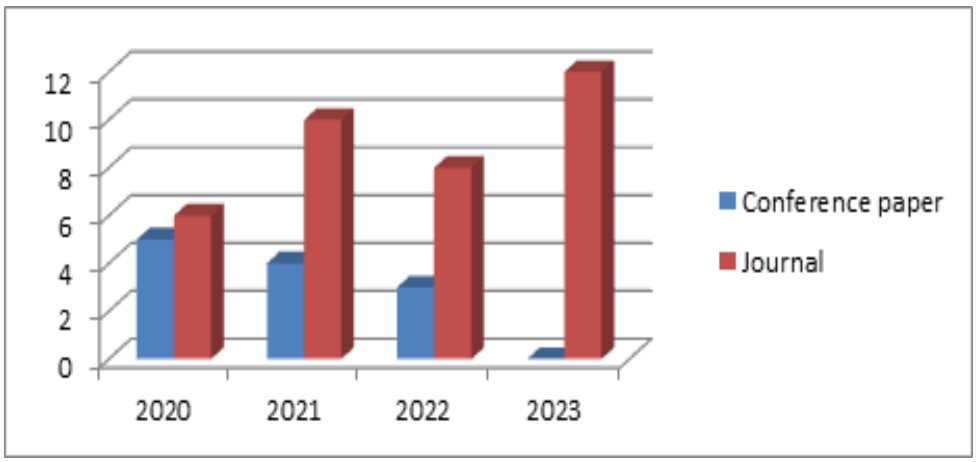

Figure 8: Year-Wise Distribution of the Number of Primary Studies

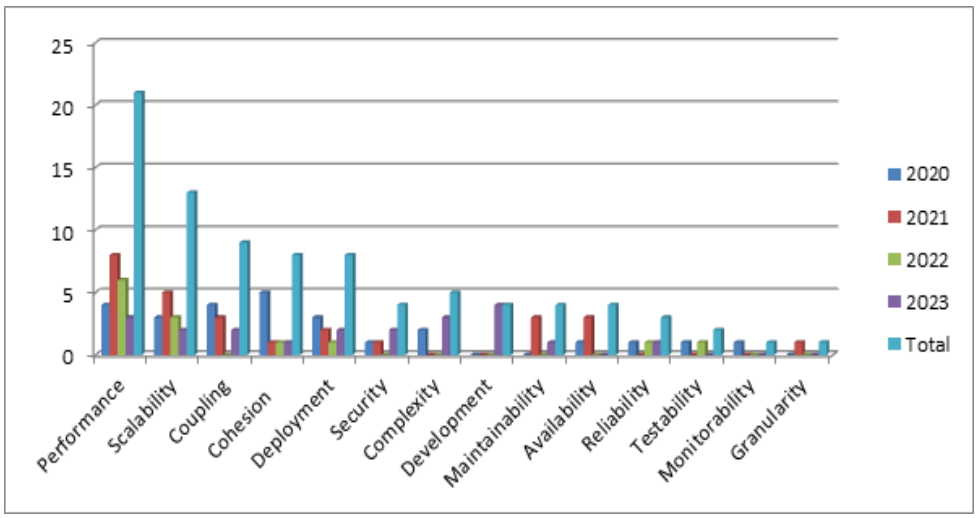

Figure 9: Amount of Papers Per Addressed Quality Attribute, Sub-Characteristics, and Year

5. Research Questions and Findings

ISO standard Liu et al. (2015) outlines a product quality model that can be utilized as a global standard for the definition of software quality attributes. As such, we selected those attributes as our initial pool upon which to base our selections. Now, from this pool, we will decide which of the qualities we shall use as our quality metrics in our empirical study. The quality model provided by the ISO offers a benchmark for the evaluation of a general application quality. Although our research mainly examines the impact of microservice architecture on these attributes, studying ISO's guidelines

on how to assess them will go deep in giving a fuller understanding of how one might effectively measure the quality criteria we have chosen, as shown in Figure 10.

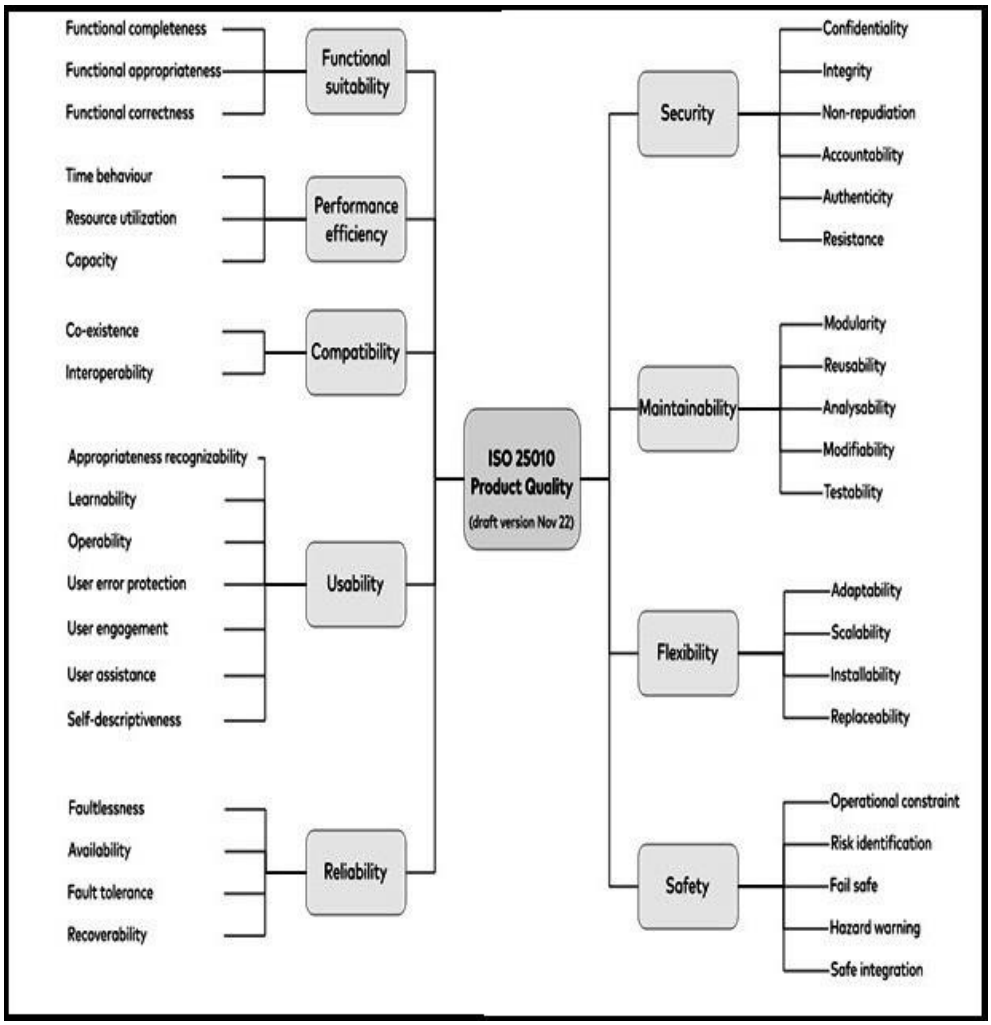

Figure 10: Details of ISO 25010 Draft 2022

Quality-in-use characteristics refer to such measurable factors that can be applied to the assessment of the quality of software and point toward the development process wherein potential problems may have been occurring 42. Indeed, such characteristics may prove whether or not the software product meets the requirements and expectations of relevant stakeholders (Souza-Pereira et al., 2022). For this purpose, we consulted a public research database to gather relevant literature for this paper. These databases allowed for a deeply advanced searching functionality, filtering important papers, and the emphasis of certain keywords by their appearance in titles or in text. Keywords like "microservices" combined with different quality attributes led us to important papers in our study. In conducting literature search, we started by opting for a reference as our starting point in the review. That required

finding the most referenced paper available using the search phrases "microservices," "quality attributes," and "migration" (see Table 6).

Table 6: Quality Attributes Were Discussed in the Selected Papers

Year	Performance	Scalability	Coupling	Cohesion	Deployment	Security	Complexity	Development	Maintainabili	Availability	Reliability	Testability	Monitorabilit	Granularity
2020	4	3	4	5	3	1	2	0	0	1	1	1	1	0
2021	8	5	3	1	2	1	0	0	3	3	0	0	0	1
2022	6	3	0	1	1	0	0	0	0	0	1	1	0	0
2023	3	2	2	1	2	2	3	4	1	0	1	0	0	0
Total	21	13	9	8	8	4	5	4	4	4	3	2	1	1

5.1 Findings of RQ1

Several quality attributes play a cardinal role while moving applications from monolithic to micro service-based architecture. For example, the quality attributes and their corresponding sub-characteristics that need to be taken into account in making this transition involve performance, scalability, coupling, cohesion, deployment, security, development, complexity, maintainability, and availability. Table 7 summarizes the quality attributes considered in the selected papers.

Table 7: Quality Attributes Were Discussed in the Selected Papers

QA and sub Characters	Performance	Scalability	Coupling	Cohesion	Deployment	Security	Development	Complexity Maintainability
Study	3 4 7 10 11 13 14 49 43 53 58 59 64 57 56 60 66 65 55 32 67	3 4 12 13 14 16 49 53 58 56 65 55 62	1 5 6 48 54 47 52 60 61	1 5 11 27 48 47 60 61	7 8 14 49 27 55 62 63	4 46 50 51 56	47 28 57	2 3 14 52 27 46 28 61 61

5.2 Performance

They allow for finer-grained scaling, meaning that all the services could scale independently so that only the required microservices were scaled in, thereby optimizing resource allocation. In a microservices architecture, all services need to be loosely coupled; therefore, in-memory calls can no longer be viable for communication. Performance is generally regarded as a mixture of response time and throughput (Estdale & Georgiadou, 2018). Throughput is defined as the number of requests in a time unit for which a microservice is able to process them, with determination being dependent upon the used technology in development, as well as by the extent of the internal optimization; this can be evaluated at runtime. Regarding workload, throughput may be determined through the use of the longest synchronous

call or the average size of messages on asynchronous calls (Hossen et al., 2022).

Of course, there are many performance metrics to be considered while migrating from monolithic architecture to microservice architecture. Again, the following factors affect the performance of microservices: response time, utilization of CPU, programming language, the path length, usage of containers, and wait time. Response time is defined as the lag between requests and responses. CPU utilization is measured as the percentage of CPU activity, excluding idle time. The type of programming language affects network performance. Path length is the number of CPU instructions executed, and waiting time refers to the processing time that occurs in the service queues (Chen et al., 2017). While there are potential challenges in performance during the migration to microservices, strategic planning, optimization efforts, and the utilization of appropriate technologies can help mitigate these challenges and lead to a more scalable and resilient system, in the long run (Eyitemi & Reiff-Marganiec, 2020; Kalia et al., 2021). ISO 25010 attributes, especially those related to performance efficiency, provide a comprehensive framework for evaluating and ensuring the optimal performance of a system. Throughput, response time, capacity, and resource utilization are directly related to how well a system performs its functions, handles user interactions, and manages its resources. By considering these attributes, developers and testers can assess, measure, and improve the performance of software systems throughout their lifecycle (Estdale & Georgiadou, 2018).

5.3 Scalability

Scalability is a crucial factor in migrating from a monolithic to a microservice-based Scalability architecture. It allows for improved horizontal scalability, isolation of scaling concerns, flexibility in the technology stack, elastic and dynamic scaling, improved response to load variability, fine-grained resource allocation, fault isolation, decomposition of monolithic bottlenecks, and optimized resource utilization. Microservices also promote fault isolation, reduce bottlenecks in specific components, and optimize resource utilization across resources, leading to improved system performance and scalability (Al Qassem et al., 2023; Zhong et al., 2024; Jatkiewicz & Okrój, 2023). The scalability of a software system is influenced by a number of ISO 25010 properties, including those concerning resource utilization, modularity, maintainability, and performance efficiency. To be scalable, a system must be able to handle growing loads as well as adjust to changes and alterations without degrading its functionality. Developers and evaluators can evaluate and improve the scalability of software products (Liu et al., 2015).

5.4 Coupling

A software system could be composed of various components put together in modules, with components of one module able to interact with components of another module. The level of coupling in a software system defines the strength of interdependence between its components (Vale et al., 2022). In this regard, there ought to be a minimization of the level of connectivity of components and modules to ensure a modular architecture of loose coupling. Minimizing coupling is a fundamental consideration during the migration from monolithic to microservices. It allows for greater independence, flexibility, and agility in the development and deployment of individual services, contributing to a successful and efficient migration process

(Apolinário & de França, 2021; Hassan et al., 2020). Although coupling isn't specifically addressed in ISO 25010, the standard's emphasis on maintainability and modularity is in line with the controlling coupling principle of software engineering. Low coupling systems that adhere to modular and maintainable design principles are likely to display attributes like adaptability and ease of change that are consistent with ISO 25010 objectives. Coupled with the maintainability features outlined in ISO 25010, coupling principles can help developers and assessors create and evaluate software systems that are both dependable and simple to maintain (Bourque, 2000; Estdale & Georgiadou, 2018; Vale et al., 2022).

5.5 Cohesion

The concept of high cohesiveness in software design is in line with ISO 25010's emphasis on maintainability, particularly through the modularity feature. This is true even though the term "cohesion" is not used specifically in the standard. High modularity and cohesiveness, where components are arranged to be self-contained, focused, and readily maintainable, are characteristics of a well-maintained system, as defined by ISO 25010. In order to guarantee the long-term sustainability and simplicity of the maintenance of software systems, software engineers and assessors can find it beneficial to take into account cohesiveness principles in addition to the maintainability criteria outlined in ISO 25010 (Hasan et al., 2023; Sellami, Saied, & Ouni, 2022).

5.6 Deployment

Microservice architectures offer small, independent deployable services that can be developed on multiple middleware stacks and scaled independently. In contrast, the disadvantages of monolithic designs in which all of the application's logic and data are bundled into a single deployable unit highlight why microservices address those shortcomings. Further, microservices back deployments of services and related issues that have to do with organizing the development teams. The use of smaller services allows the entire structure to be understood more rapidly by new developers. In addition, every microservice can be deployed independently, which permits continuous improvement and faster updates. Technologically, separate microservices can be given to dedicated teams of development, and then those teams will be given the ability to focus solely on just one service or feature. The companies should have autonomy; therefore, teams are free to work without problems which involve other parts of an application. Each time a feature is introduced or changed in a software system, it needs to be released into the right environment that suits that deployment, which could be test, staging, or production.

Deploy ability denotes all forms of artefacts and activities required in the deployment of the software system. A deployable software system ought to be simple. In terms of deploy ability, the average build time is an essential measure. This metric measures in minutes the steps for validation, code compilation, test compilation, test execution, static code analysis, and application packaging to produce deployable artefacts. As explained by ISO (2022), microservices often take less time to develop than a monolithic application. Updating functionality occurs much more rapidly for a deployable microservice compared to a monolithic application. Strategy behind deployment would be the key to successful migration from monolithic architecture to

a microservices architecture. For this reason, containerization, orchestration, CI/CD practices, and checking the way of data migration ahead are also positive migration enablers. Modern best practices in deployment need to be adopted as points of technology would only help to reap its rewards, which are promised by the use of microservices, and thus will have a smooth journey (Söylemez et al., 2024). While ISO/IEC 25010 does not specifically define deployment, some of its quality attributes are particularly directly relevant in relation to that phase. Not only must resource management, maintainability, portability, reliability, and the ease of installation work together to help insure a smooth and efficient flow through the process of deployment, but it also must be considered by development teams and deployment teams before it may be undertaken to ensure the software product is properly deployed so that it meets the expectations of users in any number of operational environments (Chen et al., 2017; Hassan et al., 2020).

5.7 Security

In software terms, security means protecting the information and data so that access is granted on actual premises with permission from the user, system, or entity. The data dealt with is of extreme importance; therefore, the security aspect requires to be stringent. However, with this aspect, there lies a risk when putting microservices architecture into play because it depends on network communication. Because service-to-service communication typically happens through messaging mechanisms, security integration needs to be very strong and provision for confidentiality and integrity. Confidentiality can be provided for data access by just authorized people, while integrity provides immunity of data by methods like encryption. It is much more important in a microservices architecture, in which service-to-service communication happens over open networks, as the risk of potential intrusions increases. Three significant security vulnerabilities have been identified by monolithic applications: inappropriate settings for cross-site request forgery, insecure use of command-line parameters, and poor management of the roles of the users. These concerns are lesser for microservices because functionality has a specific view or web service and the role definition of users is much more explicit. However, these latent weaknesses should not be ignored during software design. Secondly, every microservice typically has its own security approach, and proper consideration must be taken in regard to problems these different methods of security may pose. Secure communication between microservices carries its load on the development process to ensure that it is secure and, therefore, makes the system complex in terms of security management (Campbell, 2018).

5.8 Maintainability

Maintainability is one of the most important aspects for a system's transition from monolithic to microservices architecture. Independent services, modularity, and incremental updates are indeed facilitated through microservices architecture; these aspects eventually support the enhancement of the maintainability and adaptability nature of a system. To effectively leverage these benefits so that the maintained is sustainable over time, careful planning, continuous monitoring, and strong focus on documentation are necessary (Chen et al., 2017; Hasan et al., 2023). All of the maintainability attributes of ISO 25010, for instance modifiability, analysability, changeability, testability, scalability, and reusability provide an overarching

framework to evaluate and ensure the maintainability of software products. They guide practice toward systems that are adaptable, understandable, and capable of accommodating change within an acceptable time frame (Hasan et al., 2023; ISO, 2022; Kalia et al., 2021).

Summary about the Response to RQ1: Out of these, it identifies 14 QAs and their associated sub qualities most relevant to MSA research. Of these QAs, performance and scalability are the two most worrisome to academics, whereas monitorability and granularity are also considerably less focused on. But the impact of quality attributes during the process of migration from monolithic architecture towards microservices is manifold. Indeed, a successful migration depends largely on the knowledge of these attributes and cautious strategic planning so that the advantages inherent to microservices, such as improved performance and scalability, and maintainability, are realized without decreasing or compromising other critical factors like security and reliability. However, getting this kind of balance between quality attributes will be necessary if the optimization and success of a microservices architecture are the objectives.

5.9 Findings of RQ2

Migration from other third-party libraries may benefit various applications through diverse quality attributes. Research indicates that changing to an alternative API of a library may reduce coupling, increase cohesiveness, and make code easier to read (Alrubaye et al., 2020). In fact, during the detection phase of the migration, one can give focus to enhancing specific quality attributes. The characteristics of virtualization and cloud-based platforms have a major influence on quality parameters such as efficiency, resource elasticity, and security. Hence, a thorough evaluation and planning for the current portfolio of applications before the database migration will be of prime importance because this may reflect upon dependent applications, thus giving rise to design-related challenges (Capuano & Muccini, 2022). Only through stiff testing of the new database platform with the migrated application may a smooth migration process be established. Microservice-based architectures allow each service to be independently developed, modified, deployed, and scaled, which increases speed, uptime, and reliability in return, thus supporting more frequent software updates. Microservices together with differing application development methodologies can greatly influence selected quality indicators, and performance will be the most significant one (Eyitemi & Reiff-Marganiec, 2020).

Microservice architecture influences QA by improving small service reliability, maintainability, performance, security, and testability. It provides modularity, reusability, and deploy ability because of loose coupling and virtualization technologies such as containerization. Independent testing of every service provides ease in scalability, rapid roll-out of new features, and flexibility. At the same time, this raises a major security concern because all services communicate over the network. However, microservices provide additional security mechanisms that supplement the advancement of QA (Glen, 2018). The transition from a monolithic to a microservice-based architecture can significantly impact application quality attributes as defined by ISO 25010. Microservices allow for greater functional modularity, potentially improving functional suitability by enabling independent development, deployment, and scaling of services. They can enhance performance efficiency by enabling

independent scaling of services, but increased inter-service communication may introduce latency and impact overall performance. Compatibility may be affected due to changes in data storage, communication protocols, and interfaces. Changes in the architecture may impact usability, reliability, security, maintainability, portability, scalability, reusability, testability, deploy- ability, operability, and security usability. To ensure a smooth and successful migration, careful planning, consideration of architectural choices, and adoption of best practices in micro services development and operation are essential (Bajaj et al., 2020; Capuano & Muccini, 2022; Mazzara et al., 2018; Michael Ayas et al., 2023).

Summary about the Response to RQ2: The change from monolithic to micro service architecture has both positive effects on various application quality attributes and overall design quality. Though performance, easier scalability, and faster release have always been at the heart of mainstream motivators for migration, some other factors influence the shift toward microservices. These include maintainability, the natural development of loosely coupled services, and the encouragement of coherent services to ease reuse as well as make users more agile.

5.10 Findings of RQ3

Understanding the way different architectural mechanisms impact the quality of an application will stand as a necessity for proper implementation of micro service architecture. This research will thus attempt to demonstrate the comparison that exists between monolithic and micro service applications, in terms of just how much they contrast with each other, as shown in Table 8.

Table 8. Comparison Summary

Performance	In microservice architecture, each service is therefore designed for its optimality and best performance with respect to its internal workings, leading to greater overall performance and resource utilization.	In a monolithic architecture, the performance of the whole system is interdependent, in that the performance of the whole system suffers.
Complexity	Microservices focus upon the integration of one single feature and reduces the complexity of monolithic applications by breaking them into smaller, manageable services. However, the development activities like testing can be more complex as a result (Campbell, 2018; Hassan et al., 2020; Valdivia et al., 2019).	The monolithic application will have everything in one structure, which is very complex especially about complexity (Campbell, 2018; Hassan et al., 2020).
Size	Microservices are small, function-oriented, independent services that mainly focus on self-management and lightweight design with the aim of improving the agility, scalability, and autonomy of software. The size and structure are the key differences between monolithic and microservice architecture. For example, the services in microservice architecture are small in size and loosely coupled, allowing them to be developed, deployed, and scaled	The term monolithic architecture refers to the traditional approach of developing an application as one large and tightly integrated unit. Such architectures tend to be large and complex, making it challenging to scale and maintain the applications (Ma et al., 2022).

	independently. The nature of modularity offers more flexibility, faster delivery, and greater scalability (Li et al., 2021; Ma et al., 2022).	
Agility	The application has been divided into several independent services. So, the changes applied to one service will not affect other parts of the system. It also allows the applications to be released faster as well as the problems get resolved quicker because a change applied to a service will not influence the whole application (Andrade et al., 2022).	Because of the strong coupling among modules that operate together, changing the code or function requires extensive testing (De Lauretis, 2019).
Resilience	The whole application has been divided into several independent services, and each is maintained by its own database so that the failure of one service does not influence other services. In addition, this modular design also facilitates system availability maintenance while individual services are in the process of being deployed or updated.	Interconnection is strong, so it is hard to reach resilience. The application works upon a single codebase with cautious planning (Andrade et al., 2022).
Flexibility	This architecture, therefore, offers flexibility; teams can choose the right tools for a specific task, leverage available expertise, and incorporate new technologies into the system without disturbing the whole system (Capuano & Muccini, 2022; Valdivia et al., 2019).	This makes the monolithic architecture more rigid as compared to the microservices architecture. Monolithic systems are defined as being in single codebase and tightly coupled components which makes it increasingly difficult to scale and maintain the application as it grows (Capuano & Muccini, 2022).
Deployment	In the case of a microservices architecture, services can be deployed independently. Therefore, there is a lower chance of errors, and deployment is faster. More importantly, each microservice can be deployed and rolled back individually, thus allowing for rapid and agile development and deployment (Bandara & Perera, 2020; Esposito et al., 2016; Selmadji et al., 2020).	In a monolithic architecture, one needs to deploy the whole application together at one time, which in itself is very cumbersome and also exposes such systems to higher risks of deploying errors. This is a one-time deployment, which in itself incorporates upgrades, updates, and patches and, therefore, constitutes a relatively long time from the initial development phase (Bandara & Perera, 2020).
Coupling	Microservices architecture therefore supports lower coupling as it breaks up the system into loosely coupled, autonomous services. The services of microservices architecture are therefore independent. That will enable the growth and evolution of individual services. This	Such are the differences between microservices and monolithic architectures along the dimension of coupling. In a monolithic architecture, the system is unified because its modules are tightly connected

	decoupling makes it easy for microservices to be constructed, tested, and then deployed one by one. Transitioning from monolithic architecture to microservices architecture is demanding due to the structural complexity and the reliance on third-party framework libraries. In general, more decoupling and flexibility of a system can be achieved with the microservices design in comparison to the monolithic architecture (Capuano & Muccini, 2022; Taibi & Systä, 2020; Wei et al., 2020).	leading to tight coupling. Monolithic applications consist of interdependent modules that cannot be extended independently and are subject to tying the same technology stack (Wei et al., 2020; Zhang et al., 2020).
Cohesion	Microservices are more cohesive because they strongly support modularity and separation of concerns. This makes it possible for the microservice to concentrate on one specific task, which offers increased coherence and specialization of services (Sellami, Saied, Ouni, et al., 2022; Taibi & Systä, 2020; Zhang et al., 2020).	Monolithic architecture integrates everything in one system. Because all the features are quite tightly coupled together in the same structure, monolithic designs generally tend to have lesser cohesion. It encourages strong dependencies and interactions among different components that eventually minimize overall cohesion (Sellami, Saied, Ouni, et al., 2022; Wei et al., 2020; Zhang et al., 2020).
Technologies	DevOps, Docker, Kubernetes, Lambda, Java, Python (Capuano & Muccini, 2022).	NET, Java, PHP, or Ruby, Python/D-Jango (Capuano & Muccini, 2022; Harris, 2023).
Reusable	Software can be divided into clear, definable modules and functionalities that teams can exploit for all sorts of purposes. A feature may be based upon an existing service originally developed for a different purpose. This modular approach enables developers to add new features without having to start from scratch, allowing the application to self-bootstrap (Schneider & Scandariato, 2023).	As the application size increases along with complexity, it is very hard to manage and maintain a codebase. Implementation and testing might also be more challenging.

ISO 25010 quality attributes guide the comparison of monolithic and microservice architectures. Monolithic architectures offer simplicity in design and development, while microservices offer greater modularity and scalability. They both have their strengths and challenges, and the choice should align with the project's requirements and goals. Monolithic architectures have a single code base, which can lead to less inter-process communication overhead, while microservices have a unified code base that simplifies compatibility. The microservice has usability, reliability, security, maintainability, portability, scalability, and reusability. Monolithic architectures offer centralized management and a single codebase, while microservices offer modularity

and scalability. Both have their strengths and challenges, and the decision should align with the project's requirements and goals. The ISO 25010 quality attributes serve as a useful framework for evaluating and comparing the effectiveness of these architectures in different contexts (Milić & Makajić-Nikolić, 2022).

Summary of RQ3 Reply: Monolithic Vs Microservice Architecture: The characteristics and trade-offs based on quality attributes are much different. The key is in how each one determines the overall system attributes. Here, in monolithic architecture, each quality attribute is locked within itself, and so changing one attribute might be bound to affect others. In contrast, quality attributes in microservice architecture are decoupled, and thus a modification in any one does not affect the others.

5.11 Findings of RQ4

MSA metrics provide an objective means by which architects and developers can assess architectural quality. They are useful in diagnosing defects or errors, indicating possible improvement areas, and can be used when prioritizing quality attributes. The phases and methodologies to be followed in identifying the quality attributes together with the metrics to be applied to measure them were determined through preliminary studies such as shown in Figure 11.

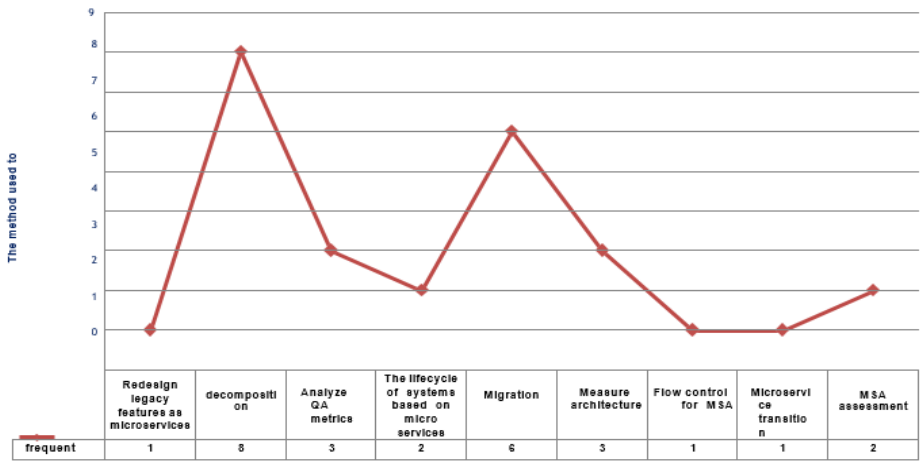

Figure 11. The Method used to Choose QA

Depending on the specific aims of each investigation, various metrics were employed. As indicated in the table below, we identified single attribute metrics and those that measure multiple quality attributes, which are aggregated in the research paper (see Table 9).

Table 9. Comparison Summary

Quality Attributes	Quality Metrics for Monolithic	Quality Metrics for Microservices
Performance (Kalia et al., 2021; Liu et al., 2020)	Accuracy (A), Timeliness(T), Precision (P), Response Time (R), Throughput (TP), Latency (L)	Accuracy (A), Timeliness (T), Precision (P), Modularity Quality (MQ), Percentage of Runtime Calls (PRC), Average Entropy (AE), the Number of Interfaces in a Microservice (NOI), Size of a

		Microservice (SM), Optimum Resource Allocation (ORA), Accuracy(A), Probability (PB), Mean Absolute Error(AE), Root Mean Squared Error(RMSE)
Performance, Cohesion(Sellami, Saied, Ouni, et al., 2022)	Overall Monolithic System, Score (OMSS), Normalized Cohesion Metric(NCM)	Semantic Similarity (SS), Structural Modularity (SM), Class Similarity (CS), Overall Monolithic System Score (OMSS), Interface Number IN, Non-Extreme Distribution (NED), Inter Call Percentage (ICP)
Coupling, Cohesion(Wei et al., 2020)	Normalized Cohesion Metric (RFC)	Coupling Cohesion Network Feature(CCN), Overhead Modularization (OM), External Coupling (EC), Composability (CPB), Fitness function to measure service quality (FFMS), the Controller Objects (CO), Subordinate object (SO)
		Percentage of Calls(PC)
Cohesion (Sellami, Saied, Ouni, et al., 2022; Taibi & Systä, 2020; Zhang et al., 2020)	Lack of Cohesion in Methods (LCOM)	Service Interface Data Cohesion (SIDC), Service Interface Usage Cohesion (SIUC), Microservice Cohesion (MC)
Coupling, Maintainability (Apolinário & de França, 2021)	(OMDM)=(Normalized Coupling Metric Normalized Maintainability Metric)/2	Service Importance Distribution (SID), Service Dependency Distribution (SDD), Service Coupling Factor (SCF), Average Number of Directly Connected services (ACS)
Complexity (Campbell, 2018; Hassan et al., 2020; Valdivia et al., 2019)	Class Count (CLS), Monolithic Complexity Metric (MCM)=(Normalized Codebase Size+ Metric Normalized Code Structure +Metric Normalized Dependency Metric)/3	Class Count (CLS), Microservices Complexity Metric (MCM)=(Normalized Dependency Metric Normalized Intercommunication Metric Normalized Service Size Metric)/3
Performance, Deployment (Bandara & Perera, 2020)	Monolithic Performance and Deployment Metric (MPDM) =(Normalized Throughput Metrics Normalized Latency Metrics- Normalized Deployment Overhead Metrics)/3	Response Time (RT), Costs(CTS), Microservice Performance and Deployment Metrics (MPDM) =(Normalized Throughput Metrics Normalized Latency Metrics- Normalized Deployment Overhead Metrics)/3
Development, Maintainability, Testability, Deployment, Portability, Reliability, Scale Ability, Performance (Bandara & Perera, 2020; Esposito et al., 2016; Selmadji et al., 2020)		Accuracy, Timeliness, Precision
Deployment		Focused on One Function, the Structural and Behavioural Autonomy, Internal and External

		coupling, Coupling Between Microservice (CBM), Structural Coupling, Coupling of Service (COS)
Efficiency, Satisfaction (Souza-Pereira et al., 2022)		Task Time(TT), Psychometric Scale Value (PSV), Response to a Question(RTQ)
Scalability (Zhong et al., 2024)	Time of Convergence to an Adaptation Decision (TAAD)	Time of Convergence to an Adaptation Decision (TAAD)
Security, Performance (Kalia et al., 2021;Liu et al., 2015)		Precision (P)
Functional , Suitability (Alshuqayran et al., 2016;Liu et al., 2015)	Functional Appropriateness Measures (FAM)	Functional Correctness Measures (FCM)
Flexibility (Hossain et al., 2023)	Coupling Measures (CPM)	
Security (Minna & Massacci, 2023)	Confidentiality Measures (CM) Integrity Measures(IM)	
Size (Kalia et al., 2021)	Number of Operations (NO), Number of Services (NS)	

The quality attribute applied as an indicator for the evaluation of implementations in monolithic and microservices is considered to be vital to help in prioritizing a large number of quality attributes by the system. Various factors which decide the performance of microservices include path length, container utilization, programming language, CPU utilization, and response time. Response time pertains to the period that elapses between the request submission and the response. CPU utilization pertains to the percentage non-idle CPU time. This means the language of the program will make a difference on the network performance due to the difference in the communication protocols.

6. Conclusion

This research has demonstrated that quality attribute metrics enable a more accurate assessment of monolithic and microservices systems. As the structured literature review is done, performance and scalability would appear to be major concerns, but while monitorability and granularity are not viewed as critical ones. Improving the quality of software becomes a motivation and enhances design, which makes adopting the microservices approach encourage quality driven development. A quality-based evaluation will be proposed to help companies evaluate the value of migration to microservices, thereby enhancing their adaptability in response to changing technological and business needs. Findings form a basis for further research on non-functional requirements and show that there is a need for additional assessment criteria, different quality models, and empirical studies validating efficiency and scalability. Then, quality attributes in microservices will be further fine-tuned through the evaluation of different real-world scenarios in the future support of academic as well as industry efforts that focus on quality in software engineering.

References

Abdelfattah, A. S., & Cerny, T. (2023). Roadmap to reasoning in microservice systems: a rapid review. *Applied sciences*, *13*(3), 1838. https://doi.org/10.3390/app13031838

Aggarwal, A., & Singh, V. (2024). Migration aspects from monolith to distributed systems using software code build and deployment time and latency perspective. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, *22*(4), 854-860. http://doi.org/10.12928/telkomnika.v22i4.25655

Aksakalli, I. K., Çelik, T., Can, A. B., & Tekinerdoğan, B. (2021). Deployment and communication patterns in microservice architectures: A systematic literature review. *Journal of Systems and Software*, *180*, 111014. https://doi.org/10.1016/j.jss.2021.111014

Al-Debagy, O., & Martinek, P. (2018). A comparative review of microservices and monolithic architectures. In *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)* (pp. 000149-000154). IEEE. https://doi.org/10.1109/CINTI.2018.8928192

Al Qassem, L. M., Stouraitis, T., Damiani, E., & Elfadel, I. A. M. (2023). Proactive random-forest autoscaler for microservice resource allocation. *IEEE Access*, *11*, 2570-2585. https://doi.org/10.1109/ACCESS.2023.3234021

Alrubaye, H., Alshoaibi, D., Alomar, E., Mkaouer, M. W., & Ouni, A. (2020). How does library migration impact software quality and comprehension? an empirical study. International Conference on Software and Software Reuse, Springer. https://doi.org/10.1007/978-3-030-64694-3_15

Alshuqayran, N., Ali, N., & Evans, R. (2016). A systematic mapping study in microservice architecture. 2016 IEEE 9th international conference on service-oriented computing and applications (SOCA), 1509047816. https://doi.org/10.1109/SOCA.2016.15

Andrade, B., Santos, S., & Silva, A. R. (2022). From monolith to microservices: Static and dynamic analysis comparison. *arXiv preprint arXiv:2204.11844*. https://doi.org/10.48550/arXiv.2204.11844

Apolinário, D. R., & de França, B. B. (2021). A method for monitoring the coupling evolution of microservice-based architectures. *Journal of the Brazilian Computer Society*, *27*(1), 17. https://doi.org/10.1186/s13173-021-00120-y

Arzo, S. T., Scotece, D., Bassoli, R., Devetsikiotis, M., Foschini, L., & Fitzek, F. H. (2024). Softwarized and containerized microservices-based network management analysis with MSN. *Computer Networks*, *254*, 110750. https://doi.org/10.1016/j.comnet.2024.110750

Bajaj, D., Bharti, U., Goel, A., & Gupta, S. (2020). Partial migration for re-architecting a cloud native monolithic application into microservices and faas. International conference on information, communication and computing technology, 111-124. https://doi.org/10.1007/978-981-15-9671-1_9

Bandara, C., & Perera, I. (2020). Transforming monolithic systems to microservices-an analysis toolkit for legacy code evaluation. 2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer), 95-100. https://doi.org/10.1109/ICTer51097.2020.9325443

Bourque, P. (2000). Guide to the software engineering body of knowledge. https://espace2.etsmtl.ca/id/eprint/20882

Bushong, V., Abdelfattah, A. S., Maruf, A. A., Das, D., Lehman, A., Jaroszewski, E., Coffey,

M., Cerny, T., Frajtak, K., & Tisnovsky, P. (2021). On microservice analysis and architecture evolution: A systematic mapping study. *Applied sciences*, *11*(17), 7856. https://doi.org/10.3390/app11177856

Campbell, G. A. (2018). Cognitive complexity: An overview and evaluation. Proceedings of the 2018 international conference on technical debt, 57-58. https://doi.org/10.1145/3194164.3194186

Capuano, R., & Muccini, H. (2022). A systematic literature review on migration to microservices: a quality attributes perspective. 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C), 120-123. https://doi.org/10.1109/ICSA-C54293.2022.00030

Chen, R., Li, S., & Li, Z. (2017). From monolith to microservices: A dataflow-driven approach. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)* (pp. 466-475). IEEE. https://doi.org/10.1109/APSEC.2017.53

De Lauretis, L. (2019). From monolithic architecture to microservices architecture. 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), 1728151384. https://doi.org/10.1109/ISSREW.2019.00050

El Akhdar, A., Baidada, C., Kartit, A., Hanine, M., García, C. O., Lara, R. G., & Ashraf, I. (2024). Exploring the Potential of Microservices in Internet of Things: A Systematic Review of Security and Prospects. *Sensors*, *24*(20), 6771. https://doi.org/10.3390/s24206771

ElGheriani, N. S. (2022). Microservices vs. Monolithic Architectures. *Al-Mansour Journal*, *37*(1), 37-44. https://journal.muc.edu.iq/index.php/journal/article/view/417

Esposito, C., Castiglione, A., & Choo, K.-K. R. (2016). Challenges in delivering software in the cloud as microservices. *IEEE Cloud Computing*, *3*(5), 10-14. https://doi.org/10.1109/MCC.2016.105

Estdale, J., & Georgiadou, E. (2018). Applying the ISO/IEC 25010 quality models to software product. Systems, Software and Services Process Improvement: 25th European Conference, EuroSPI 2018, Bilbao, Spain, September 5-7, 2018, Proceedings 25, 492-503. https://doi.org/10.1007/978-3-319-97925-0_42

Eyitemi, F.-D., & Reiff-Marganiec, S. (2020). System decomposition to optimize functionality distribution in microservices with rule based approach. 2020 IEEE International Conference on Service Oriented Systems Engineering (SOSE), 1728169720. https://doi.org/10.1109/SOSE49046.2020.00015

Faustino, D., Gonçalves, N., Portela, M., & Silva, A. R. (2024). Stepwise migration of a monolith to a microservice architecture: Performance and migration effort evaluation. *Performance Evaluation*, *164*, 102411. https://doi.org/10.1016/j.peva.2024.102411

Ghayyur, S. A. K., Razzaq, A., Ullah, S., & Ahmed, S. (2018). Matrix clustering based migration of system application to microservices architecture. *International Journal of Advanced Computer Science and Applications*, *9*(1), 284-296. https://doi.org/10.14569/IJACSA.2018.090139

Gill, S. S., Golec, M., Hu, J., Xu, M., Du, J., Wu, H., Walia, G. K., Murugesan, S. S., Ali, B., & Kumar, M. (2025). Edge AI: A taxonomy, systematic review and future directions. *Cluster Computing*, *28*(1), 1-53. https://doi.org/10.1007/s10586-024-04686-y

Glen, A. (2018). Microservices Priorities and Trends.

https://dzone.com/articles/dzone-research-microservices-priorities-and-trends

Harris, C. (2023). Microservices vs. monolithic architecture. *Retrieved May*, *20*. https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith

Hasan, M. H., Osman, M. H., Novia, I. A., & Muhammad, M. S. (2023). From Monolith to Microservice: Measuring Architecture Maintainability. *International Journal of Advanced Computer Science and Applications*, *14*(5). http://dx.doi.org/10.14569/IJACSA.2023.0140591

Hassan, S., Bahsoon, R., & Kazman, R. (2020). Microservice transition and its granularity problem: A systematic mapping study. *Software: Practice and Experience*, *50*(9), 1651-1681. https://doi.org/10.1002/spe.2869

Hossain, M. D., Sultana, T., Akhter, S., Hossain, M. I., Thu, N. T., Huynh, L. N., Lee, G.-W., & Huh, E.-N. (2023). The role of microservice approach in edge computing: Opportunities, challenges, and research directions. *ICT Express*. https://doi.org/10.1016/j.icte.2023.06.006

Hossen, M. R., Islam, M. A., & Ahmed, K. (2022). Practical efficient microservice autoscaling with QoS assurance. Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing, 240-252. https://doi.org/10.1145/3502181.3531460

Hutcheson, R., Blanchard, A., Lambaria, N., Hale, J., Kozak, D., Abdelfattah, A. S., & Cerny, T. (2024). Software Architecture Reconstruction for Microservice Systems Using Static Analysis via GraalVM Native Image. 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 9798350330663. https://doi.org/10.1109/SANER60148.2024.00008

Indrasiri, K., & Siriwardena, P. (2018). Microservices for the Enterprise. *Apress, Berkeley*, 143-148. https://doi.org/10.1007/978-1-4842-3858-5

ISO. (2022). Systems and Software Engineering—Vocabulary. https://www.iso.org/standard/71952.html

Jatkiewicz, P., & Okrój, S. (2023). Differences in performance, scalability, and cost of using microservice and monolithic architecture. Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, 1038-1041. https://doi.org/10.1145/3555776.3578725

Kalia, A. K., Xiao, J., Krishna, R., Sinha, S., Vukovic, M., & Banerjee, D. (2021). Mono2micro: a practical and effective tool for decomposing monolithic java applications to microservices. Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering, 1214-1224. https://doi.org/10.1145/3468264.3473915

Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, *51*(1), 7-15. https://doi.org/10.1016/j.infsof.2008.09.009

Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., Shan, Z., Shen, J., & Babar, M. A. (2021). Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. *Information and software technology*, *131*, 106449. https://doi.org/10.1016/j.infsof.2020.106449

Liu, D., Lung, C.-H., & Ajila, S. A. (2015). Adaptive clustering techniques for software components and architecture. 2015 IEEE 39th Annual Computer Software

and Applications Conference, 460-465. https://doi.org/10.1109/COMPSAC.2015.256

Liu, G., Huang, B., Liang, Z., Qin, M., Zhou, H., & Li, Z. (2020). Microservices: architecture, container, and challenges. 2020 IEEE 20th international conference on software quality, reliability and security companion (QRS-C), 629-635. https://doi.org/10.1109/QRS-C51114.2020.00107

Ma, S. P., Liu, I. H., Chen, C. Y., & Wang, Y. T. (2022). Version-based and risk-enabled testing, monitoring, and visualization of microservice systems. *Journal of Software: Evolution and Process*, *34*(10), e2429. https://doi.org/10.1002/smr.2429

Mazzara, M., Dragoni, N., Bucchiarone, A., Giaretta, A., Larsen, S. T., & Dustdar, S. (2018). Microservices: Migration of a mission critical system. *IEEE Transactions on Services Computing*, *14*(5), 1464-1477. https://doi.org/10.1109/TSC.2018.2889087

Michael Ayas, H., Leitner, P., & Hebig, R. (2023). An empirical study of the systemic and technical migration towards microservices. *Empirical Software Engineering*, *28*(4), 85. https://doi.org/10.1007/s10664-023-10308-9

Milić, M., & Makajić-Nikolić, D. (2022). Development of a quality-based model for software architecture optimization: a case study of monolith and microservice architectures. *Symmetry*, *14*(9), 1824. https://doi.org/10.3390/sym14091824

Minna, F., & Massacci, F. (2023). SoK: Run-time security for cloud microservices. Are we there yet? *Computers & Security*, *127*, 103119. https://doi.org/10.1016/j.cose.2023.103119

Mosleh, M., Dalili, K., & Heydari, B. (2016). Distributed or monolithic? A computational architecture decision framework. *IEEE Systems journal*, *12*(1), 125-136. https://doi.org/10.1109/JSYST.2016.2594290

Newman, S. (2015). Building microservices. O'Reilly. https://cir.nii.ac.jp/crid/1130282272986497536

Oumoussa, I., & Saidi, R. (2024). Evolution of Microservices Identification in Monolith Decomposition: A Systematic Review. *IEEE Access*. https://doi.org/10.1109/ACCESS.2024.3365079

Ponce, F., Márquez, G., & Astudillo, H. (2019). Migrating from monolithic architecture to microservices: A Rapid Review. In *2019 38th International Conference of the Chilean Computer Science Society (SCCC)* (pp. 1-7). IEEE. https://doi.org/10.1109/SCCC49216.2019.8966423

Schneider, S., & Scandariato, R. (2023). Automatic extraction of security-rich dataflow diagrams for microservice applications written in Java. *Journal of Systems and Software*, *202*, 111722. https://doi.org/10.1016/j.jss.2023.111722

Schröer, C., Wittfoth, S., & Gómez, J. M. (2021). A process model for microservices design and identification. 2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C), 1-8. https://doi.org/10.1109/ICSA-C52384.2021.00013

Sellami, K., Saied, M. A., Ouni, A., & Abdalkareem, R. (2022). Combining static and dynamic analysis to decompose monolithic application into microservices. International Conference on Service-Oriented Computing, 203-218. https://doi.org/10.1007/978-3-031-20984-0_14

Selmadji, A., Seriai, A.-D., Bouziane, H. L., Mahamane, R. O., Zaragoza, P., & Dony, C. (2020). From monolithic architecture style to microservice one based on a

semi-automatic approach. 2020 IEEE International Conference on Software Architecture (ICSA), 157-168. https://doi.org/10.1109/ICSA47634.2020.00023

Souza-Pereira, L., Pombo, N., & Ouhbi, S. (2022). Software quality: Application of a process model for quality-in-use assessment. *Journal of King Saud University-Computer and Information Sciences*, *34*(7), 4626-4634. https://doi.org/10.1016/j.jksuci.2022.03.031

Söylemez, M., Tekinerdogan, B., & Tarhan, A. K. (2024). Microservice reference architecture design: A multi-case study. *Software: Practice and Experience*, *54*(1), 58-84. https://doi.org/10.1002/spe.3241

Su, R., Li, X., & Taibi, D. (2024). From Microservice to Monolith: A Multivocal Literature Review. *Electronics*, *13*(8), 1452. https://doi.org/10.3390/electronics13081452

Taibi, D., & Systä, K. (2020). A decomposition and metric-based evaluation framework for microservices. Cloud Computing and Services Science: 9th International Conference, CLOSER 2019, Heraklion, Crete, Greece, May 2–4, 2019, Revised Selected Papers 9, 133-149. https://doi.org/10.1007/978-3-030-49432-2_7

Tapia, F., Mora, M. Á., Fuertes, W., Aules, H., Flores, E., & Toulkeridis, T. (2020). From monolithic systems to microservices: A comparative study of performance. *Applied sciences*, *10*(17), 5797. https://doi.org/10.3390/app10175797

Valdivia, J. A., Limón, X., & Cortes-Verdin, K. (2019). Quality attributes in patterns related to microservice architecture: a Systematic Literature Review. 2019 7th International Conference in Software Engineering Research and Innovation (CONISOFT), 181-190. https://doi.org/10.1109/CONISOFT.2019.00034

Vale, G., Correia, F. F., Guerra, E. M., de Oliveira Rosa, T., Fritzsch, J., & Bogner, J. (2022). Designing microservice systems using patterns: an empirical study on quality trade-offs. 2022 IEEE 19th International Conference on Software Architecture (ICSA), 69-79. https://doi.org/10.1109/ICSA53651.2022.00015

Wei, Y., Yu, Y., Pan, M., & Zhang, T. (2020). A feature table approach to decomposing monolithic applications into microservices. Proceedings of the 12th Asia-Pacific Symposium on Internetware, 21-30. https://doi.org/10.1145/3457913.3457939

Zhang, Y., Liu, B., Dai, L., Chen, K., & Cao, X. (2020). Automated microservice identification in legacy systems with functional and non-functional metrics. 2020 IEEE international conference on software architecture (ICSA), 135-145. https://doi.org/10.1109/ICSA47634.2020.00021

Zhong, C., Li, S., Huang, H., Liu, X., Chen, Z., Zhang, Y., & Zhang, H. (2024). Domain-driven design for microservices: An evidence-based investigation. IEEE Transactions on Software Engineering. https://doi.org/10.1109/TSE.2024.3385835

Ziadeh, A., & Al-Qora'n, L. F. (2024). Microservices Architecture for Improved Maintainability and Traceability in MVC-Based E-Learning Platforms: RoadMap for Future Developments. In *2024 15th International Conference on Information and Communication Systems (ICICS)* (pp. 1-6). IEEE. https://doi.org/10.1109/ICICS63486.2024.10638288