UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Viktoriia Abakumova

A Metrics-based Approach for Evaluating the Quality of Microservices-based Systems

Master's Thesis (30 ECTS)

Supervisor(s): Mohamad Gharib, PhD

Tartu 2024

A Metrics-based Approach for Evaluating the Quality of Microservices-based Systems

Abstract:

As software architecture evolves towards Microservices-based Systems (MBS), evaluating the quality of such systems remains a complex task. This thesis introduces a metrics-based approach to assessing the quality of MBS. The primary objective is to develop a set of well-defined metrics that capture the inherent characteristics of microservices, such as modularity and scalability, and align with industry standards for software quality. The research method involves a design science approach, conducting a systematic literature review to identify existing design principles for MBS, quality attributes and metrics for evaluating/assessing the identified design principles. The study progresses by defining a unique set of metrics suited for MBS (in addition to the existing metrics, several complementary metrics are also designed), which is shaped by the approach later applied to the realistic case. The results highlight the significance of each metric in contributing to a robust understanding of system quality, paving the way for future research and practical application in software development. By bridging the gap between traditional software evaluation metrics and the specific demands of microservices, this thesis provides valuable insights for developers and architects aiming to enhance the operational performance and quality of their microservices architectures.

Keywords:

Microservice-based System, Metric, Quality, Approach, Systematic Literature Review

CERCS:

P170 - Computer Science, Numerical Analysis, Systems, Control

Meetritel põhinev lähenemisviis mikroteenustel põhinevate süsteemide kvaliteedi hindamiseks

Lühikokkuvõte:

Tarkvaraarhitektuuri arenedes mikroteenustepõhiste süsteemide (MPS) suunas jääb selliste süsteemide kvaliteedi hindamine keerukaks ülesandeks. See doktoritöö tutvustab mõõdikute põhinevat lähenemist MPS-i kvaliteedi hindamiseks. Peamine eesmärk on arendada hästi määratletud mõõdikute kogum, mis tabaks mikroteenuste põhilisi omadusi, nagu modulaarsus ja skaleeritavus, ning oleks kooskõlas tarkvara kvaliteedi tööstusstandarditega. Uuringumeetod hõlmab disainiteaduse lähenemist, viies läbi süstemaatilise kirjandusülevaate, et tuvastada olemasolevad MPS-i kujunduspõhimõtted, kvaliteediomadused ja mõõdikud tuvastatud kujunduspõhimõtete hindamiseks/analüüsimiseks. Uuring jätkub, määratledes ainulaadse mõõdikute komplekti, mis sobib MPS-ile (lisaks olemasolevatele mõõdikutele on kujundatud ka mitu täiendavat mõõdikut), mis kujundatakse hiljem rakendatud lähenemisviisi abil realistlikule juhtumile. Tulemused rõhutavad iga mõõdiku tähtsust süsteemi kvaliteedi põhjalikuks mõistmiseks, sillutades teed tulevastele uuringutele ja praktilisele rakendamisele tarkvaraarenduses. Traditsiooniliste tarkvara hindamise mõõdikute ja mikroteenuste spetsiifiliste nõudmiste vahelise lõhe ületamisega pakub see doktoritöö väärtuslikke teadmisi arendajatele ja arhitektidele, kes soovivad parandada oma mikroteenuste arhitektuuride operatiivset sooritust ja kvaliteeti.

Võtmesõnad:

Mikroteenustel põhinev süsteem, Mõõdik, Kvaliteet, Lähenemine, Süstemaatiline kirjandusülevaade

CERCS:

P170 – arvutiteadus, numbriline analüüs, süsteemid, juhtimine

Table of Contents

List of Abbreviations

MBS – microservices-based system;

RQ – research question;

SLR – systematic literature review;

DS – design science;

WoS – Web of Science;

EC – exclusion criteria;

IC – inclusion criteria;

API – Application Programming Interface;

DTO – Data Transfer Object;

PMS – parking management system;

CRUD – create, read, update, delete;

DSR – Design Science Research.

1 Introduction

The advent of microservices has revolutionized software architecture, introducing a model that emphasizes small, independently deployable services over traditional monolithic structures. This approach, marked by its focus on modularity and decentralized management, offers significant benefits in terms of scalability and agility [1, 2]. However, the transition from traditional system architecture (e.g., monolithic) to a microservices-based architecture (called a Microservices-based System (MBS)) is accompanied by unique challenges, particularly in the realm of system design and evaluation. Consequently, microservices architecture principles, such as modularity, decentralized data management, and continuous delivery need to be assessed, which requires a distinct set of metrics for such effective assessment [2].

The current landscape of evaluating MBS often relies on traditional metrics like scalability and maintainability [1, 3]. However, most existing metrics do not fully address the unique characteristics and challenges of microservices architecture [3]. A notable absence of a standardized, comprehensive framework for evaluating these systems leads to difficulties in objectively measuring their adherence to microservices principles.

This thesis aims to bridge this gap by proposing a novel, metrics-based approach for the evaluation of MBS. The main objective of this thesis is to develop a framework that not only identifies the key attributes of microservices but also quantifies them through a set of well-defined metrics. This approach is intended to provide a more systematic and accurate means for assessing MBS, ensuring alignment with their core principles and maximizing their inherent benefits.

The research begins with an in-depth analysis of traditional metrics and their relevance to microservices. Attributes such as scalability, independence, and maintainability will be examined in the context of microservices, highlighting the need for a more nuanced and comprehensive set of metrics [4]. The ultimate goal is to enhance the evaluation of MBS, contributing to the field of software architecture by establishing a robust, detailed, and systematic methodology for assessment.

1.1 Research problem

The main objective of this research project is to create a detailed approach for evaluating the quality of MBS. To achieve the aforementioned objective, we have formulated the following Research Questions (RQs):

RQ1: What are the key design principles for designing high-quality MBS?

RQ2: Which quality attributes are most critical for the successful implementation of MBS?

RQ3: What metrics exist for evaluating/assessing the identified design principles and quality attributes of MBS?

RQ4: What are the strengths and weaknesses of each of the identified metrics?

The research goals:

- Identify design principles (attributes, benefits) of MBS, quality attributes, and metrics that can be applied to assess the quality of MBS. Consequently, we have chosen to achieve this goal by conducting a systematic literature review (SLR) concerning the related literature.

- Analysing the adequacy and completeness of the existing metrics to evaluate the quality of MBS. Then, developing a complete and adequate set of metrics for evaluating MBS by adopting, adapting existing metrics, and developing new metrics when required.
- Developing an approach that includes, besides the metrics, a systematic process that guides and facilitates the use of the metrics as well as the interpretation of the metrics' results.
- Validating the proposed approach and the adequacy and completeness of the metrics with domain experts and applying these metrics to a realistic scenario to demonstrate their effectiveness and applicability in real-world situations.

1.2 Thesis outline

Section 2 - Background and Related work - the evolution of microservices architecture underscores the importance of tailored quality assessment metrics for effectively evaluating the unique characteristics of MBS.

Section 3 - Research Methodology - outlines a systematic methodology for assessing the quality of MBS using a Design Science approach.

Section 4 - SLR to identify the most relevant studies to answer the RQs - conducts a comprehensive review to identify the most relevant studies to answer the research questions, underpinning the development of the metrics-based evaluation framework.

Section 5 - Developing the metrics-based approach for evaluating the quality of microservices-based systems - describes the creation of the metrics-based approach, integrating both existing and newly developed metrics to assess MBS quality effectively.

Section 6 – Evaluation - applies approach in terms of its existing and the developed metrics to a realistic scenario to demonstrate their effectiveness and the practical applicability of the approach in real-world settings.

Section 7 - Result and Discussion - discusses the outcomes of the metrics application, providing insights into the performance and impact of the approach on the quality assessment of MBS.

Section 8 - Limitations - acknowledges the limitations encountered in the research and suggests future directions to enhance the metrics-based approach.

2 Background and Related work

The era of microservices could be considered to have begun with the introduction of the term "micro-web-services" by Dr Peter Rodgers at a cloud-computing conference in 2005 [5]. The next major milestone can be considered to be the publication of Sam Newman's book "Building Microservices" (2014) and the article "Microservices - a definition of this new architectural term" by James Lewis and Martin Fowler [1, 6].

There have been several phases in the systems development field, and Cloud-native is one of them. This period can be considered the heyday of microservice architecture, as businesses widely adopted it and opened up opportunities such as the Cloud, which is a fairly quick deployment process, scalability, monitoring, etc. [7]. Traditionally, architectural decisions are often made as static, leading to a disability in delivering updated architectural versions and updates timely [8]. Thus, the creation of the architecture of a microservice system should be qualitative and adaptive.

In turn, the definition of microservices can be conveyed as follows: "Microservices are small, autonomous services that work together" (Sam Newman) [1], representing a "Loosely coupled service-oriented architecture with bounded context" (Adrian Cockcroft) [9].

In light of this definition, several general characteristics and principles of microservice architecture can be identified. As emphasised by Lewis and Fowler [6], while not all microservice architectures will contain every characteristic/principle, most of them are expected to exhibit most of the defining qualities.

There is a general trend to consider agreed advantages as the principles of the architectural style [10]. According to Newman, microservices offer numerous benefits, many of which are common to distributed systems. Nonetheless, microservices typically acquire these advantages by pushing the boundaries of distributed systems and service-oriented architecture concepts [1].

Notably, the principles laid out by Lewis, Fowler and Newman [1,6] are frequently used as foundational elements in contemporary research on microservices. From these fundamental sources, the following principles can be identified:

- Newman:
    - Characteristics: Small, Focused on Doing One Thing Well, Autonomous [1].
    - Key benefit: Technology Heterogeneity, Resilience, Scaling, Ease of Deployment, Organizational Alignment, Composability, Optimizing for Replaceability [1].
    - Loose coupling [1, 9], High cohesion [1].
    - Principle: Modeled around business concepts, Culture of automation, Hide internal implementation details, Decentralize all the things, Deploy independently, Isolate failure, High observable [1].
- Fowler & Lewis:
    - Characteristics: Componentization via Services, Organized around Business Capabilities, Products not Projects, Smart endpoints and dumb pipes, Decentralized Governance, Decentralized Data Management, Infrastructure Automation, Design for failure, Evolutionary Design [6].

- Benefits: Strong Module Boundaries, Independent Deployment, Technology Diversity [11].

Upon evaluating the list above, it becomes evident that the principles of microservices, as described by Lewis and Fowler, show considerable alignment with Newman's principles, although with nuanced differences [12]. For instance, while both advocate for componentisation, Newman emphasises hiding implementation details. Such correlations between the two perspectives support a unified understanding of microservices but also highlight areas of specialisation, such as Newman's focus on system observability—a factor not explicitly detailed by Lewis and Fowler.

Building a system based on the principles is possible, but each system must fulfill certain quality criteria. The quality of a system within microservices-based architectures must be aligned with industry standards while considering the unique characteristics of MBS [3, 13].

While ISO 25010 provides a comprehensive framework for software quality, microservices architecture demands an extension of this model to accommodate its distinct characteristics [14]. The nature of microservices — being distributed and loosely coupled — introduces unique quality attributes such as service granularity, modularity, and resilience that are not fully covered by traditional quality standards [3, 13].

In order to effectively evaluate the architectural quality of MBS, it is essential to utilize appropriate metrics that reflect the unique aspects of this architectural style. Among the most well-known metrics is the Number of Lines of Code (LOC) [15], which provides a basic quantitative measure of the size and complexity of services - although in fact it may not carry really important information if considering different programming stack for implementation.

This analysis reinforces the foundational principles, quality attributes of microservices and metrics for assessing them. It underscores the need for a tailored approach to evaluate system quality — prompting further investigation into specific metrics that can encapsulate these attributes.

3 Research Methodology

This chapter describes the proposed methodology for designing our approach that allows for analysing the quality of Microservices-based Systems (MBS). The proposed methodology for assessing the quality of MBS follows the Design Science (DS) approach [16]. Consequently, the process starts with a thorough investigation of the problem, proceeds to the development of a solution, and culminates in the validation of the design [17]. Based on the results of the evaluation, the solution might be improved and re-evaluated. In particular, the methodology adheres to the guidelines proposed by Bell et al. [18]. This approach (depicted in Fig. 1) consists of four key phases, and it is iterative in some parts:

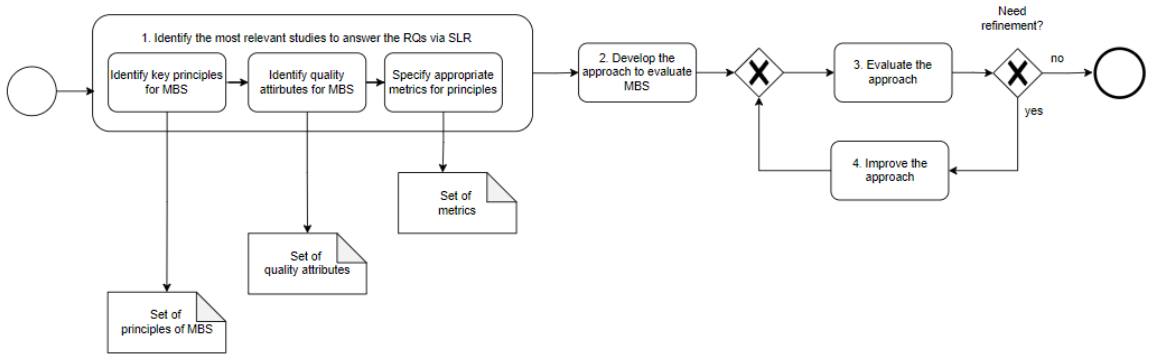

Figure 1. The methodology process

1. **Identification of the problem:** This step aims at identifying the gap in existing methods for evaluating MBS. As discussed earlier, there is a need for a comprehensive metrics-based approach that aligns with the unique characteristics and challenges of MBS.
2. **Approach design:** This step involves developing a novel approach that tackles the aforementioned problem for assessing the quality of MBS. Specifically, the approach will offer a systematic and concrete method to apply the identified metrics for evaluating MBS. The approach includes the creation of a comprehensive guideline, which provides a structured process for practitioners to measure and improve the quality of their microservices systems. This systematic process is tailored to guide software engineers through each stage of evaluation, ensuring a thorough and effective assessment of MBS in various contexts. This approach is practical and can be integrated into the workflow of software development teams, offering a standardized method for assessing the quality of MBS. The development of this approach requires identifying key principles/attributes of MBS, and developing a set of metrics that allows capturing these key principles/attributes while aligning with industry standards and the specific nature of MBS. The identification of key principles, attributes, and qualities of MBS as well as their corresponding metrics was done through an SLR. Detailed information about the SLR is provided in Section 4, and the approach is provided in Section 5.

3. **Approach evaluation:** The proposed approach is evaluated through application to a realistic case study. This evaluation focuses on the approach's effectiveness in accurately assessing the quality of MBS and its alignment with core microservices principles. This step involves a comprehensive evaluation of the developed approach, applying it to actual MBS scenarios to determine its practical effectiveness. This ensures the approach is robust, relevant, and ready for adoption in professional environments. Detailed information about the evaluation of the approach is provided in Section 6.
4. **Improve and re-evaluate the approach:** This phase will be dedicated to improving the approach based on the results of the evaluation received in the previous phase. Then, re-evaluate the approach. After that, the approach will be critically reviewed to identify limitations and areas for improvement. The thesis concludes by suggesting future research directions for refining and enhancing the proposed metrics-based evaluation framework for MBS.

The process involves identifying the problem, designing an approach, evaluating its effectiveness through a case scenario. By adhering to established guidelines and incorporating comprehensive metrics, this methodology provides a structured and practical framework for software engineers to evaluate and enhance the quality of MBS.

4 SLR to identify the most relevant studies to answer the RQs

To identify actual and appropriate information about MBS a Systematic Literature Review (SLR) was conducted. Following [19], the SLR process (shown in Fig. 2) consists of three main phases: 1. Planning the review, in which we formulate the research questions (RQs), and we define the review protocol. 2. Conducting the review through a search process after identifying the search terms and the literature resources, followed by study selection activity, and 3. Reporting the results of the review by extracting detailed information from search results, and then using it to answer the research questions. In what follows, we describe each of these phases.

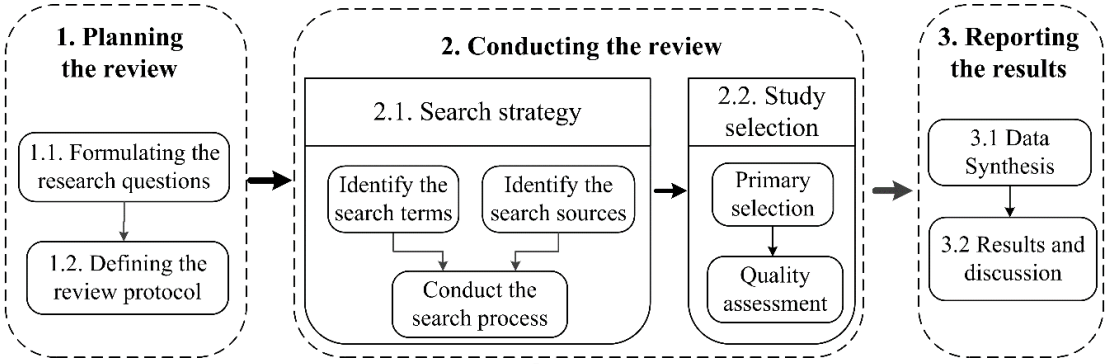

Figure 2. Systematic literature review process

1. Planning the review: includes two key activities:

1.1. Formulating the RQs is a very critical task since these RQs are used to guide the entire systematic review methodology. The main aim of this review is to identify the most mature studies relevant to our study concerning available MBS design principles, their corresponding assessment metrics, etc. In this context, we use research questions for the thesis mentioned earlier in 1.1 Research problem.

1.2. Defining the review protocol. A review protocol specifies the strategy that will be used to search for relevant studies; study inclusion and exclusion criteria, study selection criteria; and data extraction strategies. In what follows, we discuss how we perform each of these activities.

2. Conducting the review. This phase is composed of two main activities: 1- search strategy; and 2- study selection, where each of them is composed of several sub-activities.

2.1. Search strategy aims to find studies related to the research questions using an objective and repeatable search strategy. The search activity consists of three main sub-activities:

2.2.1. Identify the search terms. We derived the main search terms from the research questions. In particular, we used the Boolean AND to link the major terms, and we used the Boolean OR to incorporate alternative synonyms of such terms. The resulting search terms are:

(“Microservice” OR “Microservices” OR “Microservices Architecture” OR “Microservices”) AND (“Principle” OR “Quality” OR “Qualities” OR “Attribute”) AND (“Metric” OR “Measure”)

2.2.2. Identify the literature resources. We have selected several electronic database sources, namely Scopus, IEEE Xplore, Web of Science (WoS), and Springer as they index the main scientific publications in the fields of computer science, software engineering, and information science.

2.2.3 Conduct the search. We have used the search terms to search the two selected electronic database sources, and all returned studies were considered.

2.2. Study selection. Using the search terms to search the electronic database sources, returned 51 for Scopus, 246 papers for IEEE Xplore, 30 for WoS and 222 for Springer resulting in 549 papers. After removing 40 duplicated papers, we read the title, abstract, and skimmed through the rest of the paper for the remaining 509 papers, applying the inclusion and exclusion criteria (shown in Table 1). Specifically, we removed all the papers that were not published in English (EC2), published earlier than 2005 (EC1), and those not in open access (EC4). We also discarded papers whose source type was not a journal or conference proceeding (EC3). On the inclusion side, we focused on papers related to at least one of the RQs (IC1) and which contain sufficient information for at least one of the RQs (IC2). This comprehensive approach ensured that the final selection of papers was highly relevant and contributed significantly to answering our research questions. The remaining 41 papers were fully read, and only papers that contained sufficient information to contribute to the answer to at least one of the RQs were included. This resulted in the selection of 41 papers. A simplified representation of the search and selection process is shown in Fig. 3.

Table 1. Inclusion and exclusion criteria

Exclusion criteria	Inclusion criteria
EC1: Published earlier than 2005 (Peter Rodgers introduced the term "Micro-Web-Services" at the Web Services Edge conference in 2005).	IC1: Studies related at least to one of our RQs
EC2: Language is not English	IC2: Studies that contain sufficient information to be used to answer at least one of our RQs
EC3: Source type is not journal or conference/workshop proceeding (peer-reviewed)	
EC4: Is not in open access	

3. Reporting the results. The final phase of the systematic review involves summarizing the results, and it consists of two main activities:

3.1. Data synthesis aims at combining the findings of the selected studies in a way that allows answering the research questions. To facilitate this activity, each of the 41 selected papers has been analyzed and its contribution to our RQ has been summarized. A simplified representation of how data is synthesized concerning each RQ is shown in Table 2.

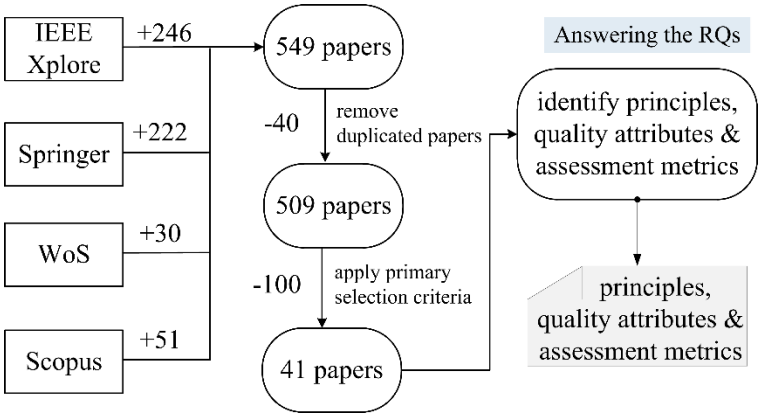

Figure 3. Paper search and selection process

Table 2. Proposed data synthesization

Data Item	Value	RQ
A list of principles extracted from selected studies	Principles/attributes/benefits of MBS	RQ1
A list of quality attributes extracted from selected studies	Quality attributes that are specific for MBS	RQ2
A list of assessment metrics extracted from selected studies	Metrics that are used to assess MBS	RQ3
A discussion concerning the strengths and weaknesses of identified metrics	Strengths and weaknesses of the identified metrics	RQ4

3.2. Review results and discussion. The final results of SLR are placed in the table and are available at the link[1] below. In what follows, we present and discuss the findings of this review concerning each RQ in its corresponding step, as follows:

Identify key principles for MBS: This phase, which relates to **RQ1**, involves establishing the list of core principles that define MBS and would be a baseline for this research. The principles are extracted from the systematic literature review, filtered, and are critical for guiding the subsequent development of quality attributes and metrics. For a comprehensive list of the principles extracted from the systematic literature review, please refer to Appendix 1. This appendix contains a table that encapsulates the core principles of microservices architecture as identified by various researchers.

Discussion RQ1: The principles identified reflect a complex interconnection of technical, organisational, and operational aspects unique to MBS. The focus on small, autonomous services that adopt technological diversity and facilitate ease of deployment represents a significant shift from traditional monolithic architectures. Furthermore, loose coupling, high cohesion, and decentralisation principles are critical for achieving the desired agility and scalability in a microservices landscape. However, while these principles provide a solid foundation for designing high-quality MBS, they also introduce challenges. The significant limitations are the complexity of managing multiple independent services, ensuring system-wide observability, and maintaining consistent performance and reliability across diverse technological stacks. Moreover, aligning organisational structures with microservices architectures requires thoughtful consideration to leverage the benefits fully.

[1]https://docs.google.com/spreadsheets/d/1Wr3j-W3SaLHOd-c62FSj6k1cB_-YmysqcLfERY6_t_Q

Discussion RQ2: The extended model within [14] adapts traditional quality standards to the MBS context, offering a nuanced framework for evaluating system quality that reflects MBS specifications. The alignment of quality attributes with microservices principles reveals a purposeful emphasis on autonomy, scalability, and resilience. Attributes such as scalability, independence, and maintainability highlight the architectural aim for adaptable, flexible systems that can succeed in dynamic environments. Similarly, the focus on deployment ease, modularity, and technology heterogeneity reflects a strategic move towards architectures that support rapid evolution, robust service composition, and technological agility. These quality attributes align with and extend the foundational principles of microservices, addressing the multifaceted challenges associated with implementing and managing MBS. For instance, the complexity inherent in orchestrating numerous autonomous services is countered by attributes emphasizing modularity, manageability, and independence. Challenges in achieving system-wide observability and maintaining consistent performance across varied technological landscapes are addressed through attributes dedicated to observability, performance, and reliability.

Identify quality attributes for MBS: Based on the key principles identified in RQ1, this step determines the quality attributes that are most relevant to MBS, which directly corresponds to RQ2. These attributes serve as the criteria for evaluating the MBS. For a comprehensive list of quality attributes extracted from the literature review, please refer to Appendix 2. This appendix contains a table that encapsulates the core quality attributes for microservices architecture as identified by various researchers.

Discussion RQ3: Metrics serve as quantitative measures for the attributes and principles identified earlier, providing a structured approach to assess the alignment of MBS with its foundational design principles. The range of metrics, from granularity to various metrics for maintainability, autonomy, and interoperability, highlights the multifaceted nature of microservices architecture. This diversity in metrics could ensure that different aspects of microservices, such as their size, complexity, reliability, and scalability, are covered, allowing superficial assessment. It is also worth noting that the highest number of metrics was found for the two principles: cohesity and coupling. Regarding metrics for size estimation, some metrics may not fully evaluate the architectural solution (e.g. Number of Classes per Microservice, Microservice Line of Code, etc.) due to factors such as solution stack, algorithmic complexity of the solution, etc. It is also worth noting that some metrics are purely descriptive and cannot be easily applied. There is no predefined set of metrics that would reflect the real picture in the system, the correlation between not only coupling - cohesion - maintainability, but also in general, to carry a recommendation on all possible improvements.

Having analysed different views on MBS in general, it is worth highlighting that a large number of studies focus on the transition from monolithic architecture. Therefore, most of the tools and metrics are devoted to assessing the quality of reorganisation of components of existing systems. This in turn affects, for example, the quantitative and qualitative set of metrics for MBS designed from scratch. Thus, if you start designing MBS from scratch - there is no single generally accepted solution how to measure the quality of architectural solution design, up to the stage of code writing/application launching, when actually all the defects appear (delay in network data transmission - latency, excessive number of transaction components in SAGA pattern, etc.).

It is also worth highlighting that the proposed metrics have lack of usability: that the use of a metric can point to an implicit problem in the architecture, point to the "bottleneck" of the system, and even more importantly that the set of metrics and their correlation should produce real results - not a metric for the sake of a value that is hard to understand and apply.

Specify appropriate metrics for the MBS principles: With the list of core principles and quality attributes in place, this step, considering **RQ3,** involves specifying the metrics that can quantitatively measure these attributes. This step is crucial for ensuring that the metrics are not only relevant but also effective in assessing the alignment of MBS with its defined principles. Furthermore, understanding the strengths and weaknesses of these metrics, as per **RQ4**, is integral to this step. This process helps in forming a comprehensive framework for evaluating MBS quality, ensuring that the chosen metrics are both appropriate and insightful for the assessment.

Having a list of selected principles and a list of quality attributes, it is necessary to map them and select the appropriate metrics to do so. For mapped principles, quality attributes and appropriate metrics please refer to Appendix 3.

Discussion RQ4: As a standard way, the authors did not express their opinion about the metrics and just described or/and applied them and showed the result: how much the proposed metric can be of value for evaluating the microservice system as a whole and whether to use it in the system design remained unanswered. It was also common to evaluate a migration solution from a monolithic solution to microservices and evaluate this migration using a set of metrics. Therefore, based on the data, it appears that this research problem is not solved and it is worth creating an approach to evaluate MBS as a stand-alone solution (not a transition from monolithic solutions).

In the development of a metrics-based approach for evaluating the quality of MBS a detailed examination of each selected metric's strengths and weaknesses is crucial. This thorough analysis, which appears in Section 5, aims to provide a clear understanding of how each metric contributes to the overall assessment of MBS and supports system design.

5 Developing the Metrics-based Approach for Evaluating the Quality of Microservices-based Systems

Having analysed three aspects at once: principles, quality criteria, and relevant metrics, we can proceed to create an approach for comprehensive quality assessment of the built MBS using metrics (existing and newly designed). The rest of this section is structured as follows: first, we list and define the selected key principles for evaluating MBS that our approach considers, followed by the metrics to be used for assessing the MBS, which have been grouped into five categories. Finally, we present the process underlying our approach.

5.1 Selected principles of MBS for the research

Following the detailed examination of principles and definitions outlined in Table 3, it clearly describes the key attributes that constitute a high-quality MBS. The principles selected for further study are foundational to the architecture itself and integral to the quality and operability of such systems. These principles underpin the design and functional criteria that MBS aims to meet, from goal orientation to resilience and maintainability. This understanding serves as a precursor to developing a metric-based evaluation framework, which will be designed to measure the efficiency and robustness of MBS in subsequent sections of this research.

Table 3. Principles and definitions.

Principle	Definition
Goal-oriented	Emphasizes designing microservices with specific business goals in mind, ensuring each service aligns closely with strategic business objectives. This approach helps maintain focus on delivering value through targeted functionality rather than just technical outcomes [1].
Autonomous	Each microservice operates independently from others with minimal dependencies, which allows for independent development, deployment, scaling, and failure management. This independence supports a robust and resilient system architecture where services communicate through well-defined interfaces [1].
Interoperability	Services are designed to effectively communicate and work together despite being developed independently. This involves using standard protocols and data formats to ensure seamless interaction and integration across different services and platforms [1].
Composability	Refers to the ability to assemble and reassemble services into various configurations to meet changing business needs or to enhance functionality. This modularity allows for greater flexibility and adaptability in the system's architecture [1].
High cohesion	The degree to which the services within a system are self-contained with a well-defined purpose, reducing unnecessary dependencies and enabling more efficient system operations [20]. Ensures that the microservices are internally consistent and focused, where each service is tightly aligned with a specific business capability or function. This minimizes unnecessary complexity and enhances the maintainability of the system [1].
Loose coupling	Services are designed to minimize dependencies on each other, which enhances modularity and reduces the impact of changes in one service on others. This principle supports independent development cycles and reduces the risk of cascading failures across services [1].

Decentralization	Decision-making and data management are distributed across various services, avoiding central points of control or failure. This fosters resilience and flexibility, allowing individual services to evolve independently while contributing to overall system robustness [1].
Resiliency	Services are designed to handle failures gracefully, maintaining overall system availability and reliability even when individual components fail. This includes implementing patterns like circuit breakers, fallbacks, and retry mechanisms to isolate and manage failures effectively [1].
Message-based communication	Microservices interact using asynchronous messaging, which decouples service dependencies in terms of both time and technology. This communication style enhances flexibility and scalability by allowing services to interact without requiring simultaneous availability [6].
Evolutionary design	A software design approach where the design is continually adapted and improved throughout the development process in response to changing requirements and new insights [6].
Maintainability	Services are designed to be easily understandable, tested, and updated, which supports rapid adaptation to new requirements or fixes without significant overhead. Independent serviceability enhances system robustness by enabling focused updates and reducing the scope of impact from changes [21].
Scalability	The architecture supports scaling individual components independently to respond to varying loads, which optimizes resource use and ensures system responsiveness under varying loads [21].
Reliability	The system consistently performs the defined functions under stated conditions, handling expected loads and interactions without failure [21].
Availability	Ensures that the system is operational and capable of fulfilling its intended functions at any given time, despite the failure of individual services or components [21].
Size	Refers to the scope and boundaries of a microservice, emphasizing that services should be small enough to be managed and understood by a small team yet large enough to be functional. The size should enable single-responsibility, allowing a service to encapsulate a single business capability or function effectively. This principle supports the microservice philosophy of building a system that is decomposable into independently deployable, modular components that perform one function well. Size should be measured not just in terms of lines of code, but also by the number of endpoints, data schema complexity, and the cognitive load required to maintain the service [1].

The foundational principles mentioned in Table 3 outline the vector for developing a metric-based evaluation approach to assess MBS. These principles are inalienable to the design and functionality of MBS architectures. This approach will facilitate the quantitative and qualitative assessment of MBS efficiency and robustness, thereby providing valuable insights into the quality and operability of such systems.

5.2 The five assessment metrics categories

The approach to assess the effectiveness of MBS emphasizes a structured examination across multiple crucial dimensions of microservice design and operation. We propose the following dimensions—Domain Design, Data Design, API, Communication, and Essential—as critical for assessing the quality of a MBS.

- **Domain Design:** By initially focusing on the business domain, this dimension ensures that microservices closely align with the specific business requirements and goals of the organization. This alignment guarantees that the system is crafted with a clear understanding of the business context, enhancing its relevance and operational efficiency.
- **Data Design:** The data dimension evaluates how effectively the microservices handle and manage data. Assessing this dimension ensures that the system can effectively process and utilize data to support business operations.
- **API:** The API dimension examines the design and functionality of the application programming interfaces (APIs) used within the microservices architecture. Evaluating this dimension ensures that the microservices are interoperable and can seamlessly communicate with each other and with external systems.
- **Communication:** The communication dimension assesses the mechanisms and protocols used for inter-service communication within the microservices architecture. Evaluating this dimension ensures that the microservices can effectively communicate and collaborate with each other in a distributed environment.
- **Essential:** The essential dimension encompasses fundamental characteristics such as cohesion, coupling, modularity, scalability, and resilience. This dimension serves as a comprehensive evaluation of the overall architecture and design of the MBS, ensuring that it exhibits the desired qualities in terms of flexibility, maintainability, and performance.

The order of these dimensions is significant as it reflects a systematic approach to evaluating the microservices architecture. Starting with the business domain ensures that the system is aligned with organizational goals, while subsequent dimensions discover into various technical aspects of the architecture. By systematically evaluating each dimension, we can gain comprehensive insights into the strengths and weaknesses of the microservices architecture, enabling informed decision-making and continuous improvement.

In the context of microservices, this multi-dimensional evaluation approach is critical due to the distributed nature of the architecture and the complex interactions between individual services. By assessing various dimensions, we can identify potential bottlenecks, performance issues, and areas for optimization, thereby ensuring that the microservices architecture remains robust, scalable, and aligned with business objectives. In what follows, we discuss each of these assessment criteria along with their corresponding metrics.

5.2.1 Domain Design:

We start from the most abstract level of the system to evaluate the system and its components without much detail (technical tools, etc.). Whatever system is developed there is a domain area and initial requirements for the system. We aim to define boundaries within the domain to group similar and related behavior within the microservice and ensure that communication with other microservices are as loose as possible, i.e., the microservices are loosely coupled. Defining the boundaries of the microservice incorrectly can lead to numerous adjustments in how services collaborate, resulting in a costly process [1]. Therefore, the first version of the scheme will often require modification following a full MBS assessment in the next stages of the approach.

So, the following steps can be distinguished:

1. Break down the system into business domains, defining bounded contexts;
2. Encapsulate functionalities within modules (microservices);
3. Minimize dependencies among domains.

While presenting the number of services for the domain, consider such points as *Functionality* (at least microservice has a "well-defined purpose" (scope of actions)) and *Own data* to operate. Third-party API can be also added. As for the data - there is a reference to the Database per service pattern (we can achieve a loosely coupled system, technology heterogeneity is increased, and different databases can be selected for each service). It is worth considering the possibility of system expansion in the future to distribute the functionality in the whole system adequately (potential to scale (scale-data, scale-API, etc.)).

What is the right size of the microservice? The answer is ambiguous - there is no ideal size, but it is worth considering the criteria: as long as the service follows Single Responsibility (for functionality), has its own data to perform the task, is loosely coupled in the system, communicates to other services and can be independently developed, deployed, and managed without having any awareness of the service around it. Therefore, from the principles and quality attributes selected above, we consider at this level: **Goal-oriented** and **Size**.

Regarding metrics, for the Goal-Oriented principle, the Business Context Purity (BCP) metric is re-used (it indicates the mean entropy of business use cases per partition). A microservice is considered functionally cohesive if it implements a lesser number of use cases [22]), but for the Size principle, there is no relation in the identified metrics at the Domain level. Therefore, we also design a new metric to assess the size of the microservice, namely: Microservice Functional Density (MFD).

Metric: Microservice Functional Density (MFD)

Definition: The Microservice Functional Density (MFD) measures the ratio of a microservice's functional capabilities to its operational scope within a specified domain. It aims to quantify the balance between the functionality provided by a microservice and the breadth of the business domain it covers.

Hypothesis: A well-defined MFD will help identify microservices that are either too broad, indicating a need for decomposition, or too narrow, suggesting potential for merging with other microservices. The optimal MFD range should ensure microservices are sufficiently focused on their domain-specific tasks while maintaining the flexibility for future scalability and system expansion.

Observations: To calculate MFD, one would analyse the microservice's defined functionalities against the domain's requirements and operational breadth. This involves reviewing the microservice's responsibilities, data ownership, and interaction with other services. Services with high functionality but limited domain scope would have a high MFD, whereas services with wide domain scope but low functionality would have a low MFD.

Impact on defined architecture: Employing MFD as a metric facilitates a more structured approach to microservice design, promoting services that are tightly aligned with business capabilities. It supports architectural decisions by highlighting when a microservice's size may be impacting its efficiency, maintainability, or scalability. An optimized MFD can lead to an architecture where services are neither too granular nor too monolithic, thereby enhancing the system's overall resilience and adaptability.

Evaluation is relative: MFD values should be interpreted relative to the system's current and future needs. A service considered adequately sized today may need to be split or merged as business requirements evolve. Therefore, continuous reassessment of MFD values is crucial for maintaining an architecture that can adapt to changing demands, ensuring that functionality and data are appropriately distributed across the microservice landscape.

Criteria:

- **Functionality Score** 0 (ill-defined purpose) to 1 (well-defined purpose)
- **Data Score** 0 (doesn't have own data (will use others microservices data)) to 1 (have own data)
- **Connections Score** (Number of connections / Total number of services)
- **Autonomy Score** 0 (highly dependent on other services) to 1 (completely autonomous)

The weights should sum to 1 and be adjusted based on the architectural priorities. For example, if service autonomy is highly valued in the system design, its weight could be higher.

$$MFD = (\text{Functionality Weight} * \text{Functionality Score}) + (\text{Data Weight} * \text{Data Score}) + (\text{Connections Weight} * \text{Connections Score}) + (\text{Autonomy Weight} * \text{Autonomy Score})$$

Recommendations according to the metric value: A higher MFD indicates a microservice densely packed with functionality relevant to its domain, while a lower MFD suggests a broader, potentially under-utilized service scope. So, if the overall MFD is low it's better to rethink the microservices entities.

Table 4. Set of metrics to evaluate MBS at the Domain Desing stage

Principle	Quality attribute	Metrics
Goal-oriented	Functional suitability (ISO 25010)	Business Context Purity (BCP) [22]
Size	Complexity	**Microservice Functional Density (MFD)**

In summary, evaluating MBS at the domain design stage, as outlined in Table 4, involves defining boundaries, encapsulating functionality, and minimizing dependencies. Principles like goal orientation and size, along with metrics such as BCP and proposed MFD, ensure alignment with business objectives and optimal service size. These metrics guide architectural decisions, enhancing system adaptability, while continuous reassessment ensures alignment with evolving business needs.

5.2.2 Data Design:

In today's world data plays a huge role, the system will also collect, process and transfer data between services and to the client (Web, mobile app, other system). Therefore, having the domain diagram and the system requirements, we can start modelling the data schema for each microservice separately. There is also an important privacy aspect to consider when designing the system and the data to be worked with, to leave room for change, so that the data system should be able to be changed (e.g. by migrations). At this stage, you can already see the interrelation of services between each other and notice the data that will be required for interaction/transmission between services. It is important to measure the extent of data dependency between services in a MBS, so we propose the "Shared Data Rate" (SDR) metric.

Metric: Shared Data Rate (SDR)

Definition: Shared Data Rate quantifies the degree of data sharing and dependency between different microservices within a system. It is defined as the ratio of the data elements accessed or required by a service that are owned by another service to the total

number of data elements the service uses. This metric helps in understanding the level of coupling introduced through data dependencies between services.

Hypothesis: A lower SDR suggests that microservices are more autonomous and less dependent on the data owned by other services, promoting loose coupling and enhancing service scalability and resilience. Conversely, a higher SDR indicates a higher degree of coupling through shared data, which could lead to potential bottlenecks, higher complexity in managing data consistency, and increased impact of changes in one service on others.

Observations: To calculate the SDR, each microservice's interactions with data owned by other services should be analyzed. This involves:

- Identifying all data elements (e.g., database tables, columns, or documents) a service reads or writes to.
- Determining which of these elements are owned (needed) by other services.
- Calculating the ratio of external data elements accessed to the total data elements used by the service.

Impact on defined architecture:

Promotes Data Encapsulation: By striving for a lower SDR, the architecture encourages services to encapsulate their data, reducing direct dependencies on the internal data schemas of other services.

Guides Service Design: Services with high SDR may need to be redesigned or refactored to reduce dependencies, possibly through data duplication, event-driven data synchronization, or service decomposition.

Influences Database Selection: Understanding data sharing patterns can guide decisions on choosing the right database and data management strategies, whether it's opting for a shared database with careful governance, separate databases, or a hybrid approach.

Evaluation and Strategy:

The evaluation of SDR should be contextual, considering the specific characteristics of the system and its domain. While a lower SDR is generally desirable, practical considerations, such as the need for real-time data sharing or transactional consistency, may justify certain dependencies. The goal should be to manage and mitigate the risks associated with high SDR values through architectural and design strategies that ensure system robustness and flexibility.

1. Identify Data Elements Used by the Service:

List all the data elements (e.g., database tables, columns, documents) that the microservice reads from or writes to during its operation. This includes both data it owns and data owned by other services.

2. Identify External Data Elements:

From the list created in step 1, identify which data elements are owned by other services. These are the external data elements that the microservice accesses.

$$\text{SDR} = \frac{Number\ of\ External\ Data\ Elements\ Accessed}{Total\ Number\ of\ Data\ Elements\ Used}$$

The result will be a ratio ranging from 0 to 1, where 0 means the service is completely autonomous (uses no external data), and 1 means the service entirely depends on data owned by other services.

Recommendations according to the metric value: Aim for a lower SDR to promote loose coupling and enhance service autonomy. Encapsulate data within microservices to

reduce direct dependencies on internal data schemas of other services. Consider database selection based on data sharing patterns, balancing the need for real-time data sharing with transactional consistency

Table 5. Set of metrics to evaluate MBS at the Data Desing stage

Principle	Quality attribute	Metrics
Autonomous	Data Autonomy	**Shared Data Rate (SDR)**

The Data Design stage of MBS development emphasizes the importance of modeling data schemas for each microservice while considering privacy and the potential for change. The SDR metric offers insights into the degree of data sharing and dependency between services, guiding architectural decisions to promote autonomy and manage coupling effectively (shown in Table 5).

5.2.3 API Design:

The next stage is API Design and here it's possible to apply API First approach [23]. With the foundation from the previous steps, you can look at the API as the product of this stage. It is still necessary to be aware of the initial requirements and to take into account evolutionary design.

First, API definition with expected input and output parameters should be done. There is logic behind every request. The API-first approach encourages modularity, scalability, and reusability of an application's components [23]. Principles of the API-First Approach: Design APIs for the users, Design APIs that meet business needs, Adopt a modular approach to design the APIs etc [23]. As for metrics for Size, a subset that refers to endpoints is taken: Number of Produced Endpoints, Number of Consumed Endpoints and Average Number of Endpoints per Service. Incorporating these metrics into the API Design stage allows for a systematic evaluation of the APIs in terms of complexity.

Metric: Number of Produced Endpoints (NPE)

Usage: This metric counts the total number of unique endpoints that a microservice exposes.

Importance: A higher number of produced endpoints may indicate a richer feature set but can also imply increased complexity and potential for security risks. It's important to ensure that the microservice provides necessary functionalities without overexposure, which aligns with the principle of minimal API surface for security.

Metric: Number of Consumed Endpoints (NCE)

Usage: Measures the number of unique endpoints that a microservice consumes from other services.

Importance: This metric helps in understanding the dependencies and interaction levels of a microservice with other parts of the system. Lower numbers suggest better isolation and independence, which are key for microservice scalability and reliability.

Metric: Average Number of Endpoints per Service

Usage: Calculated by averaging the number of endpoints across services.

Importance: Provides a quick overview of how interlinked and potentially complex the microservices are. An optimal number suggests a balance, avoiding both overly simplistic services that do little and overly complex ones that are hard to maintain and scale.

For evaluating resilience at the API level, especially with considerations like the use of circuit breakers etc., we propose a relevant metric to assess the robustness and fault tolerance of the API. Here's how such a metric could be structured:

Metric: API Resilience Index (ARI)

Definition: The API Resilience Index measures the robustness of an API by assessing its implementation of resilience patterns such as circuit breakers, rate limiting, and retry mechanisms. ARI quantifies the degree to which an API is designed to handle and recover from errors, excessive loads, and failures.

Hypothesis: A higher ARI indicates an API that is well-equipped to maintain functionality and performance under adverse conditions, such as high traffic or service failures. This resilience contributes to overall system stability, improving user experience and system reliability.

Observations: To calculate the ARI, analyse the API for:

- The presence and effectiveness of circuit breakers to prevent cascading failures.
- Rate limiting capabilities to manage load and protect against abuse.
- Retry mechanisms and their intelligence (e.g., exponential backoff).
- Implementation of fallback methods to provide graceful degradation of service.
- Timeouts to prevent system hang-ups from waiting indefinitely for responses.

Impact on Defined Architecture:

- **Promotes System Stability:** By encouraging the design of fault-tolerant APIs, ARI leads to a more stable and reliable system.
- **Enhances User Experience:** A resilient API ensures that users experience minimal disruptions, even when underlying services are unstable or overburdened.
- **Facilitates Scalability:** APIs designed for resilience can better handle scaling challenges by managing load efficiently and maintaining functionality under stress.

Formula: While ARI is conceptually qualitative, it can be quantified through a checklist approach or weighted factors, for example:

$$\text{ARI} = \frac{\text{Weighted Sum of Resilience Features}}{Total\ Possible\ Resilience\ Features}$$

Where:

Weighted Sum of Resilience Features is calculated by assigning points to each resilience feature (e.g., circuit breaker = 2 points, rate limiting = 2 points, retries = 1 point, etc.), and summing them based on the API's implementation.

Total Possible Resilience Features represent the maximum achievable points if all resilience patterns were implemented optimally.

A perfect score would indicate that the API fully implements all considered resilience patterns effectively. Adjustments to the weights and features can be made based on specific architectural priorities and resilience strategies.

Recommendations according to the metric value: A high ARI score indicates that your API extensively implements resilience patterns effectively, suggesting robust fault tolerance and high reliability under various conditions.

A moderate score suggests that while some resilience patterns are implemented, there may be room for additional enhancements. Recommendations for moderate value: identify gaps and analyze which resilience patterns could be enhanced or are missing. Focus on critical areas like rate limiting, retries, and circuit breakers.

A low score indicates insufficient resilience practices, making the API vulnerable to failures and high loads, which could lead to system instability. Recommendation for low value: Conduct a thorough review of current resilience implementations. Identify any critical endpoints lacking sufficient fault tolerance measures.

Table 6. Set of metrics to evaluate MBS at the API Desing stage

Principle	Quality attribute	Metrics
Size	Complexity	Number of Produced Endpoints [NPE] [24], [25] Number of Consumed Endpoints [NCE] [24], [25] Average Number of Endpoints per Service [26]
Resilience	Reliability (ISO 25010)	**API Resilience Index (ARI)**

Incorporating these metrics into the API Design stage allows for a systematic evaluation of the APIs in terms of complexity and resilience (shown in Table 6). It helps in identifying potential design flaws that could compromise the system's resilience.

5.2.4 Communication Design:

Microservices can communicate with each other using different technologies and approaches and this communication is inevitable, so the next plane in which to consider the system is communication. A service that is loosely coupled minimizes its knowledge and dependency on the services it interacts with. This approach also suggests limiting the variety and frequency of calls between services. Excessive communication not only poses a risk to performance but also increases the degree of interdependence, leading to a tightly coupled architecture [1].

For Autonomous principles, there is a subset of relevant metrics form the metrics identified earlier but for Message-based communication were chosen only two relevant metrics that can be calculated during the design but not during operation and other identified metrics do not help to assess the quality of the MBS at this stage.

Metric: Inter-Partition Call Percentage (ICP)

Metric Usage: Assesses the level of service independence by quantifying how many calls are made to services outside a given service's immediate environment or partition.

Importance: A lower ICP value indicates better service autonomy and less interdependence, aligning with microservices principles of loose coupling and high cohesion.

Metric: Service Messaging Persistence utilization metric (SMP)

Metric Usage: Measures the adoption of persistent messaging systems which are crucial for data integrity and reliability in asynchronous communications.

Importance: Enhances fault tolerance and ensures reliable message delivery, critical in distributed systems where services operate on different nodes or environments.

For the Communication Design stage in evaluating MBS, focusing on how microservices share and expose data through their communication interfaces is essential because each

operation increase chance of failure, increase latency, etc. An important aspect to consider is the balance between the granularity of Data Transfer Objects (DTOs) and the necessity of data exposure. So, we propose a metric to measure this balance:

Metric: Data Exposure Efficiency (DEE)

Definition: Data Exposure Efficiency measures the balance between the granularity of Data Transfer Objects (DTOs) and the minimization of unnecessary data exposure. It evaluates how effectively a microservice's communication design limits the data shared to only what is necessary, aiming to reduce over-fetching or under-fetching of data. It quantifies the presence of non-mandatory fields in DTOs used across the microservice architecture.

Hypothesis: An optimal DEE suggests that the system effectively balances the number and structure of DTOs against the need to minimize data exposure, enhance data privacy, reduce bandwidth usage, and improve response times. A poorly designed communication schema, either by exposing too much unnecessary data or by creating too many granular DTOs leading to over-complexity, can negatively impact system performance and maintainability.

Observations:

- Assess the DTOs used in the system for their structure and the fields they expose in various API endpoints or message payloads.
- Evaluate scenarios where excessive data is returned that may not be used or needed by the consuming service or client.
- Consider cases where too many DTOs may be designed to limit data exposure but lead to increased complexity in data management and integration.

Impact on Defined Architecture:

- Promotes Efficient Data Sharing: Encourages the design of communication interfaces that share data efficiently, optimizing network traffic and response times.
- Enhances Data Privacy: Reduces the risk of data leakage by ensuring that only necessary data is exposed through microservices interactions.
- Improves System Scalability: By reducing unnecessary data transfer, the system can handle more requests with lower resource consumption, enhancing scalability.
- Simplifies Integration and Maintenance: Balancing the granularity of DTOs simplifies the integration process for clients and services and makes the system easier to maintain.

$$\text{DEE} = 1 - \frac{\text{Number of Unnecessary Data Fields in all DTOs}}{\text{Total Number of DTOs fields}}$$

Total Number of DTOs includes all DTOs designed for service communication.

A DEE value closer to 1 indicates a higher efficiency in data exposure, suggesting that most DTOs are optimally designed to share only necessary data.

Table 7. Set of metrics to evaluate MBS at the Communication Design stage

Principle	Quality attribute	Metrics
Autonomous	Independence	Inter-Partition Call percentage (ICP) [22]
Message-based communication	Interoperability	Service Messaging Persistence utilization metric (SMP) [27] **Data Exposure Efficiency (DEE)**

These metrics (mentioned in Table 7) collectively provide a part of the approach for evaluating the communication aspects of a microservices architecture, focusing on independence, reliability, and the efficient handling of data and states across distributed environments.

5.2.5 Essential characteristics:

Having analysed the system on Domain Design, Data Design, API and Communication dimensions, it is possible to go to the holistic abstract level of the system to evaluate the basic principles: cohesion, coupling, granularity, etc. It is also worth noting that most typical metrics are considered to be tied to a codebase and thus cannot be counted at the design stage (Table 8). From the Essential characteristics, we will reuse several metrics identified in the SLR, such as:

Metric: Coupling Between Microservice (CBM)

Metric Usage: CBM measures the number of different microservices that a particular microservice interacts with. It provides a count of unique outbound interactions to other microservices, excluding the frequency of these interactions. This metric is critical for understanding the degree of dependency a microservice has on others.

Importance: Lower CBM values are typically desired as they indicate a microservice architecture with less interdependence between services, which simplifies management and scaling while enhancing fault tolerance. High CBM values might suggest a tightly coupled system that could be more difficult to maintain and scale.

Metric: Instability Index (I)

Metric Usage: The Instability Index quantifies a microservice's susceptibility to change, defined by the ratio of outgoing dependencies (efferent coupling) to the total number of dependencies (sum of incoming and outgoing, i.e., afferent and efferent coupling). This metric helps in understanding how changes in one microservice might affect others and vice versa.

Importance: The Instability Index is essential for assessing the balance between a microservice being too rigid (not easily changeable) and too unstable (too susceptible to changes). A lower index (closer to 0) indicates higher stability, suggesting that the service is less likely to be affected by changes in the rest of the system. This helps in strategic planning for service updates and mitigating the impact of changes across the microservice landscape.

Table 8. Set of metrics to evaluate Essential MBS characteristics

Principle	Quality attribute	Metrics
High cohesion	Cohesion	Partially covered at the Domain and API stages as generally measuring the alignment of services with business capabilities.
Loose coupling	Coupling	Coupling Between Microservice (CBM) [15]
Maintainability	Maintainability (ISO 25010)	Evaluation of the degree of system's complexity could be done from the set of metrics.
Goal-oriented	Granularity	Partially covered at the Domain and API stages as

		assessing whether a service is too granular or too coarse by evaluating the number of endpoints and the scope of their functionality relative to the business domain.
Composability	Modularity	It is difficult to quantify before an operational codebase exists because it depends on Runtime Behavior and Interactions
Evolutionary design	Evolvability	Instability Index (I) [22]
Interoperability	Compatibility (ISO 25010)	Covered with Service Messaging Persistence utilization metric (SMP) in Communication section
Availability	Availability	Not to be calculated as it needs working system.
Decentralization	Manageability	Partially covered in Microservice Functional Density (MFD)
Scalability	Scalability	Partially covered in the Service Granularity Metric "SGM" - Assesses whether services are designed to handle varying loads by scaling out (adding more resources) or scaling in (removing resources) efficiently, without impacting other services.
Reliability	Reliability (ISO 25010)	Partially covered in API Resilience Index

Table 8 outlines a set of metrics to evaluate essential characteristics of MBS across different quality attributes. These metrics provide critical insights into high cohesion, loose coupling, and other key principles, helping guide the design and evolution of robust and scalable microservices.

5.3 The Methodological Process

The process underlying our approach is shown in Fig. 4, and it is composed of five phases:

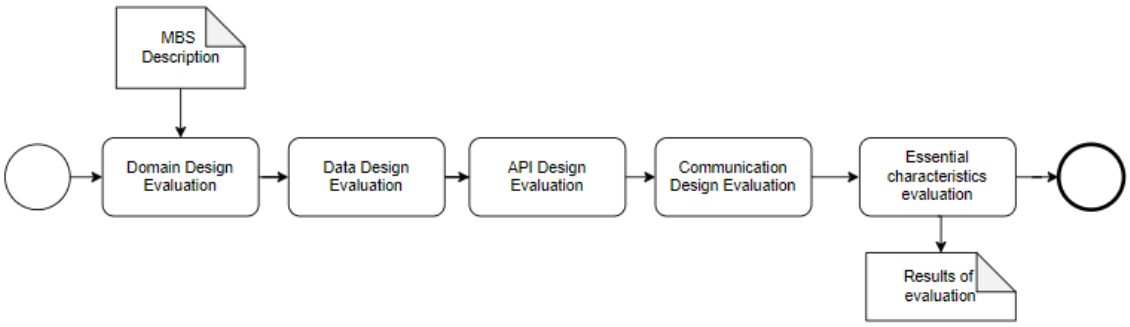

Figure 4. Methodological process

The process begins by applying the domain design metrics to the system of concern (basically to the scenario of the system and the idea of functionality for the services) to establish a foundational understanding of how well the microservices align with business domains. This is crucial for ensuring that each microservice is designed around well-defined business capabilities.

Following the initial domain evaluation, data design metrics are applied to analyse how data is managed across the system. Having a data schema makes it possible to understand the level of data autonomy.

Next, API design metrics are assessed to measure the effectiveness of the microservices' interfaces, evaluating how these APIs facilitate communication between services and with external clients, which is essential for operational efficiency and scalability. Metrics can be applied to the list of necessary APIs for service.

Communication design metrics are then reviewed to determine the effectiveness of the inter-service interactions within the architecture, focusing on the reliability and efficiency of these communications.

Lastly, the essential characteristics of the system are measured, including scalability, reliability, etc. This final step ensures that the system is designed to meet current demands and is robust enough to adapt to future needs.

Each step of this methodological process involves a critical evaluation of different aspects of the microservices architecture, ensuring a comprehensive assessment that aligns with evolving business needs and technological advancements.

This section has established a comprehensive metrics-based framework for evaluating the architectural quality of MBS. By systematically integrating key microservice principles with specific, quantifiable metrics, we have laid a foundation for objectively assessing the design and operational effectiveness of these systems from the outset. The metrics chosen outline a wide array of architectural aspects—from cohesion and coupling to granularity and scalability—ensuring a multifaceted evaluation of system designs.

The approach described not only underscores the importance of aligning architectural decisions with business objectives and technical requirements but also provides a structured pathway for architects and developers to quantify and optimize the design of microservices. This approach is especially crucial in the early stages of system development, allowing for adjustments before implementation, thereby enhancing the overall efficiency, maintainability, and robustness of microservices-based systems.

Thus, knowing the whole system designed before implementation, it is possible to anticipate what situations may occur and try to stabilise the system before launch, thus saving time and effort, taking into account the essence of microservices - distributed systems that are loosely coupled and can be scaled.

6 Evaluation

In Design Science Research (DSR), evaluation aims to demonstrate the utility of a design artifact [28, 29]. The evaluation provides evidence that the outcome of DSR achieves the purpose for which it was designed [29], i.e., artifacts are evaluated based on how well they support solutions in solving problems (e.g., utility, usability, and validity) [28]. Consequently, the proposed approach is evaluated by applying it to a realistic MBS.

In what follows, we provide a basic description of a parking management system, the use case and a set of microservices to implement and further develop and apply the proposed metrics to evaluate the architecture of the proposed MBS.

Proposed realistic case description for the parking management project: A client is looking for a parking spot in a busy area. They access the parking management system (PMS) application, which uses the **Location Service** (3rd party API, for example, Google API) provided to find nearby parking. The client selects a parking and a slot from the available options. At this point, the **Booking Service** is used to reserve the slot. Clients must be registered in the system to book the slot**. User Management Service** handles user data, credentials and permitted scope of usage. To complete the booking, the client needs to make an advance payment, which is processed by the **Payment Service**. A client arrives at the parking, they park their vehicle, and the **Parking Management Service** is updated to reflect the newly occupied slot. Over time, the **Analytics Service** collects data on the PMS metrics like price, usage, etc. The Parking Management Service allows landlords to manage parking slots by setting availability, rates, properties, etc. An administrator would manage the overall system. The administrator would have access to all internal services to handle system malfunctions and other issues and provide a positive user experience for landlords and clients.

The diagram of the proposed system is shown in Fig. 5. The source code is available in Appendix 4.

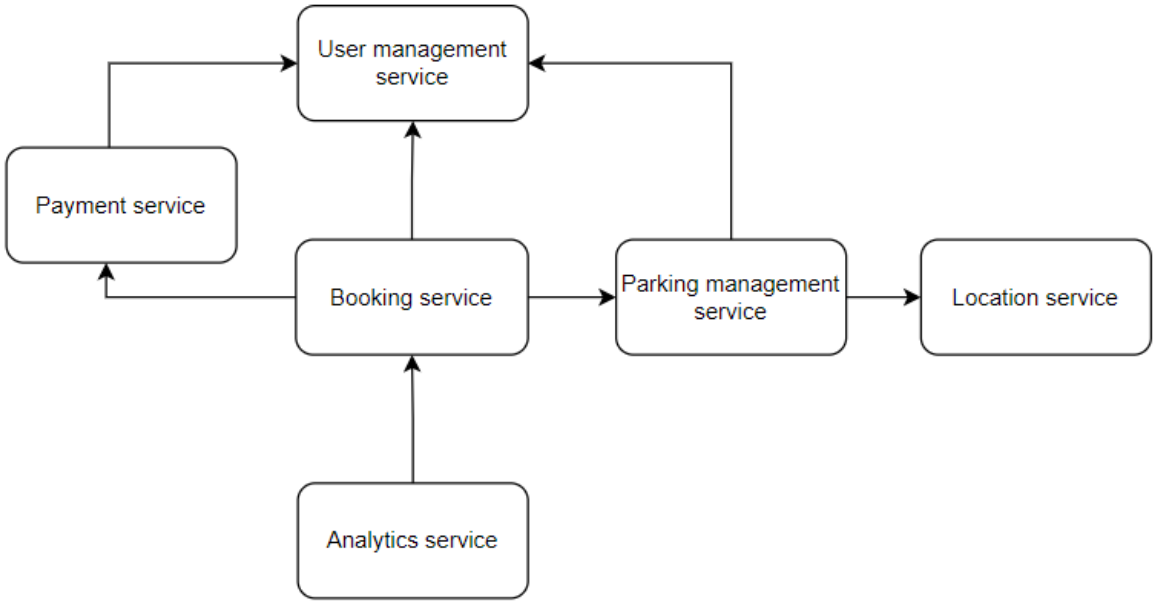

Figure 5. Microservices architecture for PMS

The approach described in Section 5 will be applied, consistently to assess the quality of the MBS mentioned above. As it was proposed the approach starts with Domain Design and there we have 2 metrics BCP and MFD.

6.1 Domain Design Metrics Calculation

1. Business Context Purity:

1. Identify Business Use Cases for Each Service:
    a. Location Service: find nearby parking (1 operation).
    b. Booking Service: reserve a parking slot (1 operation).
    c. User Management Service: manage user registration and other CRUD operations; authentication (2 operations).
    d. Payment Service: process payments (1 operation).
    e. Parking Management Service: manage parking slots (create, delete), update slot status (2 operations).
    f. Analytics Service: collect usage and price data (1 operation).

Total Number of Operations: 8

2. Calculate Proportion of Use Cases (p) for Each Service:

Proportions:

Location Service: $p_1 = \frac{1}{8}$

Booking Service: $p_2 = \frac{1}{8}$

User Management Service: $p_3 = \frac{2}{8} = \frac{1}{4}$

Payment Service: $p_4 = \frac{1}{8}$

Parking Management Service: $p_5 = \frac{2}{8} = \frac{1}{4}$

Analytics Service: $p_6 = \frac{1}{8}$

$$BCP = -\sum (p_i \log_2 p_i) = 3.25$$

2. Microservice Functional Density:

This metric applies to each service and then we have the average for the system. Table 9 presents the weight parameters for the MFD metric.

Table 9. Weight Parameters for MFD

Parameter	Value
Functionality Weight	0.3
Data Weight	0.2
Connections Weight	0.4
Autonomy Weight	0.1

MFD for each service can be calculated:

$$MFD_{service} = \sum (\text{Functionality Weight} * \text{Functionality Score}) + (\text{Data Weight} * \text{Data Score}) + (\text{Connections Weight} * \text{Connections Score}) + (\text{Autonomy Weight} * \text{Autonomy Score})$$

Table 10 details the score parameters for MFD across different services, providing specific values for functionality, data, connections, and autonomy for each service.

Table 10. Score Parameters for MFD for services

Service	Parameter	Value
Location Service	Functionality Score	1
	Data Score	1
	Connections Score	0.16
	Autonomy Score	0.1
Booking Service	Functionality Score	1
	Data Score	1
	Connections Score	0.66
	Autonomy Score	0.8
User Management	Functionality Score	1
	Data Score	1
	Connections Score	0.5
	Autonomy Score	0.3
Payment Service	Functionality Score	1
	Data Score	1
	Connections Score	0.33
	Autonomy Score	0.7
Parking Management Service	Functionality Score	1
	Data Score	1
	Connections Score	0.5
	Autonomy Score	0.8
Parking Management Service	Functionality Score	1
	Data Score	1
	Connections Score	0.5
	Autonomy Score	0.8
Analytics Service	Functionality Score	1
	Data Score	1
	Connections Score	0.16
	Autonomy Score	0.9

Location Service: MFD = (0.3*1) + (0.2*1) + (0.4* $\frac{1}{6}$) + (0.1*0.1) = 0.57

Booking Service: MFD = (0.3*1) + (0.2*1) + (0.4* $\frac{2}{3}$) + (0.1*0.8) = 0.85

User Management Service: MFD = (0.3*1) + (0.2*1) + (0.4* $\frac{1}{2}$) + (0.1*0.3) = 0.73

Payment Service: MFD = (0.3*1) + (0.2*1) + (0.4* $\frac{1}{3}$) + (0.1*0.7) = 0.7

Parking Management Service: MFD = (0.3*1) + (0.2*1) + (0.4* $\frac{1}{2}$) + (0.1*0.8) = 0.78

Analytics Service: MFD = (0.3*1) + (0.2*1) + (0.4* $\frac{1}{6}$) + (0.1*0.9) = 0.65

Average MFD for the system can be calculated:

$$MFD_{average} = \sum((\text{Functionality Weight} * \text{Functionality Score}) + (\text{Data Weight} * \text{Data Score}) + (\text{Connections Weight} * \text{Connections Score}) + (\text{Autonomy Weight} * \text{Autonomy Score})) / Total\ number\ of\ services$$

$$MFD_{average} = \frac{0.57 + 0.85 + 0.73 + 0.7 + 0.78 + 0.65}{6} = 0.71$$

Table 11 displays the results for Domain Design metrics.

Table 11. Domain Design metrics results

Metric	Result
Business Context Purity (BCP) [22]	3.25
Microservice Functional Density (MFD)	0.71

Results discussion:

The BCP result indicates a moderate level of business context purity. This value suggests that while each microservice in the parking management system tends to focus on a distinct aspect of the system's functionality, there is room for improvement in terms of aligning each service more closely with singular, distinct business functionalities.

The MFD value reflects a relatively balanced distribution of functionalities across the microservices, suggesting that each service is neither too coarse nor too fine-grained. However, some services, like the User Management Service and Parking Management Service, which scored higher MFD values, could be handling more complex or numerous functionalities. A high average MFD value implies that the services are functionally dense, potentially leading to issues related to service manageability and scalability in the future. It may be beneficial to examine whether the services with higher MFD scores are indeed optimally designed.

Cohesion and Coupling: The results suggest adequate cohesion within services but hint at potential for over-coupling, especially in services with higher MFD values. This could be addressed by revisiting service boundaries and ensuring that each microservice adheres more strictly to the Single Responsibility Principle.

Scalability and Maintainability: While the current service design suggests a good foundational structure, the relatively high MFD values for some services may impact the system's scalability and maintainability. Reducing the functional density by refining service responsibilities could enhance these attributes.

Business Alignment: The moderate BCP value across the system suggests that while services are generally aligned with business functions, there may be benefits to further refin-

ing the allocation of responsibilities to better match business capabilities with technical implementations.

6.2 Data Design Metrics Calculation

To visualise data design the overall databases schema is prepared (Fig. 6). Each service has the separate database that consists of the tables: Location service (locations table); Analytics service (analytics table); Booking service (bookings table); Payment service (payments table); Parking service (parking_slots table) and User Management service (users, payment_methods, tokens tables).

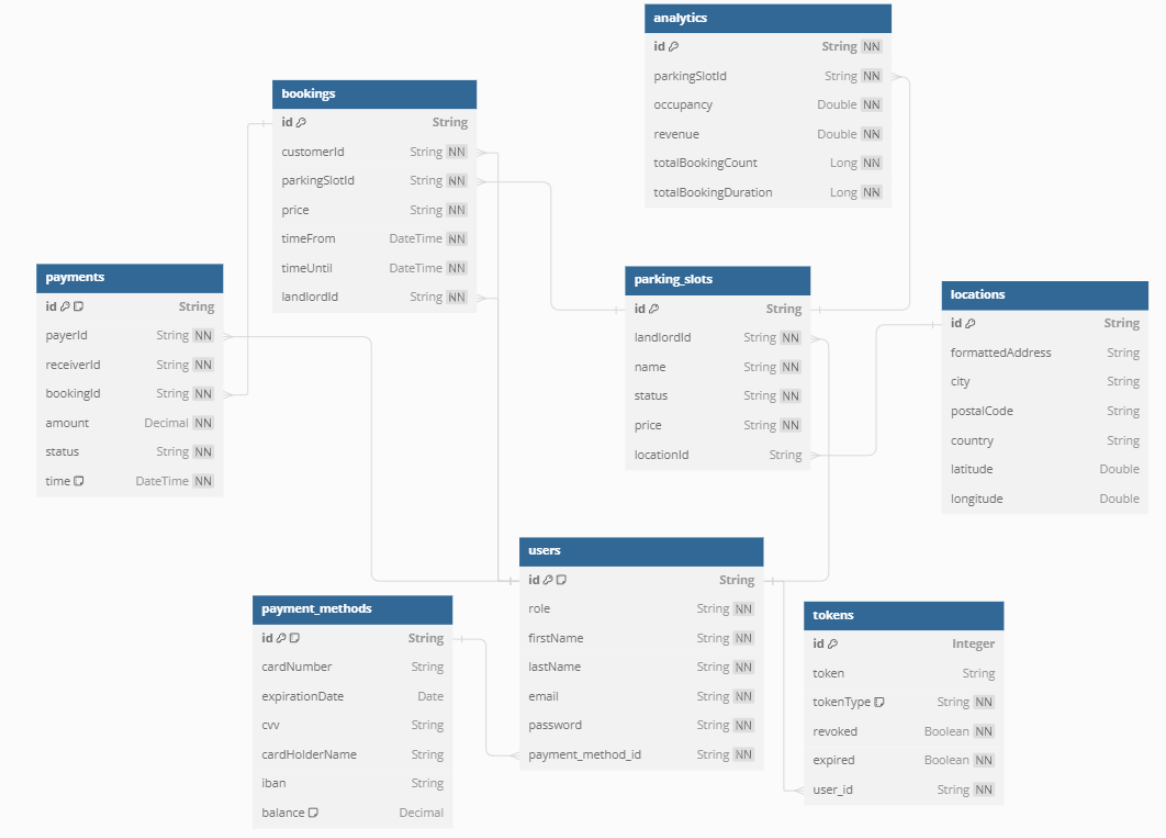

Figure 6. Databases schema for PMS

1. Shared Data Rate (SDR)

1. Identify Data Elements Used by Each Service:
2. Identify External Data Elements:

Table 12. Data Elements for SDR

Service	Data Elements Used by Each Service	External Data Elements
Location Service	locations table	None
Analytics Service	analytics table, parking_slots	None
Booking Service	bookings, parking_slots, users tables	parking_slots (Parking Management), users (User Management)
Payment Service	payments, users, bookings tables	users (User Management), bookings (Booking Service)

Parking Management Service	parking_slots, locations tables	locations (Location Service)
User Management Service	users, payment_methods, tokens tables	None

Table 12 highlights the dependencies each service has regarding data owned by other services.

SDR for the system can be calculated:

$$\text{SDR} = \frac{Number\ of\ External\ Data\ Elements\ Accessed}{Total\ Number\ of\ Data\ Elements\ Used}$$

Calculated values for SDR for each service are shown in Table 13:

Table 13. Calculated SDR

Service	SDR value	Dependency
Location Service	0	no external dependencies
Analytics Service	0.5	no external dependencies
Booking Service	0.67	depends on data from Parking Management and User Management
Payment Service	0.67	depends on data from User Management and Booking Service
Parking Management Service	0.5	depends on data from Location Service
User Management Service	0	no external dependencies

Results discussion:

Location and User Management services is highly autonomous with an SDR of 0, indicating no dependency on external data, which is ideal in microservices architecture.

Booking and Payment Services have a higher SDR (0.67), indicating significant external data dependencies. This suggests these services are less autonomous and might benefit from optimization to reduce external dependencies.

Parking Management Service and Analytics have a moderate SDR of 0.5, reflecting some level of external data dependency but still maintaining a fair degree of autonomy.

Recommendations: Optimize Booking and Payment Services: Consider refactoring these services to reduce dependencies on external data. This could involve duplicating some data locally or using event-driven approaches to synchronize data updates, enhancing service independence and reducing coupling.

6.3 API Design Metrics Calculation

To calculate appropriate metrics for API we have a list of endpoints of PMS and necessary connections (requests) between services (for example using WebClient in the project).

1. NPE is a count of the number of endpoints that each service provides [25];

2. NCE is a count of the number of endpoints a service consumes from other services to fulfill its functionality [25].

Table 14 provides calculated metrics for API Design, listing NPE and NCE for services and includes comments on specific connections between services.

Table 14. Calculated NPE and NCE

Service	NPE	NCE	Comment
Location Service	1	0	
Analytics Service	1	0	Kafka is used
Booking Service	5	2	Request to Parking management service and Payment service
Payment Service	1	1	Request to User management service
Parking Management Service	9	0	
User Management Service	10	0	

3. Average Number of Endpoints per Service

$$\text{Average Number of Endpoints per Service} = \frac{\sum \text{NPE}_i}{\text{Number of services}}$$

$$\text{Average Number of Endpoints per Service} = \frac{27}{6} = 4.5$$

4. API Resilience Index (ARI)

Circuit breaker = 2 points, rate limiting = 2 points, retries = 1 point.

$$\text{ARI} = \frac{\text{Weighted Sum of Resilience Features}}{\textit{Total Possible Resilience Features}}$$

In the project Circuit Breaker only implemented in Booking service while making a request to Payment service, but there are several places that need to be covered with resilience features (for example, rate limiting for 3d party API in Location service, connections between services Booking and Parking management - retry, Payment service and User management - retry).

Weighted Sum of Resilience Features = 2

Total Possible Resilience Features = (2x retry + rate limiting + circuit breaker) = 6

$$\text{ARI} = \frac{2}{6} = 0.3$$

Results discussion:

The range of NPE values varies significantly across the services, from as low as 1 for Location and Analytics Services (this suggests that they have a very focused set of functionalities, potentially indicating good adherence to the microservices principle of bounded context) to as high as 10 for User Management Service. This variability indicates differing levels of responsibilities assigned to each service.

User Management Service with the highest NPE (10) might be managing a wide range of functionalities related to users, which might indicate either a comprehensive service or potential for overloading with too many responsibilities.

Parking Management Service, with an NPE of 9, suggests a significant role in the system, likely handling various aspects of parking management which could be expected given its central role in the PMS.

Most services have a low NCE, with half of the services (Location, Analytics, Parking Management, and User Management) having an NCE of 0, indicating no external dependencies in terms of consuming other services' endpoints. This could imply a high level of

independence and potentially good decoupling, enhancing their reliability and making them less susceptible to failures in other parts of the system.

Booking Service shows the highest NCE (2), relying on both Parking Management and Payment Services. This interdependency could be a critical point to monitor for potential bottlenecks or failures that could affect the booking functionality (ARI can help to analyse the situation). Payment Service has an NCE of 1, depending on the User Management Service, which is logical given that payment functionalities often require user data.

The calculated average of 4.5 endpoints per service suggests a moderate level of functionality per service on average. While this number helps in understanding the general size of services, the actual distribution indicates that some services might be doing significantly more work than others.

Potential Overloaded Services: User Management and Parking Management Services, with high NPE, could be at risk of becoming too complex or taking on too many responsibilities. It might be beneficial to audit these services to see if they can be decomposed further without compromising system integrity.

Dependency Management: Booking Service, as a central orchestrator relying on multiple other services, should be designed with robust error handling and fallback mechanisms to manage the dependencies effectively.

Service Autonomy: The services with zero NCE are ideally positioned in terms of service autonomy. Ensuring that this autonomy is maintained during future developments will be crucial for sustaining system resilience and scalability.

Balanced Functionality: Consider reviewing the functionalities assigned to each service, especially those with high NPE. Redistributing responsibilities might help in achieving better load balancing and more efficient microservices.

A low ARI score indicates missed resilience practices, so it is possible to add missed resilient features to make the system more reliable.

6.4 Communication Metrics Calculation

At the project communication using Kafka is used only between Booking service and Analytics service – BookingDTO.

1. Inter-Partition Call percentage (ICP) amounts to the percentage of calls between two microservices i and j [22].

Assume total inter-service calls within the system include calls from the Booking Service to the Parking Management and Payment Services, plus the Kafka message to the Analytics Service.

Total calls from Booking Service = 2 (to Parking Management and Payment) + 1 (to Analytics via Kafka) = 3

Total inter-service calls in the system = 3 (as no other services communicate externally).

Thus, ICP for the Booking Service to Analytics Service:

$$\text{ICP} = \frac{\text{Number of Kafka messages from Booking to Analytics}}{Total\ inter-service\ calls} * 100$$

$$\text{ICP} = \frac{1}{3} * 100 = 33.3\%$$

2. Service Messaging Persistence utilization metric (SMP)

Since Kafka inherently supports message persistence, and it is used for communication between these two services, we can assume full utilization for this specific interaction.

If considering the system where only this Kafka communication is the persistent message exchange:

$$SMP = \frac{Number\ of\ persistent\ message\ interactions}{Total\ messaging\ interactions}$$

$$\text{SMP} = \frac{1}{3} = 0.33$$

3. Data Exposure Efficiency (DEE)

BookingDTO has 2 unnecessary fields and in total 7 fields.

$$\text{DEE} = 1 - \frac{\text{Number of Unnecessary Data Fields in all DTOs}}{\text{Total Number of DTOs fields}}$$

$$\text{DEE} = 1 - \frac{2}{7} = 0.72$$

Table 15 outlines the results for Communication metrics.

Table 15. Communication metrics results

Metric	Result
Inter-Partition Call percentage (ICP) [22]	33.3%
Service Messaging Persistence utilization metric (SMP) [27]	0.33
Data Exposure Efficiency (DEE)	0.72

Results discussion:

The ICP value of 33.3% indicates that one-third of all service calls within the system are dedicated to communication between the Booking Service and the Analytics Service via Kafka. This relatively high percentage underscores the significance of the data flow between these two services, particularly in terms of analytics and booking data processing.

The SMP score of 0.33 reflects that only one out of three total messaging interactions utilizes Kafka's message persistence capabilities. This score indicates moderate use of message persistence features provided by Kafka, which are crucial for ensuring reliable message delivery in distributed systems.

A DEE of 0.72 indicates a fairly efficient design in terms of data exposure, with the BookingDTO containing only two unnecessary fields out of seven total. This suggests a conscious effort to minimize unnecessary data exposure, which is crucial for protecting sensitive information and optimizing communication efficiency.

6.5 Essential Metrics Calculation

1. Coupling Between Microservice (CBM)

CBM is a count of unique outbound interactions to other microservices. Table 16 shows the results for the CBM metric for the system.

Table 16. CBM results

Service	CBM	Comment
Location Service	0	no interactions

Analytics Service	1	receives data from Booking Service
Booking Service	2	interacts with Parking Management and Payment Services
Payment Service	1	interacts with User Management Service
Parking Management Service	1	interacted with by Booking Service
User Management Service	1	interacted with by Payment Service

The CBM values highlight the level of coupling in the system. Services with CBM=0 are highly independent and modular, mitigating risk of cascading failures or complex dependencies. Services with higher CBM values are may require careful management to maintain stability and ease of changes. The results can guide efforts to refactor or re-architect the system to reduce coupling and enhance maintainability.

2. Instability Index

Instability Index can be calculated as:

$$I = \frac{C_e}{C_a + C_e},$$

C_a(Afferent Coupling): The number of incoming dependencies to the service from other services.

C_e (Efferent Coupling): The number of outgoing dependencies from the service to other services.

Table 17 shows the results for the system's Instability Index metric.

Table 17. Instability Index results

Service	C_e	Comment	C_a	Comment	I	Comment
Location Service	0	Does not call any other service	0	No other services depend on it directly	0	Maximally stable
Analytics Service	0	Sends data but no direct service calls	1	Receives data from Booking Service via Kafka, if considered a dependency	0	Stable
Booking Service	2	Calls Parking Management and Payment services	1	Analytics depends on Booking directly	0.6	Risk of instability
Payment Service	1	Calls User Management service	1	Called by Booking Service	0.5	Risk of instability
Parking Management Service	0	Does not call any other services	1	Called by Booking Service	0	Stable
User Management Service	0	Does not call any other services	1	Called by Payment Service		Stable

The Instability Index helps in understanding how changes in one service might affect others and vice versa. A high instability value (close to 1) suggests a service is likely to change due to its dependencies on other services, indicating potential design concerns in terms of resilience and robustness. In contrast, a low value (close to 0) indicates stability and fewer concerns about the impact of changes in other services. This metric is crucial for assessing the maintainability and scalability of microservices architectures.

Overall MBS evaluation results discussion:

Domain Design Metrics: The application of BCP and MFD provided insightful results about the alignment of services with business functions and the optimal distribution of functionalities across services. This demonstrated that the metrics are capable of revealing significant aspects of service design, including potential areas for improvement in service boundaries and functional alignment.

Data Design Metrics: SDR effectively measured the data dependencies between services. It highlighted services with high data autonomy and those with potential over-dependencies, guiding necessary adjustments in service interactions and data management strategies to enhance system modularity and reduce coupling.

API and Communication Design Metrics: Metrics like the NPE and ARI were useful in assessing the API's complexity and resilience. The communication metrics, including ICP and DEE, helped quantify the efficiency of service interactions and the appropriateness of the data shared across interfaces.

Essential Characteristics (Coupling, etc.): The analysis of essential microservices characteristics through metrics such as CBM and Instability Index provided a quantitative basis to assess system architecture in terms of stability and maintainability.

7 Result and Discussion

This section presents a comprehensive analysis of the results obtained from applying the newly developed metrics-based approach to evaluate the quality of MBS.

7.1 Impact

This approach facilitated a deeper understanding of operational dynamics and fostered better alignment with business goals, ensuring that each system component contributes optimally to overall objectives. The insights gained from this approach have led to more resilient, efficient, and adaptable systems, demonstrating the effectiveness of applying a rigorous, metrics-driven methodology to the design and evaluation of microservices architectures.

7.1.1 Quantitative and Qualitative Insights

The application of the developed metrics to various microservices configurations provided both quantitative and qualitative insights into the architecture and operational dynamics of MBS. This analysis offered a deep dive into how these metrics can influence the understanding and enhancement of MBS quality in practical settings.

Quantitative Insights:

- Domain Design Metrics quantitatively assessed how effectively services encapsulate business logic, directly impacting strategic alignment between business objectives and technical execution. This helped in quantifying the degree to which services adhere to defined business functions.
- Data Design Metrics provided numerical data on consistency and integrity across services, preventing latency issues that could impact system performance.
- API Design Metrics provided measurements on interoperability and the robustness of service interfaces, offering quantitative data on API effectiveness and integration capabilities.
- Communication Metrics evaluated the reliability of data exchanges, providing hard data on communication efficiency and bottlenecks.
- Essential Metrics showed architectural robustness through metrics on coupling, cohesion, and modularity, delivering quantifiable indicators of architectural health and system adaptability.

Qualitative Insights:

- Domain Design Metrics offered insights into the practical implications of aligning service design with business logic, enhancing the qualitative understanding of strategic technology implementation.
- Data Design Metrics improved the qualitative assessment of data management practices across distributed environments, underscoring areas for improvement in handling data integrity and synchronization.
- API Design Metrics helped qualitatively evaluate how service interfaces facilitate seamless integration and communication between services, critical for maintaining system integrity and user satisfaction.
- Communication Metrics provided a qualitative understanding of how effectively the system's communication architecture supports dynamic data flows and interactions.

- Essential Metrics offered a deeper look into the fundamental principles guiding system design and how well the microservices architecture supports principles of modern software engineering practices such as scalability, maintainability, and independence.

Through the dual lens of quantitative data and qualitative analysis, the metrics fostered a comprehensive evaluation of MBS, enabling targeted improvements and fostering a robust understanding of system capabilities and areas for enhancement.

7.1.2 Metrics Performance Evaluation

The performance of each metric was evaluated across various case studies involving different configurations of microservices. This comparative analysis highlighted the strengths and limitations of individual metrics in diverse scenarios, offering a nuanced understanding of their applicability and reliability.

- Domain Design Metrics demonstrated their effectiveness in encapsulating the business logic consistency across services, enhancing the strategic alignment of technology and business goals.
- Data Design Metrics were pivotal in assessing data consistency and integrity across distributed services, highlighting issues in data synchronization and latency.
- API Design Metrics provided insights into the service interoperability and the robustness of service interfaces, which are critical for ensuring seamless service integration.
- Communication Metrics were instrumental in evaluating the efficiency and effectiveness of data exchanges between microservices. These metrics helped pinpoint potential bottlenecks in data flows and assessed the impact of communication patterns on system performance. They also facilitated the optimization of message routing and prioritization, ensuring that communication protocols were not only robust but also optimized for low latency and high throughput.
- Essential Metrics focused on fundamental architectural qualities such as service cohesion, coupling, and modularity. These metrics proved crucial for assessing the architectural soundness of microservices systems. They provided a quantifiable measure of how well services were isolated from each other, which in turn influenced their ability to be developed, deployed, and scaled independently. Essential Metrics also helped evaluate the system's ability to adapt to changes without extensive rework, thus supporting continuous integration and deployment practices.

7.2 Case Study Insights

In evaluating the metrics-based approach to MBS, the case study involving a PMS provided crucial insights into the practical application and effectiveness of the proposed metrics.

7.2.1 Application of Metrics

The metrics were applied to various components of the PMS, encompassing services such as Location Service, Booking Service, User Management Service, Payment Service, Parking Management Service, and Analytics Service. Each of these services was analyzed based on established metrics like Business Context Purity (BCP), Microservice Functional Density (MFD), Shared Data Rate (SDR), and others to evaluate their design, functionality, and interaction within the system.

Business Context Purity and Functional Density: The application of BCP and MFD highlighted areas where services could be better aligned with business objectives. For in-

stance, the Booking Service and Parking Management Service scored higher on the MFD, indicating a complex aggregation of functionalities that might benefit from decomposition to enhance modularity and maintainability.

Data Independence: The SDR metric revealed the degree of data coupling between services. Services with low SDR values, such as the Location Service, demonstrated high data autonomy, which is ideal for reducing dependencies and potential bottlenecks in a microservices architecture.

7.2.2 Insights from Service Interactions

The evaluation process also describes the interaction dynamics between different services within the PMS. Metrics like the API Resilience Index (ARI) and Inter-Partition Call Percentage (ICP) provide data on how services communicate and handle failures, which is critical for the stability and reliability of the system.

Communication and Coupling: The ICP results indicated that certain services, such as the Booking Service, were central to the communication within the system, suggesting a potential risk area for bottlenecks. Applying communication-related metrics also helped identify services that could benefit from enhanced message persistence and more efficient data-handling strategies.

Resilience and Fault Tolerance: The ARI scores helped assess the robustness of service APIs, particularly their ability to handle high loads and recover from failures. Services with lower ARI scores were identified as potential points of failure that could benefit from implementing resilience patterns such as circuit breakers and rate limiting.

7.2.3 Practical Implications and Recommendations

The insights gained from applying the metrics to the PMS case study have several practical implications for the design and operation of MBS. Based on the findings, the following recommendations were made:

1. **Service Decomposition**: Services with high functional density should be evaluated for potential decomposition to distribute functionalities more evenly, which could improve scalability and ease of maintenance.
2. **Enhancing Data Autonomy**: Services with high SDR values should consider strategies to reduce data dependencies, such as implementing event-driven architectures or service-specific databases, to enhance autonomy and reduce coupling.
3. **Strengthening Service Resilience**: Implementing resilience patterns more broadly across services could mitigate the impact of failures and enhance the overall stability of the system.
4. **Balancing Communication Efficiency**: Optimizing the use of data transfer objects (DTOs) and refining message communication strategies could lead to more efficient interactions between services, reducing overhead and improving performance.

7.2.4 Validity of Metrics

The case study not only provided a practical assessment of the proposed metrics but also validated their relevance and utility in a real-world setting. The metrics effectively high-

lighted critical areas of concern and facilitated targeted improvements in the system architecture. This validation supports the robustness of the metrics-based approach and underscores its applicability to other MBS projects.

Applying the metrics-based approach within the PMS case study has demonstrated its effectiveness in identifying key areas for improvement and validating the design decisions in microservices architectures. These insights are crucial for developers and architects aiming to optimize microservices for better loose coupling, resilience, and business alignment. The case study thus serves as a valuable example of how theoretical metrics can be effectively translated into practical improvements in software architecture.

8 Limitations

This section discusses the limitations encountered in the research and the inherent constraints of the metrics-based approach proposed for evaluating the quality of MBS.

In this thesis, we provide a preliminary validity check for the comprehensiveness of the proposed approach, which needs to be complemented in the future with empirical validation through controlled studies and feedback from experts.

8.1 Future work

- Enhancing the Metrics: Further research could focus on refining the metrics to cover more qualitative aspects and reduce dependency on extensive data.
- Broader Validation: Applying the metrics to a wider range of case studies could enhance their validity and generalizability. This might also reveal areas of improvement in the approach and its corresponding metrics.
- Usability assessment: due to time limitations, the author did not have sufficient time to assess the usability of the developed approach, i.e., identify any usability issues or challenges that software engineers may encounter while using the approach. However, such an assessment is on our plans for future work.

In summary, while the metrics-based approach provides valuable insights into the quality of MBS, it is not without its limitations. Recognizing these helps in understanding the boundaries within which the findings of this research can be effectively applied.

Conclusions

This thesis has successfully developed and evaluated a metrics-based approach for evaluating the quality of MBS, leveraging a comprehensive SLR and a multi-dimensional evaluation approach. The research synthesized existing knowledge and filled critical gaps with newly developed metrics, offering a robust toolset for practitioners aiming to enhance the architecture and operation of MBS.

The SLR was instrumental in identifying and consolidating the existing body of knowledge on microservices evaluation. This foundational work informed the development of a multi-dimensional evaluation framework that captures the characteristics of MBS. By integrating insights from the SLR, the study ensured that the metrics were grounded in well-established principles and reflected the latest developments in the field.

The multi-dimensional evaluation approach proposed in this thesis is one of its most significant contributions. It addresses various aspects of MBS quality, from business alignment and data management to service interaction and API robustness. This comprehensive approach enhances the depth of evaluation and supports the holistic improvement of MBS by providing detailed insights across multiple layers of system architecture.

This research has created a versatile and powerful toolkit for assessing MBS by combining existing metrics with newly developed ones. The existing metrics provided a solid baseline, ensuring continuity and comparability with traditional approaches. In contrast, the newly developed metrics addressed specific gaps and introduced novel perspectives crucial for modern microservices environments. This integration enables a more nuanced analysis and facilitates a targeted approach to system enhancements.

The practical application of these metrics through a case study demonstrated their effectiveness in a real-world scenario.

The findings from this thesis underscore the importance of continuous evaluation and refinement of MBS. The metrics-based approach not only aids in the initial assessment but also supports ongoing adjustments and enhancements, contributing to the system's adaptability and sustainability. Future research can expand on this work by exploring additional evaluation dimensions, integrating emerging technologies, and further refining the metrics to enhance their predictive power and applicability across diverse operational contexts.

The research presented in this thesis provides an advancement in software engineering by offering a systematic, metrics-based approach to evaluate and refine MBS. It equips developers and architects with the tools necessary to assess their MBS quality comprehensively and implement improvements that align with both technical requirements and business goals.

References

[1] Newman, S. (2015). Building Microservices. Sebastopol, CA: O'Reilly Media, Inc. ISBN: 978-1-491-95035-7.

[2] Richardson, C. (2018). Microservices Patterns: With examples in Java. Manning Publications.

[3] Bogner, J., Wagner, S., & Zimmermann, A. (2017). Towards a practical maintainability quality model for service-and microservice-based systems. In Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings.

[4] Alshuqayran, N., Ali, N., & Evans, R. (2016). A systematic mapping study in microservice architecture. In 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA). IEEE.

[5] Rodgers, Peter (Feb 15, 2005). "Service-Oriented Development on NetKernel- Patterns, Processes & Products to Reduce System Complexity". CloudComputingExpo. SYS-CON Media. Archived from the original on 20 May 2018.

[6] Lewis J., Fowler M. (2014) Microservices - a definition of this new architectural term. https://martinfowler.com/articles/microservices.html. Accessed 02.12.2023

[7] M. Mekki, N. Toumi and A. Ksentini, "Microservices Configurations and the Impact on the Performance in Cloud Native Environments,"*2022 IEEE 47th Conference on Local Computer Networks (LCN)*, Edmonton, AB, Canada, 2022, pp. 239-244, doi: 10.1109/LCN53696.2022.9843385.

[8] Amazon Web Services, Inc. (n.d.). *General design principles*. AWS Well-Architected. Retrieved 15.03.2024, from https://docs.aws.amazon.com/wellarchitected/latest/framework/general-design-principles.html

[9] Cockcroft, A. (2014). State of the art in microservices. DockerCon. Retrieved from https://www.slideshare.net/adriancockcroft/dockercon-state-of-the-art-in-microservices

[10] Taibi, D., Lenarduzzi, V., & Pahl, C. (2018). Architectural Patterns for Microservices: A Systematic Mapping Study. In Proceedings of the 8th International Conference on Cloud Computing and Services Science (CLOSER 2018), pp. 221-232. DOI: 10.5220/0006798302210232. Tampere University of Technology, Tampere, Finland; Free University of Bozen-Bolzano, Bozen-Bolzano, Italy. ISBN: 978-989-758-295-0.

[11] Fowler M. (2015) Microservice Trade-Offs. https://martinfowler.com/articles/microservice-trade-offs.html Accessed 02.12.2023

[12] Zimmermann, O. (2017). Microservices Tenets: Agile Approach to Service Development and Deployment. Comput Sci Res Dev, 32, 301–310. DOI: 10.1007/s00450-016-0337-0. Springer-Verlag Berlin Heidelberg

[13] Zdun, U., Navarro, E., & Leymann, F. (2017). Ensuring and Assessing Architecture Conformance to Microservice Decomposition Patterns. In M. Maximilien, A. Vallecillo, J. Weber, & J. Wuttke (Eds.), Service-Oriented Computing – ICSOC 2017 Lecture Notes in Computer Science (Vol. 10601, pp. 411–429). Springer, Cham.

https://doi.org/10.1007/978-3-319-69035-3_29

[14] Zafar, F. (2022). Investigating Quality Attributes and Best Practices of Microservices Architectures. RWTH Aachen University, MS.

[15] Hasan, M. H., Osman, M. H., Admodisastro, N. I., Muhammad, M. S. (2023). From Monolith to Microservice: Measuring Architecture Maintainability. International Journal of Advanced Computer Science and Applications, 14(5).

[16] Hevner, March, Park, Ram (2017) Design Science in Information Systems Research. MIS Quarterly 28(1):75, DOI 10.2307/25148625.

[17] Wieringa R (2009) Design science as nested problem solving. Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, DESRIST '09 pp 1–12, DOI 10.1145/1555619.1555630.

[18] Bell D, De Cesare S, Iacovelli N, Lycett M, Merico A (2007) A framework for deriving semantic web services. Information Systems Frontiers 9(1):69–84, DOI 10.1007/s10796-006-9018-z, arXiv:1011.1669v3.

[19] Kitchenham, B. Procedures for performing systematic reviews. Keele, UK, Keele University 33, TR/SE-0401 (2004), 28.

[20] Nadareishvili, I., Mitra, R., McLarty, M., & Amundsen, M. (2016). Microservice Architecture. Sebastopol, CA: O'Reilly Media, Inc.

[21] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday, today, and tomorrow. Present and ulterior software engineering, 195-216.

[22] Bajaj, D., Goel, A., & Gupta, S. C. (2022). GreenMicro: Identifying Microservices From Use Cases in Greenfield Development. IEEE Access. https://doi.org/10.1109/ACCESS.2022.3182495

[23] De, B. (2023). API Management: An Architect's Guide to Developing and Managing APIs for Your Organization. [Publisher]. https://doi.org/10.1007/979-8-8688-0054-2

[24] Apel, S., Hertrampf, F., Späthe, S. (2019). Towards a Metrics-Based Software Quality Rating for a Microservice Architecture. In Communications in Computer and Information Science, vol. 1041. Springer, Cham. https://doi.org/10.1007/978-3-030-22482-0_15

[25] Knoll, N., & Lichtenthäler, R. (2023). An Experimental Evaluation of Relations Between Architectural and Runtime Metrics in Microservices Systems. In Proceedings of the 13th International Conference on Cloud Computing and Services Science - CLOSER. Setúbal: SciTePress. pp. 147-154. DOI: 10.5220/0011728600003488

[26] Lichtenthaler, R., Wirtz, G. (2022). Towards a Quality Model for Cloud-native Applications. In F. Montesi et al. (Eds.), European Conference on Service-Oriented and Cloud Computing (ESOCC 2022), Lecture Notes in Computer Science (Vol. 13226, pp. 109–117). Springer, Cham. https://doi.org/10.1007/978-3-031-04718-3_7

[27] Ntentos, E., Zdun, U., Plakidas, K., Meixner, S., & Geiger, S. (2020). Metrics for Assessing Architecture Conformance to Microservice Architecture Patterns and Practices. In Proceedings of the 2020 IEEE 22nd Conference on Business Informatics (CBI), Vol. 01, pp. 119-128. IEEE. https://doi.org/10.1109/CBI49978.2020.00022

[28] Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. MIS Quarterly: Management Information Systems, 28(1), 75–105. https://doi.org/10.2307/25148625

[29] Venable, J., Pries-Heje, J., & Baskerville, R. (2012). A comprehensive framework for evaluation in design science research. International Conference on Design Science Research in Information Systems, 423–438.

[30] Carneiro Jr., C., & Schmelmer, T. (2016). Microservices from Day One: Build robust and scalable software from the start. Hollywood, Florida, USA: Broomfield, Colorado, USA. ISBN-13 (pbk): 978-1-4842-1936-2. ISBN-13 (electronic): 978-1-4842-1937-9. DOI: 10.1007/978-1-4842-1937-9. Library of Congress Control Number: 2016961230.

[31] Engel, Thomas, et al. "Evaluation of microservice architectures: A metric and tool-based approach." Information Systems in the Big Data Era: CAiSE Forum 2018, Tallinn, Estonia, June 11-15, 2018, Proceedings 30. Springer International Publishing, 2018.

[32] Wang, Y., Kadiyala, H., & Rubin, J. (2021). Promises and challenges of microservices: an exploratory study. Empirical Software Engineering, 26(63). https://doi.org/10.1007/s10664-020-09910-y

[33] 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. In 2020 IEEE International Conference on Software Architecture (ICSA). https://doi.org/10.1109/ICSA47634.2020.00023

[34] Selmadji, A., Seriai, A.-D., Bouziane, H. L., Mahamane, R. O., Dony, C. (2018). Re-architecting OO Software into Microservices: A Quality-Centred Approach. In European Conference on Service-Oriented and Cloud Computing (pp. 65–73). Springer, Cham. https://doi.org/10.1007/978-3-319-99819-0_5

[35] Taibi, D., Systä, K. (2020). A Decomposition and Metric-Based Evaluation Framework for Microservices. In D. Ferguson et al. (Eds.), Communications in Computer and Information Science (Vol. 1218, pp. 133–149). Springer, Cham. https://doi.org/10.1007/978-3-030-49432-2_7

[36] Bruce, M., & Pereira, P. A. (2019). Microservices in Action. ISBN 9781617294457.

[37] Abdelfattah, A.S.; Cerny, T. Roadmap to Reasoning in Microservice Systems: A Rapid Review. Appl. Sci. 2023, 13, 1838. https://doi.org/10.3390/app13031838

[38] Chapman, M., G-Medhin, A., Sassoon, I., Kokciyan, N., Sklar, E. I., Curcin, V. (2022). Using Microservices to Design Patient-facing Research Software. In 2022 IEEE 18th International Conference on e-Science (e-Science). https://doi.org/10.1109/ESCIENCE55777.2022.00019

[39] Bogner, J., Fritzsch, J., Wagner, S., & Zimmermann, A. (2019). Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality. In 2019 IEEE International Conference on Software Architecture Companion (ICSA-C) (pp. 187-195). IEEE. doi:10.1109/ICSA-C.2019.00041.

[40] Thatikonda, V.K. (2023). Assessing the Impact of Microservices Architecture on Software Maintainability and Scalability. European Journal of Theoretical and Applied Sci-

ences, 1(4), 782-787. DOI: 10.59324/ejtas.2023.1(4).71

[41] Velepucha, V., & Flores, P. (2023). A Survey on Microservices Architecture: Principles, Patterns, and Migration Challenges. IEEE Access. Digital Object Identifier: 10.1109/ACCESS.2023.3305687.

[42] Zhong, C., Zhang, H., Li, C., Huang, H., Feitosa, D. (2023). On measuring coupling between microservices. The Journal of Systems and Software, 200, 111670. https://doi.org/10.1016/j.jss.2023.111670

[43] Vale, G., Rosa, T. de O., Correia, F. F., Fritzsch, J., Guerra, E. M., & Bogner, J. (2022). Designing Microservice Systems Using Patterns: An Empirical Study on Quality Trade-Offs. In 2022 IEEE 19th International Conference on Software Architecture (ICSA). https://doi.org/10.1109/ICSA53651.2022.00015

[44] Márquez, G., & Astudillo, H. (2018). Actual Use of Architectural Patterns in Microservices-Based Open Source Projects. In Proceedings of the 25th Asia-Pacific Software Engineering Conference (APSEC 2018). Nara, Japan. DOI:10.1109/APSEC.2018.00017.

[45] Vera-Rivera FH, Gaona C, Astudillo H. 2021. Defining and measuring microservice granularity—a literature overview. PeerJ Comput. Sci. 7:e695 DOI 10.7717/peerj-cs.695

[46] Cojocaru, M.-D., Uta, A., Oprescu, A.-M. (2019). Attributes Assessing the Quality of Microservices Automatically Decomposed from Monolithic Applications. In 2019 18th International Symposium on Parallel and Distributed Computing (ISPDC). https://doi.org/10.1109/ISPDC.2019.00021

[47] Pulnil, S., Senivongse, T. (2022). A Microservices Quality Model Based on Microservices Anti-patterns. In 2022 19th International Joint Conference on Computer Science and Software Engineering (JCSSE). https://doi.org/10.1109/JCSSE54890.2022.9836297

[48] Tapia, V. C., Gaona, C. M. (2023). Research Opportunities in Microservices Quality Assessment: A Systematic Literature Review. Journal of Advances in Information Technology, 14(5), 991-1002. https://doi.org/10.12720/jait.14.5.991-1002

[49] Vera-Rivera, F. H., Puerto, E., Astudillo, H., Gaona, C. (2021). Microservices Backlog: A Genetic Programming Technique for Identification and Evaluation of Microservices From User Stories. IEEE Access. https://doi.org/10.1109/ACCESS.2021.3106342

[50] Jin, W., Liu, T., Cai, Y., Kazman, R., Mo, R., Zheng, Q. (2021). Service Candidate Identification from Monolithic Systems Based on Execution Traces. IEEE Transactions on Software Engineering, 47(5), 987-1004. https://doi.org/10.1109/TSE.2019.2910531

[51] 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

[52] Valdivia, J. A., Lora-González, A., Limón, X., Cortes-Verdín, K., & Ocharán-Hernández, J. O. (2020). Patterns Related to Microservice Architecture: a Multivocal Literature Review. Programming and Computer Software, 46(8), 594–608. Pleiades Publishing, Ltd. DOI: 10.1134/S0361768820080253.

[53] Cardarelli, M., Iovino, L., Di Francesco, P., Di Salle, A., Malavolta, I., Lago, P. (2019). An Extensible Data-Driven Approach for Evaluating the Quality of Microservice Archi-

tectures. In The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3297280.3297400

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

[55] Yilmaz, R., & Buzluca, F. (2021). A Fuzzy Quality Model to Measure the Maintainability of Microservice Architectures. In Proceedings of the 2nd International Informatics and Software Engineering Conference (IISEC 2021). IEEE. DOI: 10.1109/IISEC54230.2021.9672417.

[56] Al-Debagy, O., & Martinek, P. (2020). A Metrics Framework for Evaluating Microservices Architecture Designs. Journal of Web Engineering, 19(3-4), 341–370. https://doi.org/10.13052/jwe1540-9589.19341

[57] Bogner, J., Wagner, S., & Zimmermann, A. (2017). Automatically Measuring the Maintainability of Service- and Microservice-based Systems – a Literature Review. In Proceedings of the International Workshop on Software Measurement and International Conference on Software Process and Product Measurement (IWSM Mensura). https://doi.org/10.1145/3143434.3143443

[58] Mihai, I. S. (2023). A Systematic Evaluation of Microservice Architectures Resulting from Domain-Driven and Dataflow-Driven Decomposition. In 2023 IEEE 37th International Conference on Software Engineering and Advanced Applications (SEAA). https://doi.org/10.1109/SEAA54165.2023.00021

[59] Filippone, G., Qaisar, N. M., Autili, M., Rossi, F., & Tivoli, M. (2023). From Monolithic to Microservice Architecture: An Automated Approach Based on Graph Clustering and Combinatorial Optimization. In 2023 IEEE 20th International Conference on Software Architecture (ICSA). https://doi.org/10.1109/ICSA56044.2023.00013

[60] Wilta, A., & Kistijantoro, A. I. (2023). Automatic Measurement of Microservice Architecture Quality with Cohesion, Coupling, and Complexity Metrics. In 2023 10th International Conference on Advanced Informatics: Concept, Theory and Application (ICAICTA). IEEE. https://doi.org/10.1109/ICAICTA59291.2023.10390525

[61] Rosa, T. de O., Goldman, A., & Guerra, E. M. (2020). How 'Micro' Are Your Services? In 2020 IEEE International Conference on Software Architecture Companion (ICSA-C). https://doi.org/10.1109/ICSA-C50368.2020.00023

[62] Raj, V., & Ravichandra, S. (2018). Microservices: A Perfect SOA-Based Solution for Enterprise Applications Compared to Web Services. In 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT-2018). IEEE.

[63] Hirzalla, M., Cleland-Huang, J., & Arsanjani, A. (2009). A Metrics Suite for Evaluating Flexibility and Complexity in Service Oriented Architectures. In G. Feuerlicht & W. Lamersdorf (Eds.), Service-Oriented Computing - ICSOC 2008 Workshops (pp. 41–52).

Springer-Verlag. Lecture Notes in Computer Science, vol 5472. https://doi.org/10.1007/978-3-642-01247-1_5

Appendix

I. Microservice principles (literature review)

Table 18 presents a consolidated view of microservice principles as identified through the literature review, combined with similar terms.

Table 18. Merged principles.

Principle	Mentioned by
Size / Small	[1], [20], [21], [30], [31], [32], [33], [34], [35]
Focused on Doing One Thing Well / Single Responsibility / Goal-oriented, Limited scope	[1], [20], [30], [31], [32], [33], [34], [35], [36], [37], [38]
Autonomous / Independency	[1], [15], [20], [21], [31], [33], [34], [36], [37]
Technology Heterogeneity / Technology Diversity / Interoperability	[1], [10], [11], [20], [30], [31], [32], [33], [34], [35], [37], [38], [39]
Ease of Deployment (key benefit)	[1], [10], [15], [31], [33], [34], [35], [38], [39]
Organizational Alignment (key benefit)	[1], [20], [40]
Composability (key benefit)	[1], [20], [38]
Optimizing for Replaceability / Self-Healing	[1], [10], [20], [31], [38], [41]
Loose coupling	[1], [15], [30], [31], [32], [36], [42]
High cohesion / Cohesity	[1], [15], [20], [31], [42]
Modeled around business concepts (principle) / Organized around Business Capabilities (characteristics) / Strong Module Boundaries / Physical isolation, Modularity, Bounded context [6], Alignment [21]	[1], [6], [10], [11], [15], [20], [21], [30], [31], [32], [36], [38], [39], [41]
Culture of automation (principle)	[1], [20], [36], [39], [41]
Hide internal implementation details (principle)	[1], [41]
Decentralize all the things (principle) ⊂ Decentralized Data Management, Decentralized Governance	[1], [6], [20], [30], [31], [36], [39], [40], [41]
Deploy independently / Independent Deployment / Infrastructure Automation	[1], [6], [11], [20], [30], [31], [32], [36], [39], [41]
Isolate failure / Design for failure / Resiliency	[1], [6], [10], [20], [31], [32], [35], [36], [37], [38], [40], [41]
Observability / Transparency (logs and metrics)	[1], [10], [36], [41]
Componentization via Services (characteristics)	[6], [10], [32], [39], [40], [41]
Products not Projects (characteristics)	[6], [20], [30], [31], [32], [39], [40]
Smart endpoints and dumb pipes (characteristics) / Messaging enabled (important characteristic) / Message-based communication	[6], [20], [32], [36], [40]
Evolutionary Design (characteristics)	[6], [21]
Maintainability	[10], [15], [30], [35], [40], [43]
Unlimited application size	[10]
Flexibility	[21]
Efficiency	[20], [31]
Scalability	[15], [20], [31], [32], [35], [36], [37], [38], [40], [43]
Testability	[20], [40]

Reliability	[20], [31], [32]
Availability	[20], [30], [32]
Antifragility	[20]
Immutability	[20]
Agility	[1], [2], [43]
Performance	[43]

II. Quality attributes (literature review)

Table 19 presents a consolidated view of quality attributes and their alternatives as identified through the literature review.

Table 19. Quality attributes.

Attribute	Source	Alternative
Scalability	[4], [44], [45], [46], [47], [48], [49]	Expandable, evolutionary
Independence	[4], [26], [46], [48], [50], [51]	Reducing complexity, isolation, loose coupling, decouple, distributed, containerization, autonomy
Maintainability (ISO 25010)	[3], [4], [14], [21], [26], [39], [44], [45], [47], [48], [49], [52], [53], [54]	Expandable, adaptability, changeability, flexible implementation, dynamically changing
Deployment	[4], [51]	Deployability
Health management	[4], [21], [26], [44], [46]	Resiliency, reliability, disaster recovery, no single point of failure
Modularity	[4], [26], [45], [47], [49], [50], [51], [55]	Single responsibility, reduce complexity, separate business concern, specialization, customizable
Manageability	[4]	Self-managed, decentralized management, audibility
Performance (ISO 25010)	[4], [14], [21], [26], [39], [44], [45], [46], [48], [49], [52], [54]	Response times, transaction duration, throughput, efficiency
Reusability	[4], [26], [47]	Pluggable
Technology heterogeneity	[4], [26], [39]	Freedom to choose a lot of technologies or programming languages
Agility	[4]	Iterative, incremental, continuous delivery
Security (ISO 25010)	[4], [14], [21], [26], [39], [45], [46], [47], [48], [49], [51], [52], [54]	
Organizational alignment	[4]	Cross-functional team, reduce the conflict between developers and testers
Open interface	[4], [26]	Microservices should provide an open description of their APIs, GUIs and communication messages format
Availability	[21], [26], [44], [45], [49], [51]	
Observability	[26], [44]	
Testability	[21], [26], [51], [55]	
Portability (ISO 25010)	[14], [21], [26], [48], [52], [54]	
Reliability (ISO 25010)	[14], [26], [39], [45], [47], [48], [49], [52], [54]	Fault-tolerance
Compatibility (ISO 25010)	[14], [26], [39], [48], [52], [54]	
Usability (ISO 25010)	[14], [39]	
Functional suitability (ISO 25010)	[14], [39]	
Functionability	[45], [49]	
Granularity	[46], [48]	
Cohesion	[46], [48]	

Evolvability	[50]	
Understandability	[47]	

III. Metrics (literature review)

Table 20 presents a consolidated view of identified metrics through the literature review for assessing MBS and their mapping to identified earlier principles and quality attributes.

Table 20. Mapping principles, quality attributes and metrics.

Principle	Quality attribute	Metrics
Goal-oriented	Functional suitability (ISO 25010)	Business Context Purity (BCP) [22]
	Granularity	The Service Granularity Metric "SGM" [56] Weighted Service Interface Count (WSIC) [3], [45], [57], [58] Component Balance (CB) [45] Granularity metric (Gm) [49] Focused on One Functionality [33], [34] Ratio of Right Cuts [47] Ratio of Coarse-Grained Microservices [47]
Autonomous	Independence	Inter-Partition Call percentage (ICP) [22] Structural and Behavioral Autonomy of a Microservice [33], [34] Relying on Structural and Behavioral Dependencies [33] Frequency of External Calls (FEC) [35] Independence of Functionality [50] Ratio of Resolved Endpoints [47] Ratio of Separate Databases [47]
	Data Autonomy	Computing Fintra [33], [34] Computing Finter [33], [34] Computing DataDepend [33], [34]
Interoperability	Compatibility (ISO 25010)	Time, percentage of requests accepted [54] Ratio of Versioned APIs [47]
Composability	Modularity	Structural Modularity (SM) [22] Precision [59] Recall [59] Structural Modularity Quality [50] Conceptual Modularity Quality [50] Ratio of Manageable Connections [47] Ratio of Non-ESB Microservices [47]
High cohesion	Cohesion	The Lack of Cohesion Metric "LCOM" [48], [56], [60] Relation Cohesion (RC) [22] Service Interface Data Cohesion (SIDC) [3], [57], [58], [26] Service Interface Usage Cohesion (SIUC) [3], [57], [58], [26] Total Service Interface Cohesion (TSIC) [57] Average cohesion [59] Relational Cohesion (RCOH) [55] Cohesion (CohT) [49] Normalize Cohesion among Classes of Microservice (NCAM) [15] Intra Microservice Coupling (IMC) [15] Internal cohesion [33], [34] Cohesion at domain level [50]
Loose coupling	Coupling	Coupling between objects [48], [51]

		Number of Incoming Dependencies / Afferent coupling (Ca) [22], [55] Number of Outgoing Dependencies / Efferent coupling (Ce) [22], [55] Absolute Importance of the Service (AIS) [3], [25], [57] Absolute Dependence of the Service (ADS) [3], [57] Absolute Criticality of the Service (ACS) [25], [57], [60] Average coupling [59] Service Coupling Metric [61] Inter-Service Communication Complexity Metric [61] Service Coupling Factor(SCF) [62] Coupling (CpT) [33] Coupling Between Microservice (CBM) [15] Weighted Coupling between Microservice (WCBM) [15] Absolute Coupling between Microservice (ACBM) [15] Internal coupling [33], [34] External coupling [33], [34] Coupling (CBM) [17] Microservice Coupling Index [42], [48] Afferent Microservice Coupling Index (aMCI) [42] Efferent Microservice Coupling Index [42] Services Interdependence in the System [26]
	Independence	Number of Service Dependencies in Initialisation [NSDI] [24]
	Decentralization	Service Criticality [26]
Decentralization	Manageability	Ratio of Separate Libraries [47]
Independent deployment	Deployability	Build time (min) [51] Deployment pipelines [51] Deployable size [51] Cloud instances (AWS, Azure) [51] Deployment profiles [51] All components are independently deployable (CAID) [13] Ratio of components violating independent deployability to non-external components (RVID) [13] Ratio of independently deployable component clusters to non-external components (RIDC) [13]
	Portability (ISO 25010)	
Resilience	Reliability (ISO 25010)	Note: resilience was not directly mentioned in the selected papers, as it may heavily depend on runtime metrics – and it's hard to measure when it's the design stage.
Message-based communication	Interoperability	Service Messaging Persistence utilization metric (SMP) [27] Outbox/Event Sourcing utilization metric (OES) [27] Cohesion at message level [50] Number of asynchronous outgoing links [26] Service Link Persistence utilization metric [26] Percentage of accepted requests [48]
	Open interface	
Evolutionary design	Evolvability	Instability Index (I) [22] Independence of Evolvability: internal co-change fre-
	Agility	

		quency, external co-change frequency, Ratio of ECF to ICF [50] Ratio of Manageable Standards [47]
Maintainability	Maintainability (ISO 25010)	Maintainability [53]
Scalability	Scalability	Note: At the design stage, measuring the scalability of microservices-based systems can be challenging because there are no physical metrics, such as throughput or resource usage, to evaluate. Therefore, scalability was not mentioned in the selected papers.
Reliability	Reliability (ISO 25010)	Ratio of Acyclic Calls [47]
Availability	Availability	Healthcheck endpoints [51]
Size	Complexity	Proposed Number of Operations Per Microservice Metric "NOO" [56] Size of Container artefact [SCA] [24] Number of Produced Endpoints [NPE] [24], [25] Number of Consumed Endpoints [NCE] [24], [25] Non-Extreme Distribution (NED) [22] Number of Microservice Operations (NMO) [15] Total Response for Microservice (TRM) [15] Microservice Line of Code (MLOC) [15] Microservice Number of Classes (MNOC) [15] Microservice Class Distribution (MCD) [15] Number of Classes per Microservice (CLA) [35] Number of Duplicated Classes (DUP) [35] Average Number of Endpoints per Service [26] Cyclomatic Complexity [51] Cognitive Complexity [51]

IV. Realistic case (scenario)

The source code of the project is located: https://github.com/abakumova/parking-esi

The initial project was a final project in the Enterprise System Integration course - MTAT.03.229 (2022/2023).

V. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Viktoriia Abakumova**,

1. herewith grant the University of Tartu a free permit (non-exclusive license) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   A Metrics-based Approach for Evaluating the Quality of Microservices-based Systems,

   supervised by Mohamad Gharib,

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons license CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.
4. I certify that granting the non-exclusive license does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Viktoriia Abakumova

Tartu, **15.05.2024**