GDAŃSK UNIVERSITY OF TECHNOLOGY

TASK Quarterly 30 (3) 2026

Design and decomposition of Microservices Architecture: A Literature Review

Vadim Peczyński *

Gdańsk University of Technology, Faculty of Electronics, Telecommunications and Informatics, Gabriela Narutowicza 11/12, 80-233 Gdańsk, Poland

*vadim.peczynski@pg.edu.pl

https://doi.org/10.34808/tq2026/30.3/b

Abstract

Context: The adoption of Microservice Architectures has grown significantly as a result of their ability to overcome key limitations of monolithic systems, such as limited scalability, maintenance difficulties, and technological lock-in. However, the design of microservice-based systems remains a complex and non-trivial task. This underscores the need for well-defined methodologies and techniques to support practitioners and software architects throughout the microservice design process.

Objective: The objective of this study is to examine effective microservices design practices, microservice identification techniques, the tools employed to support design activities, and the factors that drive the decomposition of systems into new microservices.

Methods: A Systematic Literature Review was conducted following the guidelines established by the PRISMA framework. From 609 studies that were retrieved from the search execution, 150 were selected and analyzed to answer the research questions.

Results: This SLR contributes by examining design tools used for decomposition, analyzing microservice decomposition processes, identifying metrics for microservice decomposition, and framing microservice decomposition as a multi-objective optimization problem.

Conclusions: This study highlights the complexity and variability inherent in microservices design, as well as the limited availability of tools to support this process. The findings identify microservices communication and service decomposition as key challenges, with the majority of existing research focusing primarily on the decomposition of monolithic systems.

Keywords:

Microservices architecture, Systematic Literature Review, MSA design

1. Introduction

Microservices divide a single application into a set of independent, often remote modules. This approach offers the possibility of reducing costs by fully utilizing resources. It also makes it possible to easily scale small parts of the application vertically and horizontally instead of replicating parts that do not demand additional computational power. As described in [1], microservices are developed around business capabilities and built as independently deployable units by fully automated build machines. Centralized management should be kept to the bare minimum. In [2], Microservices Architecture (MSA) is defined as an architectural style that supports independent deployment, autonomous scaling, and improved maintainability of complex systems.

As discussed in [3], Microservices Architecture (MSA) is derived from Service-Oriented Architecture (SOA) but differs in its emphasis on smaller, more fine-grained services and a higher degree of loose coupling between them. In addition, [4–6] note that microservices typically rely on lightweight communication mechanisms such as RESTful APIs or event-driven messaging to improve interoperability. [6] also adds that these design characteristics enable microservices to operate effectively in cloud-based environments. However, according to [7], microservices also present unsolved problems such as maintainability and operational complexity.

As described in [8], changes that span multiple services are costly because the deployment process requires coordinated integration testing and careful synchronization. Insufficient coordination may result in failures due to missing or incompatible functionality. Consequently, cross-service changes should be minimized, making service decomposition a critical concern in the early stages of microservices system design.

As noted in [9], complexity and scalability issues are among the primary reasons organizations revert from microservices to monolithic architectures, often indicating shortcomings in the initial decomposition process. [10] also highlights that inappropriate partitioning of services may incur significant costs. To address these challenges, strategies such as decomposition by business capabilities and by subdomain were described in [11]. At the same time, the growing demand for tool support has led to the development of various approaches and frameworks [12–14].

The aim of this Systematic Literature Review is to establish the state of the art in microservices decomposition by analyzing existing strategies, algorithms, design tools (Event Storming and User Story Mapping), and evaluation metrics. The novelty of this SLR lies in its explicit emphasis on the design phase of microservices, rather than solely on migration or implementation concerns, offering a more holistic perspective on how systems are conceptualized from the outset. By synthesizing and critically analyzing disparate strands of research, this study not only exposes gaps in tool support and methodological guidance but also bridges the often-isolated discussions of decomposition strategies and criteria. Furthermore, it advances the field by framing microservices design as an integrated decision-making process, thereby providing a structured foundation for future research to develop more comprehensive design methodologies and practical tooling support.

The structure of this paper is organized as follows. Section 2 provides background on microservices architecture (MSA) and reviews related work, including discussions on monolithic versus microservices architectures, monolith-to-microservices decomposition, and existing literature reviews on microservices decomposition. Section 3 describes the methodology adopted for this systematic literature review, outlining the research questions, roles and responsibilities, source selection, search strategy and process, study selection, quality assessment criteria, data extraction, and data synthesis. Section 4 presents the research findings, organized into five main areas: Design tools used for decomposition, Approaches to dividing (decomposing) the system into microservices, Criteria used in the decomposition process of microservices, Multi-Objective Optimization algorithms used for decomposition of microservices, and Machine Learning algorithms used for decomposition of microservices. Section 5 discusses and interprets the results, while Section 6 addresses potential threats to the validity of the study. Finally, Section 7 summarizes the paper and outlines concluding remarks.

2. Background and related work

This study builds on existing research concerning the design of microservices-based systems, with particular attention to the challenges and strategies of software decomposition. Previous studies have investigated various decomposition approaches, techniques, and supporting tools used during migration and design processes. Furthermore, several literature reviews have summarized the existing knowledge in this area. However, the rapid evolution of microservices practices highlights the need for an updated and more design focused analysis of decomposition methods and design support techniques.

2.1. Monolithic versus Microservices Architectures

[15] defines a monolithic application architecture as one in which all, or most, modules are combined into a single large, self-contained component that operates independently of other applications. Monolithic architecture brings the benefit of localizing significant portions of application functionality in a single manageable space but, as it becomes overly large and complex, it can inhibit maintenance and deployment flexibility, described in [16]. As discussed in [15] and [17], whenever a module in a monolith changes, an entire application may need to be reintegrated, rebuilt, retested, and restarted upon deployment. Moreover, [3] also adds that the scalability of monolithic applications may introduce significant challenges as application usage grows due to size and load distribution. Microservices architectures are presented as an alternative to monolithic architectures. They are organized around business functions that perform a single task and maintain their data in a decentralized manner. They are built on the principles of single responsibility, high cohesion, low coupling, scalability, deployability, and low perturbation (meaning, minimal disturbance to other microservices that comprise the software system).

In [18], the authors compared monolithic and microservices architecture by analyzing their fundamental architectural features, practical implementations and results of load testing. The analysis was done using tools like Apache JMeter and Locust. The results of those experiments indicate that monolithic architecture is is characterized by simplicity and lower latency. Microservices, by contrast, offer superior scalability and elasticity, making them a more suitable architectural choice for systems with dynamic workloads and evolving requirements. The authors highlight the importance of a well-considered architectural choice. Additionally, in [19] the authors recommend a monolithic architecture for smaller applications, while in [20] the authors highlight the cost-effectiveness of a monolithic architecture in a specific business scenario - an e-commerce system developed by a startup. In [21], the the authors observe the superiority of microservices in terms of scalability, as well as their increasing adoption among large organizations.

[22] analyzes 168 primary studies to examine the evolution and current state of microservice identification in monolith decomposition. The authors categorize the research contributions into five groups: new tools and techniques, empirical studies, technique proposals, surveys, and datasets. The findings show that, although interest in the field is increasing, the area is still in an early stage of development, marked by fragmented data collection approaches and a lack of direct comparisons between existing methods. The study also identifies a shift in evaluation practices, with researchers moving from traditional cohesion-based metrics toward performance-oriented measures such as precision and recall, although standardized evaluation benchmarks remain absent. Among the state-of-the-art solutions identified are industrial tools such as IBM's Mono2Micro and AWS Microservice Extractor for .NET, as well as academic tools including DISCO and Kieker. The review concludes by emphasizing several unresolved challenges, particularly the need for tools that support multiple programming languages and a stronger focus on the feedback cycles involved in the microservice identification process.

Additionally, [23] provides a comprehensive analysis of semantic, meaning-aware approaches for identifying microservices through a review of 17 high-quality studies published between 2020 and 2025. The authors propose a novel classification framework that groups existing techniques into four categories: NLP-based (41%), hybrid (35%), ontology-based (18%), and AI-enhanced (6%). The study shows that semantic approaches consistently outperform traditional structural methods, which rely primarily on code dependencies and often generate service boundaries that conflict with business logic. In particular, hybrid approaches achieve the strongest results, with an average F1-score of 0.81, while domain-specific semantic models significantly improve identification accuracy. Despite these benefits, the field still faces several challenges, including inconsistent evaluation methodologies, a fragmented set of techniques, and limited industrial validation. The authors conclude that successful migration to microservices requires a fundamental shift toward semantically aware software design in order to achieve maintainable and business-aligned architectures.

2.2. Monolith to Microservices Decomposition

Many companies have chosen to continue with monolith implementations, as they have large existing monolith investments and limited knowledge of the complex process required to decompose their monoliths into microservices. As described in [15], even for the the large proportion of organizations embracing microservices, manual decomposition of the monolith application is a challenging task and may require existing application experts to devote a considerable volume of time to the tedious task of profiling and fully understanding the intricacies of the monolith code. As a complementary or alternative approach, automated code analysis tools have been used to identify service boundaries within a monolith.

The final MSA structure is achieved by decomposing the application into small, independent modules designed around a single business functionality [24]. Microser-

vices, as characterized in [16], possess well-defined boundaries and must collaborate with other modules to execute their tasks effectively and efficiently. [25] adds that the process of application decomposition has a fundamental impact on the future maintainability and development of the system. Conventional modeling notations prove insufficient for effectively capturing the structure and dynamics of microservices-based systems, as the models and communications in MSA are inherently distributed across multiple services. In [26], the authors report that the main challenges include defining services and their associated bounded contexts, analyzing asynchronous communication patterns, and performing event-driven decomposition.

2.3. Existing Literature Reviews on Microservices Decomposition

In [27], the authors conducted a rigorous examination of 35 research studies identified through a Systematic Literature Review (SLR) protocol. The objective of the study was to synthesize existing knowledge on the decomposition of monolithic systems into microservices, providing a structured and comprehensive overview of current research trends, methodologies, and gaps in the field. The paper introduces the Monolith to Microservices Decomposition Framework (M2MDF), which delineates the principal phases and elements of the decomposition process. The framework consists of six main phases: Input Collection, Monolith Analysis, Microservices Identification, Microservices Optimization, Microservices Evaluation, and Microservices Deployment. In addition to proposing this conceptual framework, the study offers a detailed analysis of existing decomposition approaches, tools, and methods. It also identifies the metrics and datasets used to validate decomposition processes and proposes several directions for future research. The findings indicate that monolith-to-microservices decomposition remains in its early stages of development. Current research lacks comprehensive methods that effectively integrate static, dynamic, and evolutionary data, resulting in fragmented approaches. Moreover, there is insufficient tool support and a notable absence of standardized metrics, datasets, and baselines for evaluating decomposition outcomes. The proposed M2MDF framework addresses these limitations by consolidating disparate research efforts, offering a unified and interconnected understanding of decomposition techniques, validation metrics, and evaluation strategies.

An SLR is carried out in [28], employing a thematic analysis of 35 studies published between 2014 and 2023 to examine the practical application of Domain-Driven Design (DDD) in microservices-based system development. The study systematizes existing knowledge on the purposes, patterns, technologies, and techniques associated with DDD in microservices architecture (MSA) and identifies five high-order and eleven specific themes related to its use. The findings reveal that microservice identification is the primary motivation for adopting DDD, which is predominantly employed for domain analysis to define appropriate microservice boundaries (strategic design, in 100% of the studies) and less commonly for detailed design (tactical design, in 54.28% of the studies). The Bounded Context pattern emerges as the most frequently used approach for modeling microservices. Overall, the study highlights that DDD facilitates the management of business domain complexity and promotes alignment between business and technical structures, effectively supporting Conway's Law.

The authors of [4] conducted an SLR of 72 primary studies to examine the evidence-based state-of-the-art regarding Quality Attributes (QAs) in microservices architecture (MSA). The study identifies six key QAs that are most frequently emphasized in MSA research: scalability, performance, availability, monitorability, security, and testability. It also categorizes 19 architectural tactics proposed to enhance these attributes. The findings indicate that scalability and performance are the most extensively addressed QAs, appearing in 23 and 21 studies respectively, whereas testability receives the least attention. Additionally, approximately 25% of the reviewed studies consider multiple Quality Attributes concurrently, highlighting their interrelated nature. While the adoption of MSA mitigates internal complexities inherent in monolithic systems, it simultaneously introduces external challenges that necessitate the use of targeted architectural tactics to ensure quality and maintain system robustness.

As described in [29], the authors conducted a comparative analysis to evaluate different monolith representations for the automated identification of microservice candidates, focusing on approaches based on code structure versus those derived from development history. The study analyzed 468,000 decompositions across 28 codebases using five quality metrics to assess the effectiveness of representations based on sequences of accesses (code structure) against those utilizing file changes and changes authorship (development history). The findings reveal that the changes authorship representation yields comparable or superior performance relative to the sequences of accesses representation, particularly in minimizing the number of transactions per functionality and optimizing team reduction. Moreover, the authorship-based approach demonstrates notably strong performance in projects with a large number of contributors, indicating its suitability for complex, collaboratively developed codebases.

In the study by [30], the authors conducted an SLR of 117 primary studies to investigate the methods, techniques, and tools used for re-engineering software systems into microservices. The study reports a significant increase in research activity in this domain, particularly between 2020 and 2022. It identifies four principal categories of re-engineering approaches: static analysis (44%), artifact-driven analysis (32%), dynamic analysis (12%), and hybrid analysis (12%). It also notes notes that coupling, cohesion, and modularity serve as key evaluation criteria for re-engineered systems. The review also highlights major paradigms underpinning re-engineering efforts, including DDD, interface analysis, workflow analysis, and runtime analysis. Despite advancements in automated techniques for microservice identification, the actual extraction process remains largely manual, underscoring the need for more effective automation and code refactoring tools in microservices re-engineering.

The authors in [31] present the results of a Multivocal Literature Review (MLR) that combines evidence from 36 academic papers and 34 grey literature sources to systematically identify and categorize security solutions for microservice-based systems. The study compiles a comprehensive catalog of security mechanisms and classifies them into 15 distinct mechanisms or scopes across four security contexts: detect, mitigate/stop, react, and recover. The findings indicate that authentication and authorization are the most frequently addressed mechanisms, with the majority of solutions focusing on the mitigate/stop context (61% in academic and 91% in grey literature). However, there remains a notable gap in research and practice concerning mechanisms aimed at reacting to or recovering from security incidents. Furthermore, the study identifies differences between academic and grey literature in their preferred empirical research strategies, emphasizing the need for more comprehensive and balanced security approaches within microservice-based architectures.

[32] presents a Systematic Mapping Study (SMS) of 114 research works that characterizes the migration of monolithic systems to microservices, including techniques, tools, drivers, challenges, and benefits. Based on the analyzed migration cases, the study proposes a structured migration process and categorizes migration techniques according to their degree of automation, input types, and granularity, while also providing a catalog of supporting tools. The findings reveal that migration is inherently complex, with most techniques being semi-automatic in nature. The primary motivations for migration are scalability and maintainability, yet available migration tools remain limited and predominantly Java-focused. Major challenges identified include monolith decomposition (the most frequently cited issue) along with microservice communication and database migration, highlighting areas that require further research and tool support to improve migration effectiveness.

3. Research methodology

The research methodology was designed to ensure a systematic and reproducible review process. It consisted of defining the research questions, developing the search strategy and search terms, establishing study selection criteria, and conducting the study selection procedure. The methodology also included a structured search and screening process to identify relevant studies related to microservices decomposition and design approaches.

3.1. Research questions

The following Research Questions (RQ) were the baseline for designing the research.

- RQ1 - Which tools can be used to design microservices?
- RQ2 - What are the approaches to divide (decompose) the system into microservices?
- RQ3 - Which criteria drive the decomposition process of microservices?
- RQ4 - Which MOO algorithms can be used for decomposition of microservices?
- RQ5 - Which ML algorithms can be used for decomposition of microservices?

3.2. Study selection

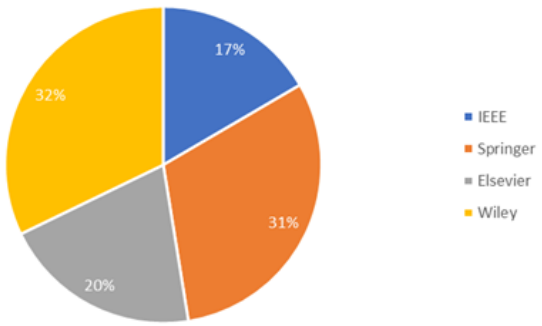

Figure 1: Distribution of the initial pool of studies.

The selection of sources for this study was guided by a set of predefined criteria defined in [33], including relevance to the software engineering domain and content availability, and was conducted in accordance with the PRISMA framework. The selected digital libraries were

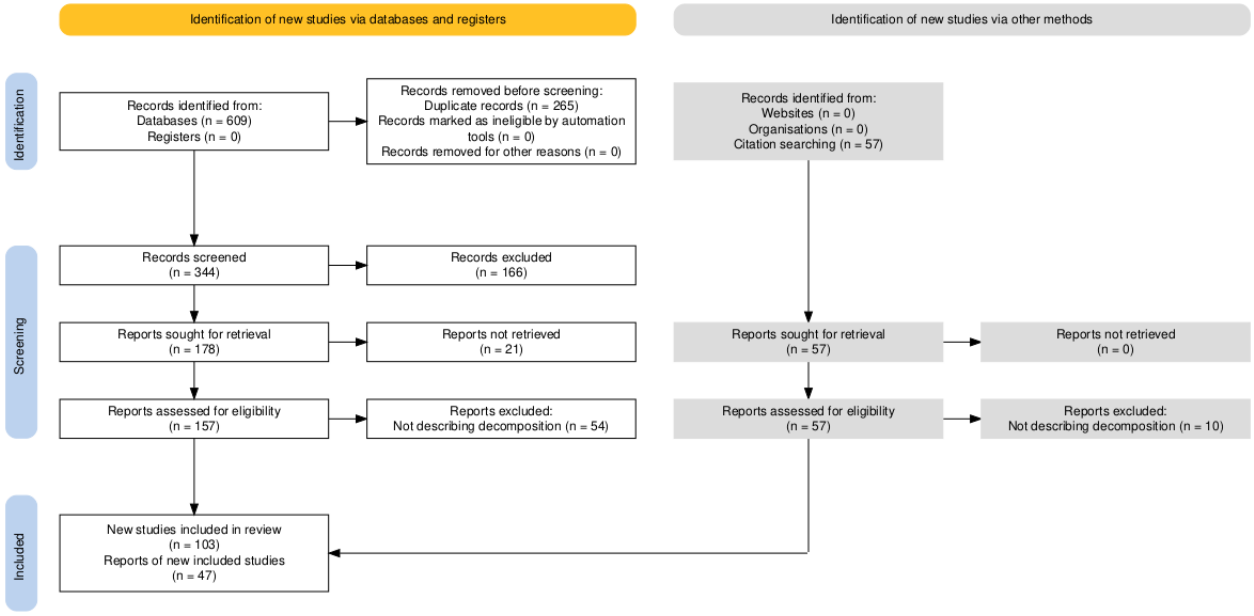

Figure 2: PRISMA Flow Diagram.

IEEE Xplore, SpringerLink, ScienceDirect, and Wiley Online Library. The final search across these sources was performed on April 12, 2026.

In total, 609 studies published between 2021 and 2026 were retrieved, forming the initial pool of candidate publications. Figure 1 illustrates the distribution of studies across the selected sources, with the majority originating from IEEE Xplore, ScienceDirect by Elsevier, and SpringerLink.

3.3. Search strategy and process

The search strategy was designed to identify studies that address the primary research questions of this review. Initially, an automated search was conducted across the previously selected data sources. The scope of the search was restricted to publications from 2021 onward. When applied to the selected bibliographic databases, this search strategy yielded a total of 609 results.

The study selection process followed the PRISMA guidelines as defined in [33]:

1. **Identification phase**, 609 records were identified through database searches across four digital libraries. An additional 57 records were identified through citation searching. Prior to screening, 265 duplicate records were removed, resulting in 344 records for screening.
2. **Screening phase**, titles and abstracts of the 344 records were reviewed, leading to the exclusion of 166 records. The remaining 178 reports were sought for retrieval, of which 21 could not be retrieved. In parallel, 57 reports identified via citation searching were sought for retrieval, with all successfully retrieved.
3. **Eligibility phase**, 157 reports from database searches and 57 reports from citation searching were assessed in full text. From the database-derived reports, 54 were excluded for not addressing the research questions. From the citation-based reports, 10 were excluded for not answering the research questions.
4. **Included phase**, 103 studies identified through database searches and 47 studies identified through other methods were included in the qualitative synthesis, resulting in a total of 150 included studies.

The process was illustrated in the Fig.2 and was designed with a PRISMA Flow Diagram developed by [34].

3.4. Search terms

The search terms were gathered according to the following keywords: microservices, microservice, microservice architecture, multi criteria, multi-criteria, multi objective, multi-objective, decomposition, extraction, sharing, duplication, metrics, quality. These were used in a search string, which is a set of keywords combined with Boolean operators that will execute the search. The search strings used to perform those searches are presented in Table 2.

Since the exclusion criteria define that only studies in English are accepted, the terms used were written in English only. To provide the best result possible, two logical tests were applied to the string. In addition, metadata specification could have been applied. The filters applied are also presented in Table 2.

3.5. Study selection criteria

To guide the selection of bibliographic sources retrieved through the search process, a set of inclusion and exclusion criteria was defined. It is important to note that restrictions on the publication period were imposed (>2021), but during the snowballing process some earlier studies were also included. Table 1 presents the inclusion and exclusion criteria adopted in this study.

Table 1: Inclusion and exclusion criteria

Inclusion criteria (IC)		
ID	**Criteria description**	
IC01	Study analyzes the design tools.	
IC02	Study analyzes Microservices decomposition.	
IC03	Study analyzes decomposition metrics.	
IC04	Decomposition algorithms as MOO.	
IC05	Decomposition algorithms as ML.	
Exclusion criteria (EC)		
ID	**Criteria description**	**%**
EC01	Duplicated study.	44
EC02	Study not available to download.	3
EC03	Not related to Microservices/Decomposition.	27
EC04	Study not written in English.	0

3.6. Studies selection procedure

The study selection process was conducted by one researcher. Initially, the titles and abstracts of all retrieved studies were screened to assess their relevance to the scope of this review. Based on this screening, studies were classified using the Mendeley reference management tool as accepted, rejected, or duplicated. Of the 609 retrieved records, 103 studies (16.9%) were accepted, 166 studies (27.2%) were rejected, 21 were not retrieved (3.4%) and 265 studies (43.5%) were identified as duplicates.

Subsequently, the full texts of the accepted studies were reviewed to determine their relevance and to identify answers to the research questions defined in the review protocol. After the full-text analysis, 54 (9%) were excluded due to not satisfying the inclusion criteria (not describing decomposition process). The overall study selection process is illustrated in Figure 2.

During the title and abstract screening phase, studies were classified as accepted or rejected based on the inclusion and exclusion criteria defined in Table 1, prior to proceeding to the full-text assessment phase.

To address the research questions, studies identified through backward snowballing were incorporated into the review process. Potentially relevant references were added to the data extraction form. After removing duplicates, 57 studies were selected for further assessment. Of these, every report could be retrieved. The remaining studies underwent full-text eligibility assessment, during which 10 studies were excluded due to lack of relevance or not addressing the research questions. Consequently, 47 additional studies were included in the final qualitative synthesis.

4. Results

This section presents the results of the systematic literature review, structured according to the defined research questions. The analysis focuses on the tools and approaches used to support the decomposition of systems into microservices, including the criteria applied during the decomposition process. In addition, this section examines the use of Multi-Objective Optimization and Machine Learning algorithms in microservices decomposition, highlighting current research trends and commonly applied techniques.

4.1. Design tools used for decomposition (RQ1)

To comprehend the system, architects often model its static structure and dynamic interactions, including internal components and external dependencies. In [35] the author reports that, primarily, notations such as Business Process Model and Notation (BPMN), Unified Modeling Language (UML), and Domain-Specific Languages (DSLs) have been used. However, they often require considerable effort to engage stakeholders outside the development team. Also in [36], the authors confirm that the mentioned methods are becoming increasingly less popular. Intuitive graphical methods, commonly referred to as “boxes and lines,” are reported to be approximately 10% more popular among practitioners for microservices design than UML, with 51% favoring “boxes and lines” compared to 41.5% using UML. The DSLs approach is reported to be less than half as prevalent as graphical approaches, accounting for 27% of their usage.

To address these challenges and to foster the active involvement of business stakeholders in software architecture design, new notations were developed to represent and describe software workflows, such as Event Storming designed by Alberto Brandolini [37] and User Story Mapping designed by Jeff Patton [38]. The workshop technique Event Storming is using stick notes to describe workflows in the system [37]. Event Storming can be conducted in three forms as described in [39]:

1. Big Picture Event Storming – encompasses the entirety of business line or domain,
2. Process Modelling – maps out current (as-is) and future (to-be) business journeys and processes,
3. Software Design Event Storming – maps the business processes to specific business scenarios.

Table 2: Search Protocol and Results (Date: 12.04.2026)

ID	DB	Query	Keywords / Filters	Number of results
RQ1	IEEE	(microservice* OR "microservice architecture") AND (decomposition OR extraction) AND design AND tools	Keywords: microservice*; Year: >2021	9
	Springer	(microservices OR microservice OR "microservice architecture") AND (decomposition OR extraction) AND design AND tools	Title: decomposition OR extraction; Year: >2021	10
	Elsevier	(Same as above)	Keywords: microservice; Year: >2021	31
	Wiley	(Same as above)	Keywords: microservice; Year: >2021	59
RQ2	IEEE	(microservice* OR "microservice architecture") AND (decomposition OR extraction)	Keywords: microservice* AND decomposition; Year: >2021	41
	Springer	(microservices OR microservice OR "microservice architecture") AND (decomposition OR extraction)	Title: microservices OR microservice OR "microservice architecture"; Year: >2021	56
	Elsevier	(Same as above)	Keywords: microservice; Year: >2021	43
	Wiley	(Same as above)	Keywords: microservice; Year: >2021	64
RQ3	IEEE	(microservice* OR "microservice architecture") AND (decomposition OR extraction) AND algorithms AND criteria	Year: >2021	5
	Springer	(microservices OR microservice OR "microservice architecture") AND (decomposition OR extraction) AND algorithms AND criteria	Title: microservices OR microservice OR "microservice architecture"; Year: >2021	58
	Elsevier	(Same as above)	Keywords: microservice; Year: >2021	20
	Wiley	(Same as above)	Keywords: microservice; Year: >2021	58
RQ4	IEEE	(microservice* OR "microservice architecture") AND (decomposition OR extraction) AND (MOO strings...)	Year: >2021	15
	Springer	(microservices OR microservice OR "microservice architecture") AND (decomposition OR extraction) AND (MOO strings...)	Title: microservices OR microservice OR "microservice architecture"; Year: >2021	26
	Elsevier	(Same as above)	Keywords: microservice; Year: >2021	11
	Wiley	(Same as above)	Keywords: microservice*; Year: >2021	6
RQ5	IEEE	(microservice* OR "microservice architecture") AND (decomposition OR extraction) AND (ML strings...)	Year: >2021	31
	Springer	(microservices OR microservice OR "microservice architecture") AND (decomposition OR extraction) AND (ML strings...)	Title: microservices OR microservice OR "microservice architecture"; Year: >2021	38
	Elsevier	(Same as above)	Keywords: (microservice OR microservices) AND (decomposition OR extraction); Year: >2021	19
	Wiley	(Same as above)	Keywords: microservice*; Year: >2021	9

Event Storming is based on a set of simple rules and is often described as a tool for modeling systems developed using DDD as mentioned in [8, 11, 40–42]. On the other hand, User Story Mapping (USM) workshops were developed to address the initial population of the product backlog and the systematic creation of user stories within Agile methodologies. During these workshops, participants (including stakeholders and developers) engage in facilitated, collaborative activities to elicit requirements. As mentioned in [43], this approach facilitates shared understanding among participants while enabling the iterative refinement and prioritization of development backlogs in accordance with Agile principles. The main principles and description of the whole process can be found in [44] and [45].

User Story Mapping and Event Storming share a common foundation: both utilize sticky-note diagrams and involve cross-functional teams of versatile specialists to develop an initial plan for the application, but results are different. However, the outcomes of the two approaches differ: in User Story Mapping, the primary goal is to create the product backlog, whereas Event Storming workshops are primarily focused on domain discovery, decomposition, and identifying bounded contexts.

4.2. Approaches to dividing (decomposing) the system into microservices (RQ2)

According to [46] and [47], in the context of Domain-Driven Design, the creation of microservices and the definition of their boundaries (bounded contexts) require careful consideration of trade-offs. As described in [8], teams must balance competing objectives, such as maximizing service cohesion, minimizing inter-service dependencies, and aligning with business capabilities, while decomposing the domain into manageable, independently deployable components. These objectives may pertain to quality attributes such as maximizing team autonomy, supporting technological heterogeneity, and enhancing application scalability. There are also cost-related objectives of decomposition, which stem from hardware limitations and constraints, such as network bandwidth. However, as discussed in [48], the high number of inter-service connections and the embedding of business logic within the communication layer increase coupling between services. As noted in [10, 49, 50], the success of the process largely depends on the proper definition of service boundaries and granularity. As described in [51–54], poor design decisions may lead to raising of technical debt and subsequently to abandoning of Microserices in favor of monolithic architecture.

In [32], the objectives of microservices decomposition were categorized into four main groups: internal quality, external quality, organizational, and external evaluation. Internal quality refers to architectural attributes that can be measured during the development stage. It encompasses objectives such as coupling (e.g., [55]) and cohesion (e.g., [56]). Database quality is also considered. For instance, the number of distributed transactions is used in [57], and data dependencies are examined in [58]. Other internal quality objectives include evolvability used in [59], microservice quality used in [60], redundancy used in [16], and service sizing used in [61].

External quality refers to architectural attributes that can be measured at the deployment stage or during runtime. It encompasses aspects such as availability, which quantifies the amount of time a system remains operational without errors used in [62], and performance, which evaluates the system's response to functional requirements using metrics like latency and throughput used in [57], average throughput used in [63], efficiency used in [62], and response time used in [64]. Resource usage is another key aspect, capturing the hardware consumption of a deployed microservices architecture, including CPU and memory usage used in [64–66], as well as average, maximum, and steady load used in [63]. Finally, scalability measures the system's ability to handle increasing load without compromising performance used in [62].

The organizational category encompasses attributes related to the impact of transitioning to a microservices architecture on the organization. This includes cost, which measures the infrastructure expenses associated with deploying the proposed microservices architecture (e.g used in [64, 67, 68]). Team considerations, which assess the resulting development team in terms of size and associated complexity like in [16] and overall complexity, which evaluates the effort required to migrate a given functionality to a microservices-based system used in [69].

External evaluation involves comparing the size and characteristics of the microservices produced by a given decomposition technique to an expert-defined decomposition. This approach provides an objective measure of how closely the automated or proposed design aligns with established best practices or domain expertise.

The study presented in [70] demonstrates that architectural decisions have a significant influence on the behavior of microservices systems. The authors propose an experimental approach based on a model-driven application generation system that enables the creation of multiple architectural variants and their subsequent benchmarking. It was observed that long execution chains significantly degrade system responsiveness. Reducing the number of services may increase throughput, however, it can also adversely affect the scalability of the system.

Asynchronous interfaces, by virtue of buffering and component decoupling, can lead to improved system performance. The authors further emphasize that key architectural decision-making requires the simultaneous consideration of multiple architectural metrics in conjunction with the characteristics of the deployment environment.

Code quality assurance in microservices architectures is a sophisticated process that requires consideration across three distinct layers: the individual service, the deployment environment, and the system as a whole. Of key importance in this context is the ISO/IEC 25010 standard, which provides a comprehensive framework for evaluating software quality by considering attributes such as portability, maintainability, performance efficiency, functionality, reliability, compatibility, and security. [70, 71] As described in [70, 71], applying this framework across all layers of a microservices system enables a systematic and thorough assessment of quality, guiding architectural and operational decisions. Additionally, [71] demonstrates that classic object-oriented metrics, such as Metrics for Object-Oriented Design (MOOD) and the Quality Model for Object-Oriented Design (QMOOD), are not directly applicable to microservices architectures. Instead, as argued in [71], it is necessary to adopt an abstract modeling approach to capture dependencies between services, enabling the analysis of quality at the level of entire microservice networks.

The approaches and quality evaluation criteria for microservices decomposition are presented in Table 3.

4.3. Criteria used in the decomposition process of microservices (RQ3)

According to [72, 73], the objective selection and decomposition of microservices are the principal challenges in the microservices design process. As described in [74], assessing the quality of a microservice decomposition is critical to avoiding "monolithic hell," a condition in which services are so tightly coupled that they cannot function independently. The identified metrics can be categorized into three main groups: communication, implementation, operational and security. Communication metrics such as number of endpoints, synchronous and asynchronous dependencies, length of service execution chain or strategies for communication error handling, enable the evaluation of interaction complexity between services. Implementation metrics, including lines of code, test coverage, the number of external libraries and programming languages used, and complexity assessment, are derived from source code analysis and influence system maintainability and functionality. Operational metrics such as container size, the number of environments, the degree of adoption of Continuous Integration/Continuous Delivery (CI/CD) practices, or server memory usage represent deployment and runtime aspects. As stated in [71], each group of metrics influences different quality attributes.

Several studies focus specifically on identifying microservice candidates in monolithic systems. In [75] authors proposes a decomposition method based on semantic similarity, where operation names extracted from OpenAPI specifications are transformed into vector representations using word embeddings. The resulting clusters are further validated using cohesion metrics such as Lack of Cohesion (LCOM) and complexity indicators like the Number of Operations (NOO). Similarly, [76] employs genetic algorithms to partition classes based on functional interaction strength and non-functional resource usage metrics, including CPU and memory consumption. This approach relies on execution logs and performance data from legacy systems such as JPetStore to balance functional coherence with system efficiency. Another optimization-oriented solution is the So4MoD framework [77], which applies a multi-objective optimization algorithm (NSGA-II) to identify decompositions that simultaneously improve modularity and reduce critical data leakage risks. The framework evaluates decomposition quality using both modularity metrics, such as Structural Modularity Quality (SMQ) and Conceptual Modularity Quality (CMQ), and security-related metrics including Critical Classes Proportion (CCP) and Critical Superclasses Inheritance (CSI).

Performance, scalability, testing, and security represent additional major research directions. [78] investigates the performance implications of migrating from a monolithic PHP application to a Java/Spring Boot microservices architecture while progressively introducing design patterns such as API Gateway, Strangler, and Singleton. The study evaluates resource consumption using metrics including CPU usage, memory utilization, network transactions, and database response times. In [79], scalability concerns are addressed through systematic scalability analysis approaches that examine how microservice granularity affects system adaptability, response time, failure rates, and feature extensibility. Using KAOS goal-obstacle modeling applied to the hypothetical Filmflix architecture, the study provides architects with a structured methodology for scalability-aware decomposition decisions. In [80], theauthors perform security-oriented research which further explores how decomposition granularity influences attack surfaces and risks to confidentiality, integrity, and availability. By modeling the 5G Network Repository Function (NRF), researchers formulate mathematical equations to compare decomposition strategies based on service size and the number of entry points.

As described in [81], security constitutes a crit-

Table 3: Synthesis of Methodological Approaches for Microservices Decomposition and Quality Evaluation Criteria

Category	Key Attribute	Specific Metrics / Focus	References
Internal Quality	Structural Metrics	Coupling, Cohesion	[55, 56]
	Database Quality	Distributed transactions, data dependencies	[57, 58]
	Architectural Health	Evolvability, service sizing, redundancy	[16, 59–61]
External Quality	Reliability	Availability (uptime, error-free operation)	[62]
	Performance	Latency, throughput, efficiency, response time	[57, 62–64]
	Resource Usage	CPU/Memory consumption, load profiles (steady/max)	[63–66]
	Scalability	Load handling capacity	[62]
Organizational	Cost	Infrastructure and deployment expenses	[64, 67, 68]
	Team Dynamics	Team size and associated complexity	[16]
	Complexity	Migration effort and overall system complexity	[69]
External Eval.	Expert Comparison	Alignment with best practices and domain expertise	[32]
Architectural Impact	DDD Trade-offs	Bounded contexts, service cohesion, dependencies	[8, 46, 47]
	Behavior & Design	Execution chains, async interfaces, granularity	[10, 49, 50, 70]
	Standards	ISO/IEC 25010 (Portability, Security, etc.)	[70, 71]

ical and complex aspect of MSA, stemming from its distributed nature, technological heterogeneity, and continuous evolution. Consequently, a defense-in-depth approach is essential, with authentication, authorization, and secure communication serving as fundamental pillars of microservice-based systems. In [82], 28 security practices for MSA were identified and grouped into six categories. Regarding authentication and authorization, suggested approaches comprise centralized authentication via the API Gateway, two-factor authentication, and adoption of OpenID Connect or Security Assertion Markup Language (SAML). In [82], the authors emphasize the critical role of ensuring that each microservice is responsible for its own security mechanisms, as well as the importance of employing Public Key Infrastructure (PKI) systems. Token management best practices involve keeping tokens free of confidential data, encrypting them, managing roles, and verifying tokens through the API Gateway. Moreover, the API Gateway must distribute system load across multiple nodes to avoid over- or underutilization. [83] reports that the misconfiguration of the load-balancing algorithm can cause bottlenecks and degrade performance.

Further recommendations address secure inter-microservice communication, with emphasis on monitoring, minimizing network traffic, isolating non-public services, and enforcing strong connection authorization. For internal communication, approaches such as OAuth 2.0 in Client Credentials mode or gRPC are recommended. The study further examines private microservice and database security, emphasizing the importance of restricting service visibility, implementing robust security measures in production, and safeguarding databases from unauthorized access.

[84] proposes a set of metrics for automated evaluation of Microservices Architectures, categorized into: secure communication, identity management, and observability. Communication metrics focus on aspects such as the proportion of encrypted connections throughout the system. Identity management metrics assess the authorization mechanisms employed, including: API keys, public data, and security protocols, in both back-end interactions and UI-service communication. Observability refers to the monitoring of services and associated facade components through dedicated mechanisms. Metrics are given as values between 0 and 1, providing a straightforward and comparable evaluation of security practices.

Avoiding anti-patterns is a crucial aspect of designing and developing microservices. As outlined in [85], the identification of anti-patterns requires a structured and systematic approach consisting of three phases: information gathering, intermediate representation, and strategy detection. During the information-gathering phase, source code or binary files are analyzed alongside environmental data and system logs. If available, architectural models and definitions, including UML diagrams, metamodels, and ontologies, can also be analyzed. Collected data are subsequently transformed into intermediate representations, including dependency graphs, project structure matrices, or metamodels, facilitating abstract analysis. The last phase, detection strategy, may comprise automatic technical reasoning or expert analysis aided by model visualizations across multiple levels, including domain, technological, service, and operational perspectives.

Quality assessment and validation are also central themes in current microservices research. In [86], the author proposes a refined set of quality assessment criteria based on both academic literature and industrial feedback, including granularity, cohesion, coupling, response time, health management, and reusability. The study synthesizes findings from 29 scientific papers and an industry interview to define a minimum set of quality indicators for migrated microservices. Complementing this work, the MicroValid framework [87] introduces an open-source validation environment for evaluating decomposition results generated by automated tools. Using coefficient of variation (Cv) analysis, the framework identifies outliers and imbalances in granularity, coupling, and cohesion across decomposed services. Inputs include JSON-based system models and remote repositories, enabling extensible and unbiased validation of microservice "cuts." In addition, research evaluating the modularity of Domain-Driven Design (DDD) quantitatively measures the impact of DDD on maintainability and modularity using metrics such as LCOM, Service Granularity Metric (SGM), Relational Cohesion (H), and Instability (I) in a real-world academic information system implemented as a modular monolith.

[85] presents a catalog of 58 distinct Microservices Architecture anti-patterns, organized into five categories: Project Within Service, Decomposition Between Services, Services Interaction, Security and Team Organization. The first category, Project Within Service, addresses common design errors in single services, including overly granular (nanoservices) or excessively complex (megaservices) components, unused services, interfaces primarily supporting CRUD operations, and operations exhibiting low cohesion. The Decomposition Between Services category addresses common mistakes in microservice decomposition, including overlapping responsibilities, redundant service creation, circular dependencies, and reliance on shared databases. The Services Interaction category encompasses common communication issues, including missing API Gateways, static endpoints, and insufficient error handling. The Security category addresses common security problems, including absent authorization, over-privileged accounts, unencrypted data, and the use of proprietary cryptographic algorithms. The Team Organization category addresses issues in team and DevOps practices, including lack of CI/CD, manual configurations, high technological heterogeneity, inadequate support mechanisms, missing documentation, and insufficient monitoring or message tracking. [88] also highlights the importance of the API Gateway pattern in mitigating the increased attack surface by providing a single entry point for service communication, thereby strengthening security at the communication layer, which is particularly susceptible to man-in-the-middle (MITM) attacks. Alternatively, JSON Web Tokens (JWT) can be used for authorization across microservices. However, token propagation may introduce security issues such as the confused deputy problem. In addition to emphasizing the importance of the API Gateway, [89, 90] argues that each microservice should be granted read–write access only to its own data and recommends the use of hybrid interprocess communication (e.g., asynchronous RPC), as fully synchronous communication patterns can incur significant computational overhead and risk runtime infinite loops.

Reducing coupling between services is a key aspect of microservices systems. [91] proposes the Microservice Coupling Index (MCI), which enables the assessment of coupling by considering not only observed dependencies but also potential ones. The Microservice Coupling Index provides a more accurate assessment of service coupling, as higher values correspond to increased difficulties in error localization and more complex change implementation. It further indicates reduced independence in microservice development. Analyses based on Spearman's correlation and Principal Component Analysis (PCA) demonstrate that the MCI conveys more information than conventional coupling metrics and allows for more precise evaluation of alternative project decisions. [73] presents an extensible tool to support the decomposition of monolithic systems into microservices, proposing a standardized pipeline for data collection, decomposition generation, quality evaluation, and visualization to address the lack of comparative environments for existing approaches. [92] reports that MSAs often face reuse challenges due to code duplication, technology heterogeneity, and unclear service boundaries. Context-specific dependencies and complex inter-service interfaces further limit portability, while managing multiple service versions can introduce

compatibility issues that hinder consistent reuse across an organization. To address these challenges, the authors propose the Reusable Microservices Framework, which aims to enhance microservice reuse by providing structured support for identifying, sharing, and managing reusable service components.

Methods for microservice decomposition may be categorized into four main approaches: static and dynamic code analysis, analysis of business models and data, application of artificial intelligence, and hybrid strategies. In code analysis approaches, static methods are focused on application's structure such as presented in [10, 93]. This analysis investigates class dependencies, semantic similarities (such as identifiers, comments, and variable names), and employs Abstract Syntax Trees (ASTs) alongside connection diagrams to model interactions between components as described in [14, 94, 95]. Moreover, in [69], the authors propose a code vectorization approach (Code2Vec) to facilitate the representation of code fragments in vector form, supporting the identification of semantic similarities. Dynamic code analysis, in contrast, monitors the application during execution, providing a more precise view of real data flows and system dependencies as presented in [49, 96]. Mono2Micro, as described in [97], integrates execution trace analysis with use case information to facilitate microservice decomposition within concrete business contexts. In [98], the authors leverage state-of-the-art natural language processing and dynamic tracing techniques to identify microservices as groups of correlated API operations. The final decomposition is achieved by refactoring the source code using execution traces from dynamic analysis and dependencies from static analysis, enabling the handling of both dynamic and static language features.

The second category of approaches is based on business models and data analysis. DDD serves as the leading methodology, concentrating on identifying cohesive bounded contexts that correspond to candidate microservices ([94, 99]). Alternative approaches leverage Business Process Models (BPMs), use cases, and Data Flow Diagrams (DFDs) to determine business functions and guide their decomposition ([56, 95, 100]). As described in [101], relational databases can also provide valuable information, as analyzing SQL queries and data access patterns facilitates the refactoring of operations into well-defined, cohesive service units.

The third category of decomposition approaches employs machine learning and artificial intelligence. A variety of Machine Learning and optimization techniques have been applied to support microservices decomposition. Common approaches include clustering algorithms [14, 46, 102–104], Graph Neural Networks (GNNs) [2, 105, 106], evolutionary algorithms [14, 55, 107–110], and Fuzzy Cognitive Maps (FCMs) [110, 111], which are used to identify service boundaries, optimize cohesion and coupling, and analyze structural or semantic dependencies within systems. These approaches and their applications are described in greater detail in Sections 4.4 and 4.5.

The final category includes hybrid approaches and supporting tools. Combining multiple data sources, for example source code and business process models, is increasingly adopted to enhance decomposition accuracy [94]. A number of supporting tools are also developed to support the process of monolith decomposition such as Service Cutter [112], Mono2Micro [97], RapidMS [113] and MicroRefact [114]. Certain tools and methods offer not only decomposition analysis and recommendations but also automate code refactoring and facilitate service integration, for example by generating REST interfaces in place of direct dependencies [114]. Although many of these tools are not universally applicable and work only in particular technological contexts, they constitute a significant advancement in automating and systematizing the microservice decomposition process.

Research on microservice decomposition and evaluation has expanded significantly, covering a wide range of architectural, quality, security, and performance-related concerns. One notable approach is the Microservice Dependency Matrix, which focuses on identifying dependencies between services through endpoint communication patterns and shared data entities corresponding to bounded contexts. Using the source code of Java-based Spring Boot microservices as input, the approach automates the extraction of dependency information directly from the codebase, enabling development teams to monitor architectural evolution, reduce coupling, and detect functional bottlenecks. In contrast, comparative studies of meta-data-based extraction tools evaluate the practical applicability of automated decomposition tools such as Service Cutter, Decomposer, and Microservice-extractor. These studies compare automated results with decompositions performed manually by domain experts using data from banking systems, open-source repositories, and system specifications, highlighting challenges such as over-decomposition and limited industrial maturity of current tools.

The criteria and techniques used in microservices decomposition are presented in Table 4.

4.4. Multi-Objective Optimization (MOO) algorithms used for decomposition of microservices (RQ4)

Existing literature widely investigates approaches for decomposing monolithic systems into microservices, aiming to enhance quality attributes including scalability, maintainability, and elasticity ([107–109]).

Table 4: Criteria, Techniques, and Approaches Used in Microservices Decomposition

Category	Sub-Category / Focus	Key Metrics, Techniques, or Practices	References
Quality Metrics	Communication, Implementation, Operational, and Security	Endpoints, dependencies, LOC, test coverage, container size, CI/CD adoption, memory usage, encrypted connections, observability.	[71–74, 84]
Security Practices	Defense-in-depth and Communication	API Gateway (centralized auth), JWT, PKI, 2FA, OAuth 2.0, gRPC, token encryption, and load-balancing algorithms.	[81–83, 88]
Anti-patterns	Identification and Classification	Detection phases (gathering, representation, detection). 58 patterns across: Project within service, Decomposition, Interaction, Security, and Team.	[85, 88–90]
Coupling & Reuse	Measurement and Frameworks	Microservice Coupling Index (MCI), Reusable Microservices Framework, and extensible decomposition tools.	[73, 91, 92]
Decomposition: Code Analysis	Static and Dynamic Analysis	ASTs, connection diagrams, Code2Vec, execution trace analysis, use case integration (e.g., Mono2Micro).	[10, 14, 69, 93, 94, 97, 98]
Decomposition: Business/Data	Domain and Model-driven	DDD, Bounded Contexts, BPM, DFDs, and SQL query/data access pattern analysis.	[56, 94, 95, 99–101]
Decomposition: ML & AI	Automated boundary discovery	Clustering (k-means, DBSCAN, Agglomerative), Graph Neural Networks (GNNs), Evolutionary Algorithms (NSGA-II), and Fuzzy Cognitive Maps.	[2, 46, 55, 102, 103, 106, 111]
Hybrid & Tools	Integration and Automation	Combining code and business models. Support tools: Service Cutter, Mono2Micro, RapidMS, MicroRefact.	[94, 97, 112–114]

The problem is frequently modeled as a multi-objective optimization task that seeks to minimize coupling and maximize cohesion, while also accounting for service granularity and additional architectural quality attributes ([55, 111, 115]). [72] reveals that industry practitioners typically consider at least four objectives simultaneously when making extraction decisions. In this context, evolutionary algorithms play a crucial role.

Evolutionary Algorithms (EAs) are especially effective for multi-objective optimization, since their population-based nature allows multiple Pareto-optimal solutions to be identified concurrently [116]. The fundamental concepts involve the Pareto-domination principle, convergence to the Pareto front, and maintenance of solution diversity along the front. Vector Evaluated Genetic Algorithm (VEGA) laid the foundation for multi-objective genetic algorithms, but this approach is currently regarded as less suitable for handling complex, high-dimensional problem spaces [117]. NSGA-II and NSGA-III improve upon earlier methods by introducing elitism and advanced selection techniques like crowded-comparison and reference-point association ([116, 118]). NSGA-II enhances efficiency by employing fast non-dominated sorting and removing the requirement for sharing parameter settings, which has contributed to its widespread adoption in multi-objective optimization. In contrast, NSGA-III is well suited for many-objective optimization problems, owing to its adaptive normalization and reference-point-based niching mechanism [118].

Additional notable algorithms are Strength Pareto Evolution Algorithm (SPEA) and SPEA2, which employ an external archive and a strength-based metric to assess the quality of individuals [119]. Furthermore, SPEA2 incorporates refined density estimation and an archive truncation strategy, enhancing its ability to manage boundary solutions effectively.

Multiobjective Evolutionary Algorithm Based on Decomposition (MOEA/D) is an alternative method that divides a multi-objective optimization problem into multiple single-objective subproblems via weight vectors [120]. In the context of SaaS systems, [121] emphasizes variability management through the SAMIM method, leveraging MOEA/D to balance granularity, commonality, and business alignment. However, it can struggle when Pareto fronts are non-uniformly distributed [118]. Effective evolutionary algorithms share a common goal of balancing convergence and diversity, using environmental selection, fitness assessment, and methods to preserve solution distribution across the objective space.

The MOPSO (Multi-Objective Particle Swarm

Optimization) algorithm extends the classical Particle Swarm Optimization (PSO) to multi-objective problems by employing Pareto dominance and a global repository that stores non-dominated solutions and guides the movement of particles in the solution space [122]. The diversity of solutions is maintained through a hypercube-based structure that segments the objective function space. [122] shows that MOPSO achieves competitive results compared to other Multi-Objective Evolutionary Algorithms (MOEAs) such as NSGA-II and Pareto Archived Evolution Strategy (PAES), often with lower computational costs.

In contrast, IBEA bases its selection process on quality indicators, such as IBEAε+ and IBEA hypervolume (IHD), which allow decision-maker preferences to be flexibly incorporated without the need for additional diversity-preservation mechanisms [123]. Each solution is assigned a fitness value based on its comparisons with the other members of the population, enabling effective selection. In [123], IBEA has demonstrated superior performance compared to NSGA-II and SPEA2 in numerous benchmarks, offering greater generality and improved population scalability.

An evolutionary algorithm that is particularly popular among the proposed solutions to the decomposition problem is NSGA-II, which generates diverse Pareto-optimal microservice decompositions. NSGA-II has also been applied in cloud migration scenarios, taking into account constraints related to network traffic and hosting costs and were used in [14, 109, 110]. Expanding on scalability challenges, [124] introduce MSExplorer, which incorporates a preliminary clustering step to reduce problem complexity before applying NSGA-II optimization. Beyond design, [125] address deployment concerns with Yonga, an adaptive placement strategy optimized for resource-constrained environments, while [126] focus on post-design analysis by extracting feature models to support reuse and configurability in microservices-based product lines.

The newer version of this algorithm, NSGA-III, has been applied to more complex cases of software domain decomposition, supporting a larger number of optimization objectives and their implementations were described in [108, 109]. [127] also frame microservice identification as a many-objective problem, demonstrating that simultaneously optimizing criteria such as coupling, cohesion, feature modularization, reuse, and network overhead yields superior architectural solutions compared to simpler formulations. An effective alternative is the Indicator-Based Evolutionary Algorithm (IBEA), employed, among others, in the MSExtractor approach, where it outperformed both NSGA-II and SPEA2 by achieving more balanced trade-offs between coupling, cohesion, and granularity [55]. Beyond purely evolutionary techniques, the effectiveness of classical Genetic Algorithms (GA) can be further enhanced through integration with Fuzzy Cognitive Maps (FCM), which enable the modeling of uncertainty and complex interdependencies [94, 111]. Along similar lines, [128] combine combinatorial optimization with software synthesis to not only derive optimal service boundaries but also automatically generate consistent microservice implementations. More recently, deep reinforcement learning (DRL) approaches have gained traction, particularly actor–critic methods, which learn reward functions to autonomously plan optimal migration paths from monolithic systems to microservice architectures [110].

The decision to migrate to a MSA is complex and requires support in the form of decision-making frameworks and appropriate tools. Frameworks such as those proposed by [107] enable decision-making based on objective data, while Multi-Layer Fuzzy Cognitive Map (MLFCM) models additionally provide the capability for scenario analysis and enhance the transparency of the process [111]. Techniques for algorithm hyperparameter tuning are also critical for decomposition quality, as they can substantially enhance the outcomes [129]. Moreover, the use of weighted loss functions allows for the direct incorporation of business objectives into the optimization process, making it more precise and tailored to the organizational context [130]. The process is iterative, requiring ongoing monitoring and adjustment of the architecture to meet changing requirements and technical conditions [131, 132].

The multi-objective optimization algorithms used in different works are presented in Table 5.

Over the years, various aspects of MSA have been extensively analyzed, with researchers investigating its different components, implications, and effects from multiple perspectives, contributing to a deeper understanding of its significance and evolution. In [133], the focus is on the migration process, including the activities such as forward and reverse engineering and architectural transformation, as well as the challenges faced during the transition to a Microservices Architecture. In [134], the authors examined the advantages and disadvantages of industrial microservice practices using semi-structured interviews. In [135], interviews with nine participants were conducted to explore trade-offs in microservice quality metrics. In [136], the authors carried out a mixed-methods study, combining surveys and interviews, to collect and categorize best practices and challenges, as well as solutions identified by practitioners with successful MSA implementations. In [36], a mixed-methods study was conducted to gain a deeper understanding of how microservices are designed (e.g., decomposition processes, modeling methods), monitored (metrics, practices, tools), and tested within industrial

systems. In [82], the authors proposed security practices derived from issues observed in GitHub repositories, Stack Overflow, and Wiki pages. To assess the validity of these results, a survey of 74 participants was carried out, resulting in a compiled set of practices that effectively mitigate security challenges in MSA.

Table 5: Multi-objective optimization algorithms used in research paper with (B)enchmarks

Research paper	NSGA-II	NSGA-III	SPEA2	IBEA	MOEA/D
[13]		✓			
[55]	✓(B)		✓(B)	✓	
[59]	✓				
[108]	✓				
[109]	✓				
[115]	✓		✓		
[77]	✓				
[137]	✓				
[127]	✓(B)	✓			
[124]	✓				
[121]					✓
[125]	✓				
[126]	✓		✓		

4.5. Machine Learning algorithms used for decomposition of Microservices (RQ5)

The transition from monolithic to microservices architectures has progressed from manual, heuristic-based partitioning toward advanced automated approaches that employ Machine Learning (ML), Graph Neural Networks (GNNs), and Reinforcement Learning (RL), while integrating structural, dynamic, and semantic analyses [2, 101, 138, 139].

The literature classifies monolith-to-microservice decomposition approaches into several major methodological categories, reflecting the increasing maturity and diversification of the field. A substantial body of work focuses on static and structural analysis, where source code artefacts, such as classes, methods, and attributes, are examined to construct dependency graphs based on relationships including method invocations, inheritance hierarchies, and aggregations. These representations are subsequently used to identify cohesive clusters that can serve as candidate microservices, as exemplified by tools such as MicroMiner [140] and Service Cutter [141].

Complementary to static approaches, dynamic and log-based analysis techniques incorporate runtime information to capture actual system behavior [104, 138, 140]. By analyzing execution traces and access logs, these methods provide insights into functional interactions that may not be evident from static structures alone. For instance, approaches such as MINDS ([138]) integrate inductive process mining with natural language processing to derive behavioral models from event logs, thereby enabling more accurate and context-aware decomposition.

Semantic and domain-driven approaches further extend decomposition techniques by aligning technical components with business semantics. These methods employ natural language processing techniques, including Word2Vec and BERT [106], as well as Large Language Models [101, 142, 143], to extract domain knowledge from source code identifiers and documentation. MonoEmbed presented in [144] represents a notable advancement in this area by applying contrastive learning to tailor language model embeddings specifically for decomposition tasks.

More recently, graph-based learning methods have gained prominence. These approaches model software systems as graphs and apply Graph Neural Networks [102, 143, 145, 146], such as Graph Convolutional Networks [147], Graph Attention Networks [148], and Gated Graph Neural Networks [149], to capture complex structural and semantic dependencies. In parallel, reinforcement learning-based approaches conceptualize decomposition as an optimization problem [110, 141]. Frameworks such as RLDec [139] and MARS [148] utilize deep reinforcement learning agents to explore the solution space and identify decompositions that optimize architectural quality metrics.

In addition to methodological advancements, several specialized tools have been proposed to address specific challenges in microservices modernization. Atlas, presented in [110], functions as a hybrid cloud migration advisor that employs data-driven techniques to optimize API latency and cloud deployment costs. S-Quality, described in [103], focuses on evaluating candidate microservices with respect to maintainability-oriented architectural objectives. ALGE-MLFCM [111] introduces a decision-support model based on genetically evolved fuzzy cognitive maps, assisting organizations in determining the suitability of adopting microservices. MonoMorph, introduced in [142], extends beyond decomposition by leveraging Large Language Models to automate the refactoring process, generating deployable microservice implementations directly from decomposition outputs.

The machine learning algorithms used in the studies are summarized in Table 6.

The evaluation of decomposition approaches is predominantly conducted using a set of widely adopted

Table 6: ML Algorithms used in Microservice decomposition

Ref.	Primary Algorithm(s)	Key Focus
[140]	Shifted Weighted K-means (SWK), Elbow Method	Web application URI space partitioning
[105]	Personal PageRank, Egocentric Networks	Business functionality inference from seeds
[101]	Louvain, Leiden, GPT API (Semantic Analysis)	DB schema and source code alignment
[145]	Multi-head Attention GNN (MEGNN), K-means++	Multi-view feature fusion for Java projects
[110]	Actor-Critic DRL, Genetic Algorithm	Hybrid cloud migration and API latency
[111]	Multi-layer Fuzzy Cognitive Maps (MLFCM)	Decision support for migration strategy
[141]	Deep Reinforcement Learning, MoDisco	Model-driven reverse engineering and mapping
[46]	Graph Centrality (Degree, PageRank)	DDD principles and graph centrality analysis
[150]	Louvain, Fix-and-Relax (ILP Optimization)	Method-level extraction and feasibility
[143]	Zero-shot LLM Prompting, GNN-driven Grouping	Domain-aligned identification using LLMs
[103]	K-Means, Mean-Shift, Leiden, S-Quality	AI-based maintainability quality objectives
[138]	Inductive Miner (IM), BERT Embeddings	NLP-driven process mining on event logs
[146]	Eigenvector Centrality, GNN	SSO module migration and network formation
[2]	Dual View Fusion (GDC-DVF), FCM Network	Fusion of structural and business views
[151]	Service Graph (SG) and Task Graph (TG)	Automated extraction from SOA monoliths
[14]	Hierarchical DBSCAN	Noise-robust extraction via density clustering
[142]	LLM-powered Automated Refactoring	Code-level refactoring and service generation
[139]	Deep Reinforcement Learning (RLDec)	Optimizing extraction via architectural rewards
[144]	Contrastive Learning, LoRA-tuned LLMs	Specialized embeddings for decomposition
[102]	VAE-based GNN, Fuzzy C-Means (FCM)	Static analysis and dimensionality reduction
[152]	Spectral Clustering, Cosine/Jaccard Similarity	API similarity graph analysis expert system
[106]	SVM, CodeBERT, Louvain (MicroMiner)	Type-based service layer identification
[149]	Gated GNN (GGNN), Semantic Attention	Self-supervised graph reconstruction
[147]	Graph Convolutional Networks (GCN)	Library-based node and package classification
[104]	Weight Fusion Spectral Clustering	Content-insensitive log fusion for legacy systems
[148]	Actor-Critic RL (PPO), GAT (MARS)	Goal-oriented iterative architectural refactoring

open-source Java applications, including JPetStore, DayTrader, AcmeAir, PetClinic, and PlantsByWebSphere [2, 145, 149]. These benchmarks provide a common basis for comparison across studies. Assessment is typically performed using a range of quantitative metrics that capture key architectural properties. Structural Modularity (SM) [101, 149] is used to evaluate the balance between cohesion and coupling, while Non-Extreme Distribution (NED) [14] assesses the balance of cluster sizes. Interface Number (IFN) [106] measures the number of entry points into a service, and Business Context Purity (BCP) [149] evaluates the functional consistency of identified microservices. Together, these metrics support a systematic and comparative evaluation of decomposition quality, although their reliance on standardized benchmarks may limit external validity.

5. Discussion

This study aims to characterize the design of microservices systems by investigating techniques for microservice identification, the factors influencing decomposition into microservices, the tools utilized throughout the design phase, and the application of multi-objective optimization algorithms during this process.

5.1. Main findings and lessons learned

This section summarizes the main findings and lessons learned from the reviewed studies in relation to the defined research questions. The discussion covers design tools, decomposition approaches and criteria, as well as the application of Multi-Objective Optimization and Machine Learning algorithms in microservices decomposition. The identified trends and observations provide insights into current practices, challenges, and emerging directions in the design of microservices-based systems.

5.1.1 RQ1

The migration process is well-described from monolithic architecture in [55, 94, 107–109, 111]. However, in this study, the focus was on the design phase – even before the actual code is written. To achieve this goal, modeling tools(Event Storming and User Story Mapping), decomposition process, metrics used for decomposition and algorithms incorporated into the decomposition process were analyzed.

Event Storming is founded on a concise set of

methodological rules and is frequently characterized as a modeling technique for systems developed in accordance with DDD principles. User Story Mapping workshops were developed to support the initial population of the product backlog and the systematic creation of user stories within Agile methodologies. During the research, no evidence of the incorporation of these techniques into research tools could be identified, despite their widespread adoption by practitioners, as described in [43].

While Event Storming and User Story Mapping share a common methodological basis, namely, the use of visual artefacts such as sticky-note diagrams and the active participation of cross-functional, multidisciplinary stakeholders, their objectives and outcomes differ. User Story Mapping is primarily concerned with structuring and prioritizing requirements in the form of a product backlog. In contrast, Event Storming workshops are principally oriented toward domain exploration and analysis, including domain decomposition and the identification of bounded contexts.

5.1.2 RQ2

The findings indicate that microservices design is fundamentally a multi-objective trade-off problem, where defining appropriate service boundaries [92] and granularity [55] is critical to system success. Effective decomposition requires balancing cohesion, coupling, scalability, organizational factors, and infrastructure constraints. Poor design decisions, however, can increase communication complexity, introduce technical debt, and even lead to a return to monolithic architectures [51–54]. The literature further shows that decomposition objectives span internal and external quality attributes, organizational considerations, and alignment with expert-defined designs. Moreover, architectural decisions, such as service granularity, interaction patterns, and execution flow, have a direct and measurable impact on runtime performance and system behavior [30]. Importantly, traditional object-oriented metrics are insufficient for evaluating microservices, necessitating system-level, dependency-aware approaches supported by adapted quality frameworks [71]. Overall, successful microservices design demands holistic, context-aware evaluation that integrates technical, organizational, and runtime perspectives.

5.1.3 RQ3

The process of selecting the proper set of metrics is the most crucial part of getting the desired solution. The most common metrics mentioned in the literature are: coupling and cohesion as in [7, 10, 53, 77, 91, 98, 137], security levels as in [77, 88], network traffic as in [88, 137] and microservices quality such as silhouette coefficient in [6, 93] or complexity analysis like SQALE index mentioned in [54]. Security level is based on the defense in depth concept and its pillars: authentication, authorization, and secure communication. Proposed approaches to authentication and authorization include centralized authentication through an API gateway, two-factor authentication, and the adoption of OpenID Connect or Security Assertion Markup Language (SAML). Secure communication involves usage of public keys and tokens with OAuth 2.0 (Client Credentials).

The quality of Microservices Architectures can be evaluated through the identification and analysis of architectural anti-patterns. Common design deficiencies include excessively fine-grained services (nanoservices) as well as overly complex and oversized services (megaservices), low cohesion, overlapping service responsibilities, redundant service creation, circular dependencies, and the use of shared databases across multiple services [85]. Additional challenges arise from insufficient monitoring and message tracing mechanisms, as well as inadequate error-handling strategies, all of which may negatively affect system observability, maintainability, and reliability [72, 73].

Another significant concern relates to the configuration and usage of API gateways. Both the absence and the excessive reliance on API gateways may introduce architectural and security-related issues, particularly with respect to token validation, role and access management, and traffic distribution [83]. Furthermore, improperly configured load-balancing mechanisms can create performance bottlenecks and contribute to the overutilization or underutilization of computational resources. Security vulnerabilities may also emerge when sensitive data remain unencrypted, whether in persistent storage, inter-service communication channels, or authentication and authorization tokens.

5.1.4 RQ4

As presented in [32, 55, 94, 111], the trade-offs in design process are inevitable and the most common approach is using the Multi-objective optimization algorithm with several metrics incorporated into objective function. This approach allows multiple Pareto-optimal solutions to be identified concurrently which improves the accuracy and time to discover the most optimal solution.

NSGA-II is the most commonly used algorithm for formulating microservice decomposition as a multi-objective optimization (MOO) problem [55, 59, 77, 108, 109, 115, 124–127, 137]. An effective alternative is the Indicator-Based Evolutionary Algorithm

(IBEA), which, in approaches such as MSExtractor [55], has been shown to outperform NSGA-II and SPEA2 by achieving more balanced trade-offs between coupling, cohesion, and granularity.

The performance of classical Genetic Algorithms (GAs) can be enhanced through integration with Fuzzy Cognitive Maps (FCMs) to model uncertainty and complex dependencies [2, 102, 111].

5.1.5 RQ5

The literature shows a clear shift from manual, heuristic-based decomposition to advanced AI-driven approaches. These methods combine structural, dynamic, and semantic analyses to improve accuracy. Key techniques include static code analysis [2, 101, 138, 139], runtime behavior mining [104, 138, 140], NLP-based domain modeling [138], Graph Neural Networks [102, 143, 145, 146], and Reinforcement Learning [110, 148]. Specialized tools extend these methods by supporting decision-making, quality evaluation, cost optimization, and automated refactoring into deployable microservices.

More recently, deep reinforcement learning (DRL) approaches, particularly actor–critic methods, have gained attention for learning reward functions that support automated planning of optimal migration paths to microservice architectures [110, 148]. Hyperparameter tuning [129] and the use of weighted loss functions [130] further improve decomposition quality by enabling more effective optimization and the direct incorporation of business objectives.

Despite these advances, evaluation practices remain limited. Most studies rely on a small set of open-source benchmarks and standard metrics such as cohesion, coupling, and service granularity. This indicates that, although methods are increasingly sophisticated, challenges remain. In particular, there is a lack of generalizable, context-aware solutions and limited validation in real-world industrial environments.

5.2. Implications for practitioners

Microservices Architecture (MSA) improves the scalability, maintainability, and deployment flexibility of software systems by decomposing applications into smaller, independently deployable services. One of the primary challenges in adopting MSA concerns the effective decomposition of system components into microservices, which requires balancing multiple quality attributes, including coupling, cohesion, security, scalability, and overall service quality. At the same time, the architectural and operational overhead associated with establishing a microservices ecosystem may reduce the suitability of MSA for smaller-scale projects. From an economic perspective, MSA can contribute to improved cost efficiency through more effective utilization of computational resources, deployment on infrastructure tailored to individual service requirements, and the reduction of both over-provisioning and resource underutilization [83].

The reviewed studies are mostly focused on ML algorithms (like k-means [103, 140, 145] or GNN [102, 143, 145, 146]) or MOO techniques to address the microservice decomposition problem. Clustering-based approaches typically focus on identifying structural or semantic similarities between system components, aiming to maximize cohesion and minimize coupling within service boundaries. In contrast, MOO approaches explicitly optimize several quality attributes simultaneously, enabling a more balanced consideration of architectural trade-offs. Among the identified MOO techniques, NSGA-II emerged as the most widely adopted algorithm [55, 59, 77, 108, 109, 115, 124–127, 137]. However, this algorithm is generally more suitable for optimization problems involving a limited number of objectives, commonly up to three, which may constrain the comprehensiveness of the resulting decomposition solutions. The adoption of many-objective optimization algorithms, such as NSGA-III [13, 127] os MOEA/D [121], could enable the integration of a broader range of architectural and operational criteria, thereby supporting the generation of more context-aware and tailored microservice decomposition solutions.

When implemented effectively, MSA can reduce time-to-market and facilitate continuous integration and continuous deployment (CI/CD) practices by enabling automated testing, shorter release cycles, and more efficient rollback procedures. Furthermore, when combined with Agile development methodologies, MSA can improve organizational agility and responsiveness, providing substantial operational and technical advantages for software development teams.

5.3. Future perspectives

The analysis of the reviewed literature provides important insights into the design and implementation of MSA. Based on the identified findings, several open challenges and research gaps can be recognized that require further investigation in order to support the effective and sustainable adoption of MSA in industrial environments.

A particularly significant challenge concerns the optimization of microservice decomposition and composition. Determining an appropriate set of services for a microservice-based system is critical for ensuring system stability, performance, and operational reliabil-

ity [107–109]. This problem is inherently complex, as modern software systems may consist of hundreds of interdependent services, each characterized by distinct functional responsibilities, dependencies, and performance requirements. Consequently, future research should focus on the development of systematic methods and decision-support strategies for Microservices Architecture design that explicitly account for quality attributes such as scalability, maintainability, reliability, performance and security. Furthermore, these approaches should support the evaluation of trade-offs among multiple competing quality criteria in order to identify context-aware and operationally suitable microservice compositions.

As highlighted in this study, an appropriate set of security practices must be implemented to ensure the protection of microservice-based systems. Commonly adopted measures include centralized authentication through an API Gateway, two-factor authentication, and the use of standards such as OpenID Connect or Security Assertion Markup Language (SAML). Secure communication mechanisms, such as public key cryptography and token-based authentication using OAuth 2.0 (Client Credentials), may, however, introduce performance overhead. Consequently, a promising direction for future research lies in the development of performance metrics capable of quantifying the impact of adopting such security technologies in large-scale microservice systems comprising hundreds of services.

Despite the existence of established techniques such as Event Storming and User Story Mapping, there remains considerable potential for the development of new design tools that further optimize domain understanding, knowledge elicitation, and stakeholder engagement. Future tools could aim to more effectively capture and structure domain knowledge, support continuous refinement as understanding evolves, and better integrate technical and business perspectives throughout the design process. In particular, enhanced mechanisms for engaging business stakeholders may facilitate clearer communication of requirements, assumptions, and constraints, ultimately leading to more accurate system designs and improved alignment between business objectives and technical solutions.

Multi-objective optimization algorithms continue to be actively developed and refined. A promising direction for future research lies in the systematic analysis of these algorithms when applied to microservice decomposition, particularly with respect to their performance characteristics. Such studies could examine scalability, convergence behavior, and solution quality under varying system sizes and constraints, as well as their impact on runtime performance and resource consumption. Integrating decomposition outcomes with comprehensive performance analyses would provide deeper insights into the practical applicability of MOO-based approaches and support the selection of suitable algorithms for real-world microservice systems.

6. Threats to validity

This section presents a critical assessment of the potential threats to the validity of the findings produced by the systematic literature review (SLR), with reference to the categories of validity threats discussed in [153]. The evaluation is structured according to the three-dimensional validity framework, encompassing internal, external, and construct. Particular emphasis is placed on methodological challenges inherent to conducting systematic literature reviews within rapidly evolving and emerging research areas.

6.1. Construct validity

To minimize the likelihood of omitting relevant studies due to incomplete or inadequately formulated search terms, the search strings were subjected to preliminary testing prior to the implementation of the research protocol. This procedure was intended to verify the appropriateness and relevance of the studies retrieved through the automated search process. Moreover, the review incorporated multiple academic databases in order to improve the breadth and coverage of the literature search. The Mendeley reference management software was employed to facilitate the identification and removal of duplicate entries and to support the verification and organization of bibliographic records. Additionally, the automated search procedure was supplemented through the application of a backward snowballing technique, enabling the identification and inclusion of earlier foundational studies that may not have been captured through database-driven searches alone.

6.2. Internal validity

A potential threat to internal validity arises from the fact that the review process, including study selection, screening, data extraction, and synthesis, was conducted by a single researcher. This may increase the risk of subjective bias and inconsistencies in the interpretation and classification of studies. To mitigate this limitation, a predefined review protocol specifying the inclusion and exclusion criteria was established prior to the execution of the review. In addition, all stages of the review process were carefully documented to enhance transparency, traceability, and methodological consistency.

6.3. External validity

To improve the representativeness and comprehensiveness of the findings, the literature search process combined an automated database search with a backward snowballing approach. The initial automated search was restricted to studies published from 2021 onwards in order to focus on recent developments in microservices design research. However, the subsequent backward snowballing procedure enabled the inclusion of earlier foundational studies that were identified as relevant through the references of selected publications. Although the application of a complete snowballing strategy, including forward snowballing, could potentially have revealed additional recent studies, the number and diversity of the retrieved publications were considered sufficiently representative of the current research landscape. Furthermore, only peer-reviewed studies were included in the review. Studies were excluded where a more recent and substantively equivalent version of the work had already been incorporated into the analysis.

7. Conclusion

This study provides a comprehensive analysis of research on the current state-of-the-art in microservices system design. It focuses on the tools employed during the design phase, approaches to microservice decomposition, and the metrics and algorithms used to achieve effective decomposition. The analysis reveals that no single technique is universally effective; all approaches involve inherent trade-offs that must be carefully considered. In particular, MOO algorithms often generate multiple candidate solutions, each requiring expert evaluation. To date, no comprehensive solution exists that can guide and support the entire design process. Furthermore, the approaches identified in this systematic review are generally applicable only under specific technical conditions and constraints. Widely used practitioner tools, such as Event Storming and User Story Mapping, are not yet integrated into existing design tools and frameworks. Consequently, decomposing a system into small, independent modules remains one of the most challenging aspects of microservices design.

Despite these challenges, research continues to develop novel approaches, reflecting that the problem is far from fully solved. Adopting or migrating to a microservices architecture (MSA) offers significant organizational benefits, including improved scalability and maintainability, reduced costs, accelerated time-to-market, and simplified future releases.

Building on these insights, the author plans to develop an approach that integrates the Event Storming technique with a selected set of metrics and a chosen MOO algorithm. This proposed framework is intended to provide practical guidance and support throughout the microservices design process.

Declaration of competing interest

The author declares that he has no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The author would like to express gratitude for the assistance provided by prof. Joanna Szłapczyńska and Anna Szopińska in reviewing the manuscript.

References

[1] J. Lewis and M. Fowler, "Microservices - a definition of this new architectural term." https://martinfowler.com/articles/microservices.html, 2014. Accessed: 2025-01-17.

[2] L. Qian, J. Li, X. He, R. Gu, J. Shao, and Y. Lu, "Microservice extraction using graph deep clustering based on dual view fusion," *Information and Software Technology*, vol. 158, p. 107171, 2023. https://doi.org/10.1016/j.infsof.2023.107171.

[3] S. Newman, *Building Microservices*. O'Reilly Media, 2021.

[4] S. Li, H. Zhang, Z. Jia, C. Zhong, C. Zhang, Z. Shan, J. Shen, and M. A. Babar, "Understanding and addressing quality attributes of microservices architecture: A systematic literature review," *Information and Software Technology*, vol. 131, p. 106449, 2021. https://doi.org/10.1016/j.infsof.2020.106449.

[5] H. Khalloof, E. Braun, W. Jakob, S. Shahoud, V. Hagenmeyer, J. Liu, and C. Duepmeier, "A generic distributed microservices and container based framework for metaheuristic optimization," in *GECCO 2018 Companion - Proceedings of the 2018 Genetic and Evolutionary Computation Conference Companion*, pp. 1363–1370, ACM, 7 2018. https://doi.org/10.1145/3205651.3208253.

[6] M. Oberle and T. Bauernhansl, "Leveraging openapi for microservice decomposition: A comparative study on features, encodings and algorithms on a real mes," in *6th International Conference on Artificial Intelligence in Information and Communication, ICAIIC 2024*, pp. 785–791, 2 2024. https://doi.org/10.1109/ICAIIC60209.2024.10463497.

[7] M. G. Moreira and B. B. N. D. França, "Analysis of microservice evolution using cohesion metrics," *ACM International Conference Proceeding Series*, pp. 40–49, 10 2022. https://doi.org/10.1145/3559712.3559716.

[8] S. Newman, *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. O'Reilly Media, Incorporated, 2019.

[9] R. Su, X. Li, and D. Taibi, "Back to the future: From microservice to monolith," 08 2023. https://doi.org/10.48550/arXiv.2308.15281.

[10] Y. Wei, Y. Yu, M. Pan, and T. Zhang, "A feature table approach to decomposing monolithic applications into microservices," *ACM International Conference Proceeding Series*, pp. 21–30, 11 2020. https://doi.org/10.1145/3457913.3457939.

[11] C. Richardson, *Microservices Patterns: With examples in Java*. Manning, 2018.

[12] D. Bajaj, U. Bharti, I. Gupta, P. Gupta, and A. Yadav, "Gtmicro—microservice identification approach based on deep nlp transformer model for greenfield developments," *International Journal of Information Technology*, vol. 16, pp. 2751–2761, 6 2024. https://doi.org/10.1007/s41870-024-01766-5.

[13] T. Kinoshita and H. Kanuka, "Enhancing automated microservice decomposition via multi-objective optimization," *IEEE Access*, vol. 12, pp. 55697–55710, 2024. https://doi.org/10.1109/ACCESS.2024.3389700.

[14] K. Sellami, M. A. Saied, and A. Ouni, "A hierarchical dbscan method for extracting microservices from monolithic applications," in *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering*, EASE '22, (New York, NY, USA), p. 201–210, Association for Computing Machinery, 2022. https://doi.org/10.1145/3530019.3530040.

[15] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, today, and tomorrow," *Present and ulterior software engineering*, pp. 195–216, 2017. https://doi.org/10.1007/978-3-319-67425-4_12.

[16] Z. Li, C. Shang, J. Wu, and Y. Li, "Microservice extraction based on knowledge graph from monolithic applications," *Information and Software Technology*, vol. 150, p. 106992, 2022. https://doi.org/10.1016/j.infsof.2022.106992.

[17] A. Muaz, M. E. Rana, and V. A. Hameed, "A framework for catering software complexity issues using architectural patterns," in *2021 3rd International Sustainability and Resilience Conference: Climate Change*, pp. 554–561, 11 2021. https://doi.org/10.1109/IEEECONF53624.2021.9668115.

[18] M. Horvath, V. Sakhnenko, and F. Gurbalb, "Comparison of scalability and performance in microservices and monolithic architectures," in *2024 IEEE 17th International Scientific Conference on Informatics, INFORMATICS 2024 - Proceedings*, pp. 82–87, 11 2024. https://doi.org/10.1109/Informatics62280.2024.10900892.

[19] O. Al-Debagy and P. Martinek, "A comparative review of microservices and monolithic architectures," in *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*, pp. 000149–000154, IEEE, 11 2018. https://doi.org/10.1109/CINTI.2018.8928192.

[20] N. N. Nayim, A. Karmakar, M. R. Ahmed, M. Saifuddin, and M. H. Kabir, "Performance evaluation of monolithic and microservice architecture for an e-commerce startup," in *2023 26th International Conference on Computer and Information Technology (ICCIT)*, pp. 1–5, Dec 2023. https://doi.org/10.1109/ICCIT60459.2023.10441241.

[21] G. Blinowski, A. Ojdowska, and A. Przybylek, "Monolithic vs. microservice architecture: A performance and scalability evaluation," *IEEE Access*, vol. 10, pp. 20357–20374, 2022. https://doi.org/10.1109/ACCESS.2022.3152803.

[22] I. Oumoussa and R. Saidi, "Evolution of microservices identification in monolith decomposition: A systematic review," *IEEE Access*, vol. 12, pp. 23389–23405, 4 2024. https://doi.org/10.1109/ACCESS.2024.3365079.

[23] N. A. Mansour, S. Hanae, K. Baina, and I. E. Kodssi, "Semantic approaches to microservice identification: A systematic literature review," 2025. https://doi.org/10.1109/ACCESS.2025.3618990.

[24] P. Valderas, V. Torres, and V. Pelechano, "A microservice composition approach based on the choreography of bpmn fragments," *Information and Software Technology*, vol. 127, p. 106370, 2020. https://doi.org/10.1016/j.infsof.2020.106370.

[25] S. S. de Toledo, A. Martini, and D. I. Sjøberg, "Identifying architectural technical debt, principal, and interest in microservices: A multiple-case study," *Journal of Systems and Software*, vol. 177, p. 110968, 2021. https://doi.org/10.1016/j.jss.2021.110968.

[26] H. Ünlü, D. E. Kennouche, G. K. Soylu, and O. Demirörs, "Microservice-based projects in agile world: A structured interview," *Information and Software Technology*, vol. 165, p. 107334, 2024. https://doi.org/10.1016/j.infsof.2023.107334.

[27] Y. Abgaz, A. Mccarren, P. Elger, D. Solan, N. Lapuz, M. Bivol, G. Jackson, M. Yilmaz, J. Buckley, and P. Clarke, "Decomposition of monolith applications into microservices architectures: A systematic review," *IEEE Transactions on Software Engineering*, vol. 49, pp. 4213–4242, 8 2023. https://doi.org/10.1109/TSE.2023.3287297.

[28] J. Sangabriel-Alarcón, J. O. Ocharán-Hernández, X. Limón, and K. Cortés-Verdín, "Domain-driven design in microservices-based systems development: A systematic literature review and thematic analysis [dataset]," *Programming and Computer Software*, vol. 50, pp. 742–770, 12 2023. https://doi.org/10.1134/S0361768824700749.

[29] J. Lourenco and A. R. Silva, "Monolith development history for microservices identification: a comparative analysis," in *Proceedings - 2023 IEEE International Conference on Web Services, ICWS 2023*, pp. 50–56, 7 2023. https://doi.org/10.1109/ICWS60048.2023.00019.

[30] T. I. Mohottige, A. Polyvyanyy, C. Fidge, R. Buyya, and A. Barros, "Reengineering software systems into microservices: State-of-the-art and future directions," *Information and Software Technology*, vol. 183, 7 2025. https://doi.org/10.1016/j.infsof.2025.107732.

[31] A. Pereira-Vale, E. B. Fernandez, R. Monge, H. Astudillo, and G. Márquez, "Security in microservice-based systems: A multivocal literature review," *Computers & Security*, vol. 103, p. 102200, 4 2021. https://doi.org/10.1016/j.cose.2021.102200.

[32] A. M. Saucedo, G. Rodríguez, F. G. Rocha, and R. P. dos Santos, "Migration of monolithic systems to microservices: A systematic mapping study," *Information and Software Technology*, vol. 177, 1 2025. https://doi.org/10.1016/j.infsof.2024.107590.

[33] M. J. Page, J. E. McKenzie, P. M. Bossuyt, I. Boutron, T. C. Hoffmann, C. D. Mulrow, L. Shamseer, J. M. Tetzlaff, E. A. Akl, S. E. Brennan, R. Chou, J. Glanville, J. M. Grimshaw, A. Hróbjartsson, M. M. Lalu, T. Li, E. W. Loder, E. Mayo-Wilson, S. McDonald, L. A. McGuinness, L. A. Stewart, J. Thomas, A. C. Tricco, V. A. Welch, P. Whiting, and D. Moher, "The prisma 2020 statement: an updated guideline for reporting systematic reviews," *BMJ*, vol. 372, 2021. https://doi.org/10.1136/bmj.n71.

[34] N. R. Haddaway, M. J. Page, C. C. Pritchard, and L. A. McGuinness, "Prisma2020: An r package and shiny app for producing prisma 2020-compliant flow diagrams, with interactivity for optimised digital transparency and open synthesis," *Campbell Systematic Reviews*, vol. 18, p. e1230, 6 2022. https://doi.org/10.1002/cl2.1230.

[35] O. Zimmermann, K. Luban, M. Stocker, and G. Bernard, "Continuous process model refinement from business vision to event simulation and software automation," in *Proceedings of the 5th International Workshop on Software-intensive Business: Towards Sustainable Software Business*, vol. 22, pp. 59–66, ACM, 5 2022. https://doi.org/10.1145/3524614.3528631.

[36] M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Márquez, "Design, monitoring, and testing of microservices systems: The practitioners' perspective," *Journal of Systems and Software*, vol. 182, p. 111061, 2021. https://doi.org/10.1016/j.jss.2021.111061.

[37] A. Brandolini, "Introducing event storming." https://ziobrando.blogspot.com/2013/11/introducing-event-storming.html, 2013. Accessed: 2025-05-06.

[38] J. Patton, "The new user story backlog is a map." https://jpattonassociates.com/the-new-backlog/, 2008. Accessed: 2025-01-26.

[39] E. van Kelle, G. Verschatse, and K. Baas-Schwegler, *Collaborative Software Design: How to facilitate domain modeling decisions*. Simon and Schuster, 2025.

[40] V. Vernon, *Domain-driven Design Distilled*. Addison-Wesley, 2016.

[41] V. Khononov, *Learning Domain-Driven Design*. O'Reilly Media, Inc., 2021.

[42] S. R. Goniwada, *Cloud Native Architecture and Design*. Apress, 1 2022.

[43] H. Gardner, A. F. Blackwell, and L. Church, "The patterns of user experience for sticky-note diagrams in software requirements workshops," *Journal of Computer Languages*, vol. 61, p. 100997, 2020. https://doi.org/10.1016/j.cola.2020.100997.

[44] J. Patton, *User Story Mapping: Building Better Products Using Agile Software Design*. O'Reilly & Associates, 2014.

[45] J. Patton, "Story mapping quick reference." https://jpattonassociates.com/wp-content/uploads/2015/03/story_mapping.pdf, 2015. Accessed: 2025-01-26.

[46] H. Farsi, D. Allaki, A. En-Nouaary, and M. Dahchour, "Following domain driven design principles for microservices decomposition: Is it enough?," in *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications, AICCSA*, vol. 2021-Decem, IEEE, 2021. https://doi.org/10.1109/AICCSA53542.2021.9686947.

[47] K. C. Febryanto, R. Sarno, A. F. Septiyanto, K. R. Sungkono, S. I. Sabilla, and Sholiq, "Leveraging graph-based for enhanced service identification and microservices partitioning in business process systems," in *2024 Beyond Technology Summit on Informatics International Conference (BTS-I2C)*, pp. 119–124, 12 2024. https://doi.org/10.1109/BTS-I2C63534.2024.10941720.

[48] I. Pigazzini, F. A. Fontana, V. Lenarduzzi, and D. Taibi, "Towards microservice smells detection," in *Proceedings - 2020 IEEE/ACM International Conference on Technical Debt, TechDebt 2020*, pp. 92–97, 5 2020. https://doi.org/10.1145/3387906.3388625.

[49] L. Cao and C. Zhang, "Implementation of domain-oriented microservices decomposition based on node-attributed network," *ACM International Conference Proceeding Series*, pp. 136–142, 2 2022. https://doi.org/10.1145/3524304.3524325.

[50] X. Zhou, S. Li, L. Cao, H. Zhang, Z. Jia, C. Zhong, Z. Shan, and M. A. Babar, "Revisiting the practices and pains of microservice architecture in reality: An industrial inquiry," *Journal of Systems and Software*, vol. 195, 1 2023. https://doi.org/10.1016/j.jss.2022.111521.

[51] C. Hou, T. Jia, Y. Wu, Y. Li, and J. Han, "Diagnosing performance issues in microservices with heterogeneous data source," in *19th IEEE International Symposium on Parallel and Distributed Processing with Applications, 11th IEEE International Conference on Big Data and Cloud Computing, 14th IEEE International Conference on Social Computing and Networking and 11th IEEE Internation*, pp. 493–500, 9 2021. https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00074.

[52] C. Zhong, H. Huang, H. Zhang, and S. Li, "Impacts, causes, and solutions of architectural smells in microservices: An industrial investigation," *Software: Practice and Experience*, vol. 52, pp. 2574–2597, 12 2022. https://doi.org/10.1002/spe.3138.

[53] T. de Oliveira Rosa, E. M. Guerra, F. F. Correia, and A. Goldman, "Charm — evaluating a model for characterizing service-based architectures," *Journal of Systems and Software*, vol. 206, 12 2023. https://doi.org/10.1016/j.jss.2023.111826.

[54] K. Maggi, R. Verdecchia, L. Scommegna, and E. Vicario, "Evolution of code technical debt in microservices architectures," *Journal of Systems and Software*, vol. 222, 4 2025. https://doi.org/10.1016/j.jss.2024.112301.

[55] K. Sellami, A. Ouni, M. A. Saied, S. Bouktif, and M. W. Mkaouer, "Improving microservices extraction using evolutionary search," *Information and Software Technology*, vol. 151, p. 106996, 2022. https://doi.org/10.1016/j.infsof.2022.106996.

[56] S. Li, H. Zhang, Z. Jia, Z. Li, C. Zhang, J. Li, Q. Gao, J. Ge, and Z. Shan, "A dataflow-driven approach to identifying microservices from monolithic applications," *Journal of Systems and Software*, vol. 157, p. 110380, 2019. https://doi.org/10.1016/j.jss.2019.07.008.

[57] V. Nitin, S. Asthana, B. Ray, and R. Krishna, "Cargo: Ai-guided dependency analysis for migrating monolithic applications to microservices architecture," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, (New York, NY, USA), Association for Computing Machinery, 2023. https://doi.org/10.1145/3551349.3556960.

[58] A. Selmadji, A.-D. Seriai, H. L. Bouziane, R. Oumarou Mahamane, P. Zaragoza, and C. Dony, "From monolithic architecture style to microservice one based on a semi-automatic approach," in *2020 IEEE International Conference on Software Architecture (ICSA)*, pp. 157–168, March 2020. https://doi.org/10.1109/ICSA47634.2020.00023.

[59] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng, "Service candidate identification from monolithic systems based on execution traces," *IEEE Transactions on Software Engineering*, vol. 47, pp. 987–1007, May 2021. https://doi.org/10.1109/TSE.2019.2910531.

[60] D. Taibi and K. Systä, "From monolithic systems to microservices: A decomposition framework based on process mining," in *CLOSER 2019 - Proceedings of the 9th International Conference on Cloud Computing and Services Science* (D. Ferguson, V. Munoz, M. Helfert, and C. Pahl, eds.), pp. 153–164, SCITEPRESS, 2019. https://doi.org/10.5220/0007755901530164.

[61] L. Nunes, N. Santos, and A. Rito Silva, "From a monolith to a microservices architecture: An approach based on transactional contexts," in *Software Architecture* (T. Bures, L. Duchien, and P. Inverardi, eds.), (Cham), pp. 37–52, Springer International Publishing, 2019. https://doi.org/10.1007/978-3-030-29983-5_3.

[62] A. A. C. De Alwis, A. Barros, A. Polyvyanyy, and C. Fidge, "Function-splitting heuristics for discovery of microservices in enterprise systems," in *Service-Oriented Computing* (C. Pahl,

M. Vukovic, J. Yin, and Q. Yu, eds.), (Cham), pp. 37–53, Springer International Publishing, 2018. https://doi.org/10.1007/978-3-030-03596-9_3.

[63] Z. Ren, W. Wang, G. Wu, C. Gao, W. Chen, J. Wei, and T. Huang, "Migrating web applications from monolithic structure to microservices architecture," in *Proceedings of the 10th Asia-Pacific Symposium on Internetware*, Internetware '18, (New York, NY, USA), Association for Computing Machinery, 2018. https://doi.org/10.1145/3275219.3275230.

[64] A. F. A. A. Freire, A. F. Sampaio, L. H. L. Carvalho, O. Medeiros, and N. C. Mendonça, "Migrating production monolithic systems to microservices using aspect oriented programming," *Software: Practice and Experience*, vol. 51, pp. 1280–1307, 6 2021. https://doi.org/10.1002/spe.2956.

[65] A. Cholomskis, O. Pozdniakova, and D. Mažeika, *Cloud Software Performance Metrics Collection and Aggregation for Auto-Scaling Module*, vol. 920, pp. 130–138. SPRINGER-VERLAG BERLIN, 2018. https://doi.org/10.1007/978-3-319-99972-2_10.

[66] C. Casse, P. Berthou, P. Owezarski, and S. Josset, "Using distributed tracing to identify inefficient resources composition in cloud applications," in *2021 IEEE 10th International Conference on Cloud Networking, CloudNet 2021*, pp. 40–47, IEEE, 11 2021. https://doi.org/10.1109/CloudNet53349.2021.9657140.

[67] G. Quattrocchi, D. Cocco, S. Staffa, A. Margara, and G. Cugola, "Cromlech: Semi-automated monolith decomposition into microservices," *IEEE Transactions on Services Computing*, vol. 17, no. 2, pp. 466–481, 2024. https://doi.org/10.1109/TSC.2024.3354457.

[68] S. Staffa, G. Quattrocchi, A. Margara, and G. Cugola, "Pangaea: Semi-automated monolith decomposition into microservices," in *Service-Oriented Computing* (H. Hacid, O. Kao, M. Mecella, N. Moha, and H.-y. Paik, eds.), (Cham), pp. 830–838, Springer International Publishing, 2021. https://doi.org/10.1007/978-3-030-91431-8_60.

[69] V. Faria and A. R. Silva, "Code vectorization and sequence of accesses strategies for monolith microservices identification," in *Web Engineering* (I. Garrigós, J. M. Murillo Rodríguez, and M. Wimmer, eds.), (Cham), pp. 19–33, Springer Nature Switzerland, 2023. https://doi.org/10.1007/978-3-031-34444-2_2.

[70] N. Knoll and R. Lichtenthäler, "An experimental evaluation of relations between architectural and runtime metrics in microservices systems," in *International Conference on Cloud Computing and Services Science, CLOSER - Proceedings* (M. VanSteen and C. Pahl, eds.), vol. 2023-April, pp. 147–154, SCITEPRESS - Science and Technology Publications, 2023. https://doi.org/10.5220/0011728600003488.

[71] S. Apel, F. Hertrampf, and S. Späthe, *Towards a Metrics-Based Software Quality Rating for a Microservice Architecture*, vol. 1041, pp. 205–220. Springer Verlag, 2019. https://doi.org/10.1007/978-3-030-22482-0_15.

[72] L. Carvalho, A. Garcia, W. K. G. Assuncao, R. de Mello, and M. J. de Lima, "Analysis of the criteria adopted in industry to extract microservices," in *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*, pp. 22–29, IEEE, 5 2019. https://doi.org/10.1109/CESSER-IP.2019.00012.

[73] T. Lopes and A. R. Silva, "Monolith microservices identification: Towards an extensible multiple strategy tool," in *Proceedings - IEEE 20th International Conference on Software Architecture Companion, ICSA-C 2023*, pp. 111–115, 3 2023. https://doi.org/10.1109/ICSA-C57050.2023.00034.

[74] S. Santos and A. R. Silva, "Microservices identification in monolith systems: Functionality redesign complexity and evaluation of similarity measures," *Journal of Web Engineering*, vol. 21, pp. 1543–1582, 8 2022. https://doi.org/10.13052/jwe1540-9589.2158.

[75] O. Al-Debagy and P. Martinek, "Extracting microservices' candidates from monolithic applications: Interface analysis and evaluation metrics approach," in *2020 IEEE 15th International Conference of System of Systems Engineering (SoSE)*, pp. 289–294, 6 2020. https://doi.org/10.1109/SoSE50414.2020.9130466.

[76] Y. Zhang, B. Liu, L. Dai, K. Chen, and X. Cao, "Automated microservice identification in legacy systems with functional and non-functional metrics," in *2020 IEEE International Conference on Software Architecture (ICSA)*, pp. 135–145, 3 2020. https://doi.org/10.1109/ICSA47634.2020.00021.

[77] X. Liu, Z. Chen, Y. Qian, C. Zhong, H. Huang, S. Li, and D. Shao, "Towards a security-optimized approach for the microservice-oriented decomposition," *Journal of Software: Evolution and Process*, vol. 36, 10 2024. https://doi.org/10.1002/smr.2670.

[78] D. Guaman, L. Yaguachi, C. C. Samanta, J. H. Danilo, and F. Soto, "Performance evaluation in the migration process from a monolithic application to microservices," in *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, pp. 1–8, 6 2018. https://doi.org/10.23919/CISTI.2018.8399148.

[79] S. Hassan, R. Bahsoon, and R. Buyya, "Systematic scalability analysis for microservices granularity adaptation design decisions," *Software: Practice and Experience*, vol. 52, pp. 1378–1401, 2022. https://doi.org/10.1002/spe.3069.

[80] S. Behrad, D. Espes, P. Bertin, and C.-T. Phan, "Impacts of service decomposition models on security attributes: A case study with 5g network repository function," in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, pp. 470–476, 6 2021.

[81] N. Mateus-Coelho, M. Cruz-Cunha, and L. G. Ferreira, "Security in microservices architectures," *Procedia Computer Science*, vol. 181, pp. 1225–1236, 2021. https://doi.org/10.1016/j.procs.2021.01.320.

[82] A. Rezaei Nasab, M. Shahin, S. A. Hoseyni Raviz, P. Liang, A. Mashmool, and V. Lenarduzzi, "An empirical study of security practices for microservices systems," *Journal of Systems and Software*, vol. 198, p. 111563, 2023. https://doi.org/10.1016/j.jss.2022.111563.

[83] A. E. Malki and U. Zdun, "Combining api patterns in microservice architectures: Performance and reliability analysis," in *Proceedings - 2023 IEEE International Conference on Web Services, ICWS 2023*, pp. 246–257, 7 2023. https://doi.org/10.1109/ICWS60048.2023.00044.

[84] U. Zdun, P. J. Queval, G. Simhandl, R. Scandariato, S. Chakravarty, M. Jelic, and A. Jovanovic, "Microservice security metrics for secure communication, identity management, and observability," *ACM Transactions on Software Engineering and Methodology*, vol. 32, p. 16, 2 2023. https://doi.org/10.1145/3532183.

[85] T. Cerny, A. S. Abdelfattah, A. A. Maruf, A. Janes, and D. Taibi, "Catalog and detection techniques of microservice anti-patterns and bad smells: A tertiary study," *Journal of Systems and Software*, vol. 206, 12 2023. https://doi.org/10.1016/j.jss.2023.111829.

[86] M.-D. Cojocaru, A. Uta, and A.-M. Oprescu, "Attributes assessing the quality of microservices automatically decomposed from monolithic applications," in *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*, pp. 84–93, 6 2019. https://doi.org/10.1109/ISPDC.2019.00021.

[87] M. Cojocaru, A. Uta, and A.-M. Oprescu, "Microvalid: A validation framework for automatically decomposed microservices," in *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 78–86, 12 2019. https://doi.org/10.1109/CloudCom.2019.00023.

[88] M. Matias, E. Ferreira, N. Mateus-Coelho, and L. Ferreira, "Enhancing effectiveness and security in microservices architecture," *Procedia Computer Science*, vol. 239, pp. 2260–2269, 2024. https://doi.org/10.1016/j.procs.2024.06.417.

[89] A. Parikh, P. Kumar, P. Gandhi, and J. Sisodia, "Monolithic to microservices architecture - a framework for design and implementation," in *2022 1st International Conference on Computer, Power and Communications, ICCPC 2022 - Proceedings*, pp. 90–96, 12 2022. https://doi.org/10.1109/ICCPC55978.2022.10072238.

[90] J. Daniel, E. Guerra, T. Rosa, and A. Goldman, "Towards the detection of microservice patterns based on metrics," in *Proceedings - 2023 49th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2023*, pp. 132–139, 9 2023. https://doi.org/10.1109/SEAA60479.2023.00029.

[91] C. Zhong, H. Zhang, C. Li, H. Huang, and D. Feitosa, "On measuring coupling between microservices," *Journal of Systems and Software*, vol. 200, 6 2023. https://doi.org/10.1016/j.jss.2023.111670.

[92] M. A. Said, A. Ezzati, S. Mihi, and L. Belouaddane, "Enhancing reusability in microservice architecture," in *2024 4th International Conference on Innovative Research in Applied Science, Engineering and Technology, IRASET 2024*, pp. 1–7, 5 2024. https://doi.org/10.1109/IRASET60544.2024.10549318.

[93] A. Santos and H. Paula, "Microservice decomposition and evaluation using dependency graph and silhouette coefficient," *ACM International Conference Proceeding Series*, pp. 51–60, 9 2021. https://doi.org/10.1145/3483899.3483908.

[94] M. Abdellatif, A. Shatnawi, H. Mili, N. Moha, G. E. Boussaidi, G. Hecht, J. Privat, and Y. G. Guéhéneuc, "A taxonomy of service identification approaches for legacy software systems modernization," *Journal of Systems and Software*, vol. 173, 3 2021. https://doi.org/10.1016/j.jss.2020.110868.

[95] X. Wei, J. Li, X. He, W. Peng, Y. Zhu, R. Gu, Y. Zhu, and J. Huang, "Extracting microservices from monolithic applications using consistent graph enhanced graph transformer," *Journal of Systems and Software*, vol. 222, 4 2025. https://doi.org/10.1016/j.jss.2025.112345.

[96] Z. Ding, Y. Xu, B. Feng, and C. Jiang, "Microservice extraction based on a comprehensive evaluation of logical independence and performance," *IEEE Transactions on Software Engineering*, vol. 50, pp. 1244–1263, 5 2024. https://doi.org/10.1109/TSE.2024.3380194.

[97] A. K. Kalia, J. Xiao, R. Krishna, S. Sinha, M. Vukovic, and D. Banerjee, "Mono2micro: A practical and effective tool for decomposing monolithic java applications to microservices," in *ESEC/FSE 2021 - Proceedings of the 29th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1214–1224, ACM, 8 2021. https://doi.org/10.1145/3468264.3473915.

[98] L. Zhu, D. A. Tamburri, and G. Casale, "Radf: Architecture decomposition for function as a service," *Software: Practice and Experience*, vol. 54, pp. 566–594, 4 2024. https://doi.org/10.1002/spe.3276.

[99] H. Zhu and H. Cai, "Low-coupling and high-cohesion microservice partitioning method to support rapid development of steel industry management systems," in *Proceedings of the IEEE International Conference on Computer Communication and the Internet, ICCCI*, pp. 78–82, 6 2024. https://doi.org/10.1109/ICCCI62159.2024.10674297.

[100] M. Daoud, A. E. Mezouari, N. Faci, D. Benslimane, Z. Maamar, and A. E. Fazziki, "Towards an automatic identification of microservices from business processes," in *Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE*, vol. 2020-Septe, pp. 42–47, IEEE, 9 2020. https://doi.org/10.1109/WETICE49692.2020.00017.

[101] A. Ashraf, A. H. Yousef, and H. Mahdi, "An innovated microservices identification approach based on database and source code analysis," in *NILES 2024 - 6th Novel Intelligent and Leading Emerging Sciences Conference, Proceedings*, pp. 463–468, 10 2024. https://doi.org/10.1109/NILES63360.2024.10753165.

[102] K. Sooksatra, M. S. H. Chy, M. A. R. Arju, T. Cerny, and P. Rivas, "Using static analysis to aid monolith to microservice system transformation: Tuning fuzzy c-means in a vae-based gnn approach," in *Proceedings - 2024 39th ACM/IEEE International Conference on Automated Software Engineering Workshops, ASEW 2024*, pp. 43–53, Association for Computing Machinery, 2024. https://doi.org/10.1145/3691621.3694933.

[103] M. H. Hasan, M. H. Osman, N. I. Admodisastro, and M. S. Muhammad, "Ai-based quality-driven decomposition tool for monolith to microservice migration," *ACM International Conference Proceeding Series*, pp. 181–191, 10 2023. https://doi.org/10.1145/3634814.3634839.

[104] T. Zhong, Y. Teng, S. Ma, J. Chen, and S. Yu, "A microservices identification method based on spectral clustering for industrial legacy systems," in *2023 IEEE Globecom Workshops, GC Wkshps 2023*, pp. 1331–1337, 12 2023. https://doi.org/10.1109/GCWkshps58843.2023.10464508.

[105] S. Agarwal, R. Sinha, G. Sridhara, P. Das, U. Desai, S. Tamilselvam, A. Singhee, and H. Nakamuro, "Monolith to microservice candidates using business functionality inference," in *Proceedings - 2021 IEEE International Conference on Web Services, ICWS 2021*, pp. 758–763, 9 2021. https://doi.org/10.1109/ICWS53863.2021.00104.

[106] I. Trabelsi, M. Abdellatif, A. Abubaker, N. Moha, S. Mosser, S. Ebrahimi-Kahou, and Y. Guéhéneuc, "From legacy to microservices: A type-based approach for microservices identification using machine learning and semantic analysis," *Journal of Software: Evolution and Process*, vol. 35, 10 2023. https://doi.org/10.1002/smr.2503.

[107] F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi, "From monolithic systems to microservices: An assessment framework," *Information and Software Technology*, vol. 137, 9 2021. https://doi.org/10.1016/j.infsof.2021.106600.

[108] T. Kinoshita and H. Kanuka, "Automated microservice decomposition method as multi-objective optimization," in *2022 IEEE 19th International Conference on Software Architecture Companion, ICSA-C 2022*, pp. 112–115, IEEE COMPUTER SOC, 2022. https://doi.org/10.1109/ICSA-C54293.2022.00028.

[109] I. Saidani, A. Ouni, M. W. Mkaouer, and A. Saied, *Towards Automated Microservices Extraction Using Muti-objective Evolutionary Search*, vol. 11895 LNCS, pp. 58–63. SPRINGER INTERNATIONAL PUBLISHING AG, 2019. https://doi.org/10.1007/978-3-030-33702-5_5.

[110] K.-H. Chow, U. Deshpande, V. Deenadayalan, S. Seshadri, and L. Liu, "Atlas: Hybrid cloud migration advisor for interactive microservices," in *EuroSys 2024 - Proceedings of the 2024 European Conference on Computer Systems*, pp. 870–887, ACM, 4 2024. https://doi.org/10.1145/3627703.3629587.

[111] A. Christoforou, A. S. Andreou, M. Garriga, and L. Baresi, "Adopting microservice architecture: A decision support model based on genetically evolved multi-layer fcm," *Applied Soft Computing*, vol. 114, p. 108066, 1 2022. https://doi.org/10.1016/j.asoc.2021.108066.

[112] M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann, "Service cutter: A systematic approach to service decomposition," in *Service-Oriented and Cloud Computing* (M. Aiello, E. B. Johnsen, S. Dustdar, and I. Georgievski, eds.), (Cham), pp. 185–200, Springer International Publishing, 2016. https://doi.org/10.1007/978-3-319-44482-6_12.

[113] Y. Zhang, Y. Li, Y. Yang, S. Chen, J. Gao, W. Wang, and Y. Yin, "Rapidms: A tool for supporting rapid microservices generation and refinement from requirements model," in *Proceedings - 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS-C 2023*, pp. 45–49, 10 2023. https://doi.org/10.1109/MODELS-C59198.2023.00017.

[114] F. Freitas, A. Ferreira, and J. Cunha, "A methodology for refactoring orm-based monolithic web applications into microservices," *Journal of Computer Languages*, vol. 75, p. 101205, 6 2023. https://doi.org/10.1016/j.cola.2023.101205.

[115] A. Christoforou, L. Odysseos, and A. S. Andreou, "Migration of software components to microservices: Matching and synthesis," in *ENASE 2019 - Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering* (E. Damiani, G. Spanoudakis, and L. Maciaszek, eds.), pp. 134–146, SCITEPRESS - Science and Technology Publications, 2019. https://doi.org/10.5220/0007732101340146.

[116] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, April 2002. https://doi.org/10.1109/4235.996017.

[117] J. D. Schaffer, *Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University, 8 1985.

[118] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, pp. 577–601, 2014. https://doi.org/10.1109/TEVC.2013.2281535.

[119] E. Zitzler, M. Laumanns, and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm," tech. rep., ETH Zurich, Computer Engineering and Networks Laboratory, 2001. https://doi.org/10.3929/ethz-a-004284029.

[120] Q. Zhang and H. Li, "Moea/d: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, pp. 712–731, 12 2007. https://doi.org/10.1109/TEVC.2007.892759.

[121] S. Khoshnevis, "A search-based identification of variable microservices for enterprise saas," *Frontiers of Computer Science*, vol. 17, 6 2023. https://doi.org/10.1007/S11704-022-1390-4.

[122] C. Coello Coello and M. Lechuga, "Mopso: a proposal for multiple objective particle swarm optimization," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, vol. 2, pp. 1051–1056 vol.2, May 2002. https://doi.org/10.1109/CEC.2002.1004388.

[123] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *Parallel Problem Solving from Nature - PPSN VIII* (E. K., L. J. A., S. Jim, M.-G. J. Julián, B. J. A., R. J. E., T. Peter, K. Ata, S. H.-P. Y. Xin, and Burke, eds.), pp. 832–842, Springer Berlin Heidelberg, 2004. https://doi.org/10.1007/978-3-540-30217-9_84.

[124] S. Izadpanah, A. Rasoolzadegan, S. Abrishami, and A. Mousavi, "Improving microservices identification for migration to cloud-native applications," *IEEE Transactions on Services Computing*, pp. 1–15, 4 2026. https://doi.org/10.1109/TSC.2026.3658420.

[125] A. Mwotil, B. Kanagwa, and E. Bainomugisha, "Yonga: An adaptive metaheuristic-based microservice placement approach for low-resource distributed systems," *IEEE Access*, vol. 14, pp. 6647–6663, 4 2026. https://doi.org/10.1109/ACCESS.2026.3653120.

[126] W. D. F. Mendonça, W. K. G. Assunção, L. V. Estanislau, S. R. Vergilio, and A. Garcia, "Towards a microservices-based product line with multi-objective evolutionary algorithms," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, 7 2020. https://doi.org/10.1109/CEC48606.2020.9185776.

[127] W. K. Assunção, T. E. Colanzi, L. Carvalho, A. Garcia, J. A. Pereira, M. J. de Lima, and C. Lucena, "Analysis of a many-objective optimization approach for identifying microservices from legacy systems," *Empirical Software Engineering*, vol. 27, 3 2022. https://doi.org/10.1007/S10664-021-10049-7.

[128] G. Filippone, M. Autili, F. Rossi, and M. Tivoli, "Migration of monoliths through the synthesis of microservices using combinatorial optimization," in *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 144–147, 10 2021. https://doi.org/10.1109/ISSREW53611.2021.00056.

[129] R. Yedida, R. Krishna, A. Kalia, T. Menzies, J. Xiao, and M. Vukovic, "Lessons learned from hyper-parameter tuning for microservice candidate identification," in *Proceedings - 2021 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021*, pp. 1141–1145, IEEE, 11 2021. https://doi.org/10.1109/ASE51524.2021.9678704.

[130] R. Yedida, R. Krishna, A. Kalia, T. Menzies, J. Xiao, and M. Vukovic, "An expert system for redesigning software for cloud applications," *Expert Systems with Applications*, vol. 219, p. 119673, 6 2023. https://doi.org/10.1016/j.eswa.2023.119673.

[131] M. Camilli, C. Colarusso, B. Russo, and E. Zimeo, "Domain metric driven decomposition of data-intensive applications," in *Proceedings - 2020 IEEE 31st International Symposium on Software Reliability Engineering Workshops, ISSREW 2020*, pp. 189–196, 10 2020. https://doi.org/10.1109/ISSREW51248.2020.00071.

[132] M. Camilli, C. Colarusso, B. Russo, and E. Zimeo, "Actor-driven decomposition of microservices through multi-level scalability assessment," *ACM Transactions on Software Engineering and Methodology*, vol. 32, 7 2023. https://doi.org/10.1145/3583563.

[133] P. Di Francesco, P. Lago, and I. Malavolta, "Migrating towards microservice architectures: An industrial survey," in *2018 IEEE International Conference on Software Architecture (ICSA)*, pp. 29–2909, April 2018. https://doi.org/10.1109/ICSA.2018.00012.

[134] H. Zhang, S. Li, Z. Jia, C. Zhong, and C. Zhang, "Microservice architecture in reality: An industrial inquiry," in *2019 IEEE International Conference on Software Architecture (ICSA)*, pp. 51–60, March 2019. https://doi.org/10.1109/ICSA.2019.00014.

[135] G. Vale, F. F. Correia, E. M. Guerra, T. D. O. Rosa, J. Fritzsch, and J. Bogner, "Designing microservice systems using patterns: An empirical study on quality trade-offs," in *Proceedings - IEEE 19th International Conference on Software Architecture, ICSA 2022*, pp. 69–79, 3 2022. https://doi.org/10.1109/ICSA53651.2022.00015.

[136] Y. Wang, H. Kadiyala, and J. Rubin, "Promises and challenges of microservices: an exploratory study," *Empirical Software Engineering*, vol. 26, no. 4, p. 63, 2021. https://doi.org/10.1007/s10664-020-09910-y.

[137] X. Zhou and J. Xiong, "Automated microservice identification from design model," in *CIBDA 2022 - 3rd International Conference on Computer Information and Big Data Applications*, pp. 116–122, 3 2022. https://ieeexplore.ieee.org/abstract/document/9898929.

[138] N. A. Mansour, H. Sbai, and K. Baina, "Minds: An nlp-driven process mining framework for superior microservices decomposition," *IEEE Access*, vol. 13, pp. 208376–208396, 2025. https://doi.org/10.1109/ACCESS.2025.3639460.

[139] K. Sellami and M. A. Saied, "Extracting microservices from monolithic systems using deep reinforcement learning," *Empirical Software Engineering*, vol. 30, 2 2025. https://doi.org/10.1007/S10664-024-10547-4.

[140] M. Abdullah, W. Iqbal, and A. Erradi, "Unsupervised learning approach for web application auto-decomposition into microservices," *Journal of Systems and Software*, vol. 151, pp. 243–257, 2019. https://doi.org/10.1016/j.jss.2019.02.031.

[141] M. H. Dehghani, S. Kolahdouz-Rahimi, M. Tisi, and D. Tamzalit, "Facilitating the migration to the microservice architecture via model-driven reverse engineering and reinforcement learning," *Software and Systems Modeling*, vol. 21, pp. 1115–1133, 6 2022. https://doi.org/10.1007/S10270-022-00977-3.

[142] K. Sellami, O. Jebbar, A. Gannoun, and M. A. Saied, "Beyond decomposition: A llm-powered automated approach to refactoring monoliths into microservices," in *2025 25th International Conference on Software Quality, Reliability and Security (QRS)*, pp. 68–77, 7 2025. https://doi.org/10.1109/QRS65678.2025.00018.

[143] J. Gandhi, R. K. Medicherla, M. Patwardhan, D. Sharma, and R. Naik, "Microservices identification using llm," in *2025 40th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, pp. 22–25, 11 2025. https://doi.org/10.1109/ASEW67777.2025.00013.

[144] K. Sellami and M. A. Saied, "Monoembed: Enhancing llm representations for monolith to microservices decomposition through contrastive learning," *Empirical Software Engineering*, vol. 31, 2 2026. https://doi.org/10.1007/S10664-025-10732-Z.

[145] D. Chen, C. Ye, H. Zhou, S. Lai, and B. Li, "Enhancing microservice migration transformation from monoliths with graph neural networks," in *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 136–146, 3 2025. https://doi.org/10.1109/SANER64311.2025.00021.

[146] L. Mardhatillah and S. Rochimah, "Migration of monolithic to microservices with an extraction design pattern in single sign on (sso) module using graph neural network (gnn)," in *2024 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, pp. 208–213, 7 2024. https://doi.org/10.1109/ISITIA63062.2024.10667894.

[147] M. S. Yang and K. H. Hsu, "Using graph convolutional networks for identifying potential microservice candidates from monolith application," in *Proceedings of the 2024 10th International Conference on Applied System Innovation, ICASI 2024*, pp. 403–405, 4 2024. https://doi.org/10.1109/ICASI60819.2024.10547753.

[148] Y. Zhu, "Automated microservice decomposition using graph reinforcement learning for system modernization," in *2025 IEEE 7th International Conference on Civil Aviation Safety and Information Technology (ICCASIT)*, pp. 1364–1372, 10 2025. https://doi.org/10.1109/ICCASIT66611.2025.11348732.

[149] Y. Xu, Y. Li, S. Wu, L. Li, X. Zhu, M. Xi, and J. Yin, "Ggrme: A ggnn-based graph reconstruction method for microservice extraction," in *2025 IEEE International Conference on Web Services (ICWS)*, pp. 976–982, 7 2025. https://doi.org/10.1109/ICWS67624.2025.00129.

[150] G. Filippone, N. Q. Mehmood, M. Autili, F. Rossi, and M. Tivoli, "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)*, pp. 47–57, 3 2023. https://doi.org/10.1109/ICSA56044.2023.00013.

[151] V. Raj and S. Ravichandra, "A service graph based extraction of microservices from monolith services of service-oriented architecture," *Software: Practice and Experience*, vol. 52, pp. 1661–1678, 2022. https://doi.org/10.1002/spe.3081.

[152] X. Sun, S. Boranbaev, S. Han, H. Wang, and D. Yu, "Expert system for automatic microservices identification using api similarity graph," *Expert Systems*, vol. 41, p. e13158, 2024. https://doi.org/10.1111/exsy.13158.

[153] X. Zhou, Y. Jin, H. Zhang, S. Li, and X. Huang, "A map of threats to validity of systematic literature reviews in software engineering," in *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, pp. 153–160, Dec 2016. https://doi.org/10.1109/APSEC.2016.031.