Practitioners' Perspective on Microservices Design Areas Challenges: A Socio-Technical Grounded Theory Literature Review

Muhammad Hamza[1], Muhammad Azeem Akbar[1], Kari Smolander[1] and Arif Ali Khan[2]

[1] *LUT University, 53851, Lappeenranta, Finland*
[2] *University of Oulu, 90570, Oulu, Finland*

Abstract
Microservice architecture has gained hype both in industry and academia. Companies are migrating their legacy monolithic systems to microservices architecture due to its promised benefits (e.g., agility and scalability). However, practitioners have faced many challenges in microservices architecture's design, development, and operation. This study investigates the challenging factors that could negatively affect the adoption of microservices architecture, as revealed by practitioners in empirical studies. We performed a socio-technical grounded theory literature review (ST-GTLR) and identified 24 key challenges from 31 empirical studies. The identified challenges were labeled as codes and mapped into seven concepts. Finally, the concepts were merged into three core categories design, development, and infrastructure. Our results serve as a body of knowledge for practitioners and researchers to understand the challenging aspects of microservices architecture in design areas.

Keywords
Microservices Architecture (MSA), Design Areas, Challenges, ST-GTLR

1. Introduction

Software organizations are continuously seeking solutions to improve product quality. To this end, many companies are migrating their monolithic systems to a microservices architecture to achieve better scalability, maintainability, and deliverability. In traditional monolithic architecture, all layers of application, e.g., user interface, business, and database, are developed as a single logical executable unit [1], which raises several challenges, such as scalability, maintainability, and deliverability. To tackle the mentioned challenges, the concept of microservices is introduced. Lewis and Flower [2], defined microservices architecture (MSA) as the counterpart to the monolith: a single application composed of loosely coupled and independently deployable smaller services.

In recent years, MSA concepts have largely been adopted across a vast array of industrial solutions. Technology giants like Amazon, Netflix, Spotify, Uber, and Twitter have successfully migrated the conventional monolithic system architectures to microservices architecture MSA to achieve structural, functional, and data decoupling [3]. Modernizing the legacy system with microservices architecture enables faster delivery, improved scalability, and greater autonomy.

However, adopting microservices architecture is not a one-go process as various challenges are likely to appear [4], e.g., decentralization of microservices requires more exertion to optimize the communication among services and to trace performance [5]. Haselböck et al. [6] have identified design areas of microservices architecture, such as microservices security, testing, communication, etc., by conducting expert interviews. Knoche and Hasselbringa [7] investigated the main motivation, challenges, and goals of adopting microservices by conducting a survey study. Similarly, Jamshidi et al. [5] identified microservices architecture's drivers, evolution, and future challenges. Yarygina et al. [8] identified and categorized security challenges in a microservices architecture. Viggiato et al. [9] conducted an industrial survey and investigated prevalent programming languages, their advantages, and challenges, e.g., distributed transactions, integration tests, service faults, and remote procedure calls in microservices systems. Zhou et al. [11] conducted a survey study of practitioners and investigated the fault analysis of a microservices system and practices for debugging. Similarly, Wang et al. [12] empirically investigated the challenges (e.g., common code management across services) faced by practitioners and their solutions in the microservices system. Moreover, Taibi et al. [13] reported migration

TKTP 2023: Annual Symposium for Computer Science 2023, June 13-14, 2023, Oulu, Finland

motivation to MSA, benefits, and challenges (e.g., data splitting). Several studies identified challenges covering different aspects of microservices [7] [8].

However, all these studies report different challenges from each other. Some studies discuss the architecture, whereas others discuss the security perspective. Therefore, no systematic study analyzes all the empirical studies that classify the microservices system's most common challenges in each design area (design, development, and operations). This study aims to identify the challenges for each design area from the empirical studies and develop a taxonomy of challenges that practitioners faced while designing, developing, and operating MSA. To achieve the study objectives, we conducted a socio-technical grounded theory literature review (ST-GTLR) to understand the industrial practitioner's perspective on the challenges and map the identified challenges into the design areas (e.g., design, development, and operation) of microservices architecture. ST-GTLR is based on the grounded theory literature review guidelines developed by Wolfswinkel et al. [18]. GTLR studies have been conducted in healthcare [19], [20], education [21], and banking [22]. This approach has also been implemented in information system research [23], and the software engineering domain [24]. Hence, in this study, we derived the following research question (RQ):

[RQ]: What are the challenges in the design area of microservices architecture, reported by practitioners in state-of-the-art empirical studies?

2. Methodology

2.1. Socio-Technical Grounded Theory Literature Review

Socio-technical grounded theory literature review (ST-GLTR) is an iterative and responsive approach that applies a five-phase framework (i.e., define, search, select, analyze, and present) of the original grounded theory literature review (GTLR) with concrete data analysis framework of socio-technical grounded theory [17].

We conducted ST-GTLR to explore challenges that practitioners face when working with microservices architecture and to develop a taxonomy of challenges with respect to design areas of microservices architecture. The main motivation of this review study is to develop a taxonomy of challenges, understand various facets of the practice areas, and guide future research in microservices architecture. Thus, all the steps to perform the ST- GTLR are presented in Figure 2, with a detailed illustration of the analysis phase. As per our knowledge, we are the first to use the ST- GTLR approach to explore the challenges of microservices architecture design areas. However, the detailed first version of the ST-GTLR was implemented in the domain of software engineering [24]. A detailed description of each stage of ST-GTLR is presented in the following subsections.

Define: In the initial phase of ST-GTLR, we set the study's scope by defining the research question and criteria, as shown in Table 1. This guided the creation of the search string, developed through a thorough analysis of keywords by the first two authors.

Furthermore, the search string was created using 'AND' and 'OR' Boolean operators to combine key terms and synonyms. After a team meeting to test eight candidate strings, the final version was selected and run on five major databases—ACM, Science Direct, IEEE Xplore, Wiley OL, and Springer Link—yielding 2,830 studies. These databases were chosen by unanimous author agreement and are commonly used in software engineering reviews [25].

Table 1
Inclusion and Exclusion criteria

Inclusion	Exclusion
The selected study must be full text published in Journal, Conference, as a thesis, and report or paper on arXiv.	Studies include microservices terms but do not focus on empirical challenges.
The study must be published in English.	Papers published in workshop, short paper (less than 4 pages).
The study must present empirical findings regarding practitioners' perspectives on microservices.	Grey literature, books, and incomplete work.
A study that presents empirical results on challenges in the implementation of microservices architecture.	Review and duplicate articles.

Search: In the second stage, the actual search was performed using the review protocols defined in the first stage's guidelines by Wolfswinkel et al. [18]. The search process was iterative and time-consuming because we had to revisit the define section as some necessary synonyms of search terms were missed. The search process was started on 21 October 2022 and ended on 13 November 2022. By executing the final search string, 2830 articles were collected. The total number of studies was large in number; thus, we exerted inclusion and exclusion criteria presented in Table 1.

Select: In the third stage, literature was filtered based on criteria in Table 1, narrowing it down to 2067 articles from various sources like journals, conferences, and arXiv. Further refinement led to 23 primary studies. The first author then used a backward snowballing approach, adding 8 more studies. The final list included 31 studies (23+8), as shown in Figure 1. If new articles emerged from snowballing, the process would restart; an article was finalized only if no new ones were found.

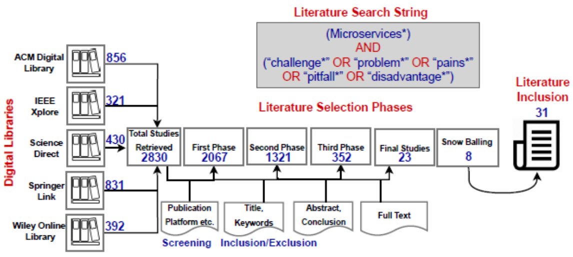


Figure 1: Studies selection process

Analyze: STGT is an effective approach to deeply understanding the state-of-the-art literature and exploring the specific research problem reported in the state-of-the-art literature [17]. Hence, we used the step-by-step guidelines STGTLR proposed by Hoda [17] to understand the practitioner's perception concerning microservices architecture design areas challenges reported in primary studies and socio-technical phenomena [16]. Our research group is highly skilled in terms of the technical background of microservices architecture and qualitative and quantitative empirical research. We used MS Excel software to collect the qualitative data from selected primary studies and applied advanced STGT data analysis techniques.

We employed Socio-Technical Grounded Theory (STGT) in our ST-GTLR study, using traditional methods like open coding and constant comparison in the early stages, and advanced techniques like targeted Data Collection and Analysis (DCA) later. The data consisted of practitioners' statements on microservices architecture, analyzed using STGT methods like open coding and theoretical structuring. Details of the analysis are depicted on the right-hand side of Figure 2.

The basic stage – Open Coding: The open coding technique was performed to develop the codes from the finding section of primary empirical studies. For instance, the primary study stated, "Without a design of microservices systems, a system could become a nightmare for developers. Several risks can affect the microservices system, including poor work and time management," is coded as "Lak of design" Similarly, we developed several codes by considering the statement of practitioners in the finding section of the primary study. All study authors carefully verified the codes in multiple meetings to check whether they met the objective of our research question.

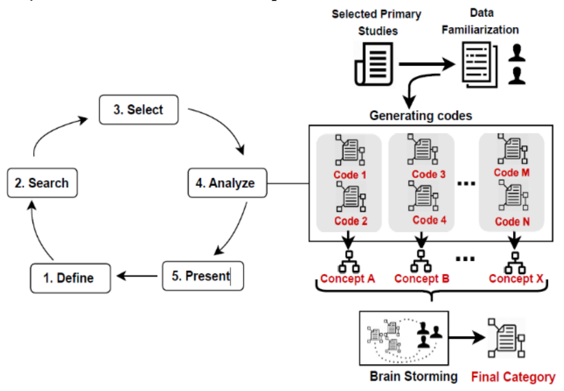
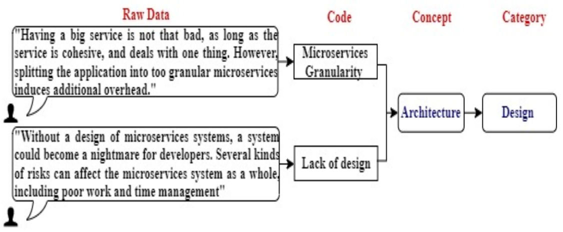


Figure 2: Socio-technical grounded theory literature review

Finally, we mapped the similar codes into their respective concepts, and similar concepts were mapped to respective categories by using the constant comparison technique. An example is presented in Figure 3 for better understanding. A similar process was performed in the findings section of all primary studies. We numbered each code as (C1, C2, C3 ....) and mapped them into respective categories. We again searched articles using snowballing once the concepts and categories were generated. We have provided the final replication package at http://tiny.cc/t6d2vz.

Advanced stage – Theory development: The codes generated through open coding techniques led us to the development of seven concepts and three categories. The seven concepts are microservices architecture, microservices security, testing, monitoring, development, microservices development, storage, and deployment. The concepts were mainly mapped into three categories: microservices design, development, and operation.


"Having a big service is not that bad, as long as the service is cohesive, and deals with one thing. However, splitting the application into too granular microservices induces additional overhead."

"Without a design of microservices systems, a system could become a nightmare for developers. Several kinds of risks can affect the microservices system as a whole, including poor work and time management"

Figure 3: Example of STGT data analysis

Present: In the final stage of the ST-GTLR, we presented the final findings through textual description along with the quotations taken from the final selected primary studies. Furthermore, Wolfswinkel et al. [18] recommend "presenting findings using visualizations such as diagrams can help reach a wider audience". The final findings are depicted in Figure 4.

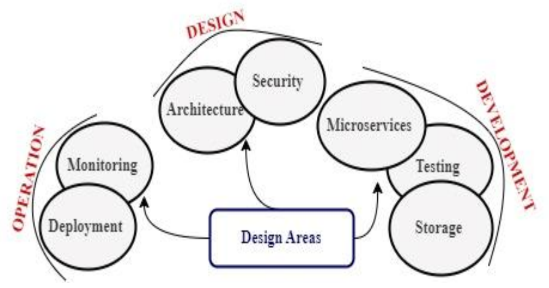


Figure 4: Mapping of concepts into categories

3. FINDINGS – KEY CATEGORIES

We derived three key categories from the rigorous analysis: (1) microservices design (2) microservices development (3) microservices operation. The codes were mapped into the seven concepts, and concepts were mapped into key categories. The mapping of concepts into categories is depicted in Figure 4, whereas their respective challenges are depicted in Figure 5.

3.1. Microservices design

Microservices design is the first category that emerged from the analysis. This category emerged from the underlying ***microservice architecture*** and ***microservices security*** concepts. The design of the microservices system should be comprised of loosely coupled microservices that can be developed, tested, and deployed independently [26].

Microservices architecture: Architecture is a crucial component of microservices system development. However, defining microservices architecture carries several challenges. The challenges are coded and numbered as C1, C2, C3.

C1 (Microservices granularity): Granularity defines the size of microservices, such as how big and short a microservices should be. However, identifying the right granularity of microservices is challenging from most practitioners' perspectives, whether they are experienced or newcomers reported in [12], [27], [28], [29], [13], [30], [31], [32], [33], [34], [35], [36], [37],[38], [39], [40], [41]. For example, the participants of [12] mentioned *"Identifying the right granularity of microservices is challenging; the size in lines of code is less crucial than having a cohesive service that focuses on one thing [....]"* (Page no. 15) [12].

C2 (Lack of microservices ownership): Services ownership states that a person should be accountable for the service in the entire lifecycle of its success or failure. Microservices ownership helps identify and fix bugs, implement new features, and train new recruiters [42]. However, companies do not have experienced persons for microservices ownership. Therefore, the lack of microservices ownership is also considered a significant challenge among practitioners [12], [27], [37], [41]. For instance, the participants of the study [37] stated *"Instead of excelling at one specific function, these 'jack-of-all-trades' services end up performing multiple tasks poorly, undermining the principle of doing one thing well"* (page no. 18) [37].

C3 (Language Diversity): The polyglot nature of the microservices architecture is more advertised as a developer can choose different programming languages for different microservices [41]. However, specifying different languages for different microservices may negatively affect the microservices system maintenance and testing [12], [13], [31], [34], [35], [36], [37], [43], [44], [40]. For instance, the participants of the study stated, *"We said, 'Hey, why not try using Golang? Why not try using Elixir?' [...] so we wrote a service in that language [......]"* (page no. 19) [43].

C4 (Lack of microservices design): Having a detailed design of the microservices system may assist (i) teams in estimating the amount of work required (ii) implementing security standards and solutions (iii) helping to understand the system for trainee developers [37]. However, companies generally do not create the design of the microservices systems. Therefore, the lack of microservices design is a significant challenge among practitioners [27], [31], [34], [37], [44], [40], [41], [7]. For instance, the participants of the study stated *"Without a design of microservices systems, a system could become a nightmare for developers. Several kinds of risks can affect the microservices system as a whole [....]"* (Page no. 17) [37].

C5 (Lack of knowledge on decomposing strategies): Decomposing monolithic systems is challenging due to the lack of a one-size-fits-all strategy. While methods like Domain-Driven Design (DDD) exist, companies often break down systems based on team structure, resource consumption, dependencies, and delivery cycles [37]. Therefore, the lack of knowledge on decomposing strategies is a significant challenge for some companies [28], [34], [37], [45], [46]. As one of the practitioners, *"We are not familiar with other strategies that can be used for decomposing the application than domain-driven design DDD and business capability. However, DDD is not always the right [...]"* (page 26) [45].

C6 (API Versioning): This is the way of managing the expected changes with full assurance that these changes will not disrupt the client. Even minor changes to your API can cause client applications to fail [40]. Therefore, it is highly encouraged not to make even minor changes to API. However, change is still unavoidable [12], [28], [29], [35], [36], [37], [39], [43], [47], [10], [9], [48]. Therefore, managing API versioning is a significant challenge among practitioners *"If we are going to delete something from the payload or we completely change the signature, we will have to bump up the major version and create another version of the API and ask people to move over [...]"* (Page no. 24) [29].

Microservices security is the second concept that emerged from the analysis of the microservices design category. Securing the microservices system is more intricate than securing monolithic architecture, as communication in the microservices system is done through the network that creates the surface attack. Besides this, malicious requests can be sent by a compromised microservice to other services in the system [8]. Therefore, practitioners face challenges when securing microservices architecture.

C7 (Lack of authentication and authorization): Authentication is the way of verifying the identity of an individual, whereas authorization verifies whether an individual is authorized to access data or services [49] [51]. The practitioner stated that *"I would say microservices can get exposed to, e.g., confused deputy problem where attackers can trick a service and get data that they should not be able to get if proper authorization is not enforced [....]"* (Page no. 8) [50].

C8 (Irrelevant privileges): This challenge arises when different microservices are given access to those functions that are not required by a particular service. These privileges could cause confidentiality and integrity issues [50], [51]. As reported by the practitioner *"In our case, what happens, e.g., when a service can write or read data stored in databases or messages posted in messages queues, even if such databases or queues are not needed by the service to deliver its business function [...]"* (Page no. 10) [51].

C9 (Lack of secure communication): Microservices architecture is highly distributed in nature, thus requiring a communication interface to interact with other microservices to perform business functionalities. The communication among microservices can be compromised by attackers [50], [51]. *"Let's say that the communication channel is not secured, and then it is possible that data transferred can be exposed to the man-in-the-middle, eavesdropping, and tampering attacks [...]" (page no. 11) [50].*

C10 (Implementing own cryptic algorithms): Most companies build and implement their own cryptographic algorithm to secure their microservices system. However, a system's confidentiality, Integrity, and authenticity can be compromised if the company's development team starts to build its own cryptographic algorithms [50], [51]. As reported by the practitioner: *"Microservice-based applications are not the exception: development teams that implement their encryption solutions may end with improper solutions [...]"* (Page no 6) [51].

C11 (Lack of data encryption): In most cases, the data in microservices storage are kept without any encryption or authenticated data protection method that the intruder eventually compromises. The data stored without encrypted could breach the confidentiality and Integrity of the system [50], [51] as explained by the practitioner *"When sensitive data is exposed, its Confidentiality and Integrity can get violated because it could be acquired or modified by an intruder who gets direct access to the microservices forming an application [...]"* (Page no. 7) [51].

3.2. Development

Microservices development is the second category that emerged from the rigorous analysis. This category is underlying by three main ***microservices development***, ***microservices testing***, and ***microservices storage*** concept. The review shows that practitioners face several challenges in each concept of this category.

Development: Microservices architecture includes small, loosely coupled services that can be independently developed, tested, and deployed into production. However, developing microservices raises several challenges, such as shared libraries/ managing common codes, variants, and cyclic dependencies [43]. Microservices development is the first concept in the underlying category of development.

C12 (Managing common codes/ shared libraries): Splitting all monolithic systems into small services that can be developed and deployed independently is not possible in most cases. Some systems require sharing the functionalities among other microservices, such as logging and authentication, database access, common utilities, etc. However, the common code cannot be shared among microservices as it will breach the common principle of microservices architecture [12], [30], [34], [52]. Therefore, practitioners stated it is challenging when managing shared libraries "*That is a [...] hassle because you change the common library, and all the services that depend on this library need to change"* (Page no. 23) [12].

C13 (Difficulties in managing variants): Most applications offer free or premium versions to their customers based on their necessities. However, managing the variants for different customers is still challenging for many companies. Most companies use the feature flag to handle it, but it requires extensive management of features [12], [36], [37]. Similarly, testing multiple features is also challenging *"When the feature is toggled for a customer, it is more like a temporary thing where it is like a hack. [...]"* (Page no. 26) [36].

C14 (Cyclic dependencies): This challenge arises when one microservices directly or indirectly rely on the other microservices to function properly, known as mutually recursive. Cyclic dependency will eventually increase the complexity of microservices [43], [44]. *"Having circular dependencies between microservices will result in hardly maintainable services. You'll be unable to think of a single service at a time [...]."* (Page no. 9) [43].

Microservices Testing is a crucial part of any system to deliver a quality product to end customers. However, monolithic testing is easier as one codebase is to test, whereas each microservice is tested in the distributed microservices architecture. There are several challenges that practitioners face when testing microservices-based systems.

C15 (Creating and implementing manual tests): Most systems are composed of many microservices in a microservices architecture. According to the survey by Kong [53], most companies have more than 184 microservices in their system. However, creating and implementing manual test cases is tedious for companies *"There are several reasons why manual testing could become a problem due to the number of microservices and communication between them"* (page no. 39) [37].

C16 (Integration testing of microservices): In integration testing, the tester usually examines the proper working of different modules in a sub-system when a high-level feature is introduced. However, integration testing becomes a challenge because of multiple connecting points where the tester has limited knowledge of other microservices [31], [35], [37]. *"Our major challenge is to write effective test cases for the integration testing of the microservices system, [...]"* (page no.36) [31].

Storage: In monolithic architecture, an application is composed of a single codebase that requires a single database, whereas microservices architecture comprises several services, and each service can have an independent database. If any change is made in the data model of a monolithic application, the entire database will be affected, whereas, in microservices, only the dedicated database will be affected. Furthermore, different services can have different storage requirements; one requires a relational database, while another requires MongoDB [54]. This distributed nature of microservices also poses storage challenges.

C17 (Distributed transactions): Microservices are distributed, and the transactions made by any client can also be distributed. They may span multiple computers over the network. However, distributed transactions lose the ACID (atomicity, consistency, isolation, durability) feature [13], [36], [39], [41], [55]. *"In an async[hronous] environment and having microservices be[ing] responsible for processing their changes, issues introduced with new releases on microservices may cause inconsistencies in data processing which are generally hard to correct after the fact"*.

C18 (Lack of consistency between heterogenous databases): A shared relational database is used to handle the data consistency in the monolithic architecture as companies use a single database for the entire application [13], [36], [39], [41], [55]. However, different microservices may use different database technologies, which may cause data consistency challenges *"Atomicity cannot be guaranteed over different storage technologies, no information or proper literature. Guessing and fixing error approach"* (Page no. 9) [39].

C19 (Query complexity): Microservice architectures extensively use online queries. There is, however, no de facto approach because developers typically rely on various ad hoc mechanisms for online queries [55]. According to the practitioner, *"We are integrating data from different sources in a global transportation network. The changes in data are flowing into our system consistently."* (Page no. 11) [55].

3.3. Operation

The third category is the operation that emerged from two underlying ***microservices monitoring*** and ***microservices deployment*** concepts. This category is related to the microservices operation team that governs and deploys the microservices in IT infrastructure. The operation team automates the repeatable task and maintain the consistency of the entire microservice system.

Microservices monitoring: Microservices systems have a distributed nature comprised of many independent services that run inside the container. However, it becomes a nightmare to track the availability and performance of numerous microservices and use the suitable monitoring infrastructure. We identified three challenges under this concept.

C20 (Lack of early setup of a logging and monitoring framework): Managing logs and monitoring in systems with numerous microservices is complex and often lacks infrastructure. A robust framework is essential from the project's start, a step often overlooked by practitioners [12], [37], [40]. Furthermore, practitioners also stated that the privacy of the client should be preserved by obscuring logged data *"Probably I will focus more on logging and monitoring, right off the bat. Because trying to retrofit monitoring and logging once we have all the services, is quite a bit of work"* (page no. 20) [37].

C21 (Lack of early setup of distributed tracing): Distributed tracing is a process of following a transaction request and recording all relevant data throughout the path of a microservices architecture. Failure is unavoidable, and when it occurs in the microservices system, most of the practitioners start from the failing services and gradually trace the source of occurrence of failure [12], [37], [40], [41]. Practitioners stated that companies should set up the distributed tracing as early as the project starts *"A lot of companies that start do not think about distributed tracing right from the get-go [...]"* (page no. 19) [12].

Microservice deployment is the second concept that has emerged in the infrastructure category. The deployment in a microservices architecture is different from that of a monolithic architecture. The operational team must deploy multiple microservices in microservices architecture, whereas only a Single deployment is required in a monolithic architecture. Considering the microservice deployment, we identified three main challenges in this concept.

C22 (Lack of an automatic process): Companies usually develop microservices and manually add them to the deployment pipeline. However, it took a lot of time as compared to automating the microservices stub, and creating the build and deployment pipeline for newly created microservices may reduce the operational cost and time [12], [37], [56]. As the practitioner stated *"The tooling is very important. There is one way to create, at least, the structure of projects for different platforms. So, like, Scala microservices, they will all look the same. They will have the same structure [...]"* (Page no. 22) [56].

C23 (Multiple services in one container): Containers are used for packaging up code and dependencies to deploy microservices ideally. However, many companies package multiple microservices in one container and deploy it. Packaging multiple microservices in one container can cause issues such as launching new instances for such services [37], [38], [41] *"We observe that placing multiple services in one container would constitute independent deployability of microservices. If two microservices would be packaged in the same Docker image [...]"*

C24 (Tool selection): With the hype of microservice architecture, numerous tools are built and publicly available in the market. There are several tools publicly available in the market for microservices development and deployment [57], [58]. However, the selection of the appropriate tool among many is a time-consuming task for many practitioners, particularly when there is a lack of knowledge on how these tools and technologies work [12], [37], [56] *"There are tools that are out or coming out that are solving a lot of problems that we have. Things like gRPC, GraphQL, code generation and documentation, service meshes, [...]"* (page no. 22) [56].

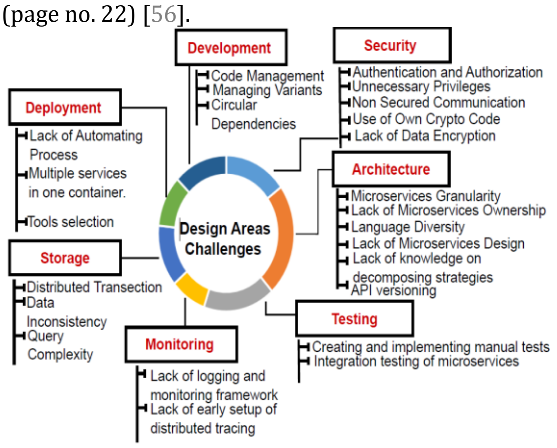

Development
Code Management
Managing Variants
Circular Dependencies
Security
Authentication and Authorization
Unnecessary Privileges
Non Secured Communication
Use of Own Crypto Code
Lack of Data Encryption
Deployment
Lack of Automating Process
Multiple services in one container.
Tools selection
Architecture
Microservices Granularity
Lack of Microservices Ownership
Language Diversity
Lack of Microservices Design
Lack of knowledge on decomposing strategies
API versioning
Design Areas Challenges
Storage
Distributed Transection
Data Inconsistency
Query Complexity
Monitoring
Lack of logging and monitoring framework
Lack of early setup of distributed tracing
Testing
Creating and implementing manual tests
Integration testing of microservices

Figure 5: Mapping of challenges in concepts

4. DISCUSSIONS

This section describes the conclusive summary of our research study, comparative analysis, implications, and limitations.

Conclusive summary: Microservices architecture offers benefits like easier development and deployment but isn't a one-size-fits-all solution. Our research question focused on challenges in microservices design as reported by practitioners. Using a Socio-Technical Grounded Theory Literature Review (ST-GTLR), we selected 31 relevant studies and identified 24 challenges. These were categorized into seven concepts and three categories. The study revealed that most of the challenges are in the microservices design and development category. Similarly, securing microservices architecture, particularly authentication and authorization of services, is extremely challenging due to the distributed nature of microservices. The study calls for industry-specific solutions to these challenges for successful microservices adoption.

Comparative analysis: Recently, several studies investigated different aspects of microservices architecture from state-of-the-art practices. Pahl and

Jamshidi [59] conducted a systematic mapping study on MSA where they taxonomically classified the emerging applications particularly related to cloud and container technologies. Similarly, Taibi et al. [60] conducted a systematic mapping study on microservices-based solutions' common patterns and principles. Furthermore, Di Francesco et al. [31] conducted an industrial survey to identify the activities and challenges when organizations migrate to microservices architecture. Similarly, Baskarada et al. [10] conducted an in-depth interview with industrial practitioners and identified the challenges and practical opportunities. Xiang et al. [38] empirically investigated the activities performed during the migration of legacy monolithic architecture and the challenges faced during the migration. Similarly, Wang et al. [12] empirically collected challenges and best practices in microservices. architecture design and deployment from the practitioners who have successfully developed microservices systems. De Almeida and Canedo [61] conducted a systematic literature review on authentication and authorization in a microservices architecture. They have identified the challenges and practical solutions related to microservices security but did not cover other design areas. Lianger et al. [55] conducted an empirical investigation to identify the practices and challenges regarding data management in a microservices architecture but did not classify and develop the taxonomy of challenges in other design areas of microservices architecture. Soldani et al. [15] conducted systematic grey literature to identify the barriers and solutions of microservices architecture. They just included the grey literature which differs from our study which particularly identifies the challenges from empirical studies. Ghofrani and Lubke [30] empirically investigated the current state of practices on barriers and advantages of using a microservices architecture. However, no study provided and systematically classified the challenges that practitioners faced in each design area (design, development, and operation) of microservices. Both of these studies addressed different aspects of microservices architecture [11][13][40].

Study Implications: The results of this study contribute to academic research by explicitly exploring the available primary studies related to the microservices architecture design areas. Similarly, the study findings make a concrete research contribution by providing a significant overview of microservices architecture design area challenges. The implications for researchers and practitioners were distilled from seven concepts (microservices implementation, architecture, security, testing, storage, monitoring, deployment) identified in this study.

Practitioners face major challenges in the architecture and implementation phases of microservices, especially in decomposing monolithic systems. Researchers should focus on empirical studies to identify industry strategies for decomposition and create universal solutions. These should help define service boundaries for low coupling and offer guidelines on which applications would benefit from transitioning to microservices.

The security of microservices systems can be compromised due to the distributed nature of the system as it gains a large attack surface. Therefore, researchers and practitioners should develop unique solutions to trace the vulnerabilities.

Concerning microservices deployment, it is still challenging for novice developers to manage and configure CI/CD tools or services. Researchers should evaluate the tools employed in the industry, and practitioners should simplify the configuration of these tools to help better support, novice developers.

Managing and monitoring microservices is considered the second most challenging phase in the industry. Logging and monitoring traces have been critical for the developer of the microservices system. Therefore, the researcher should develop guidelines for early setup logging and monitoring. Furthermore, practitioners should develop simple tools for monitoring the traces.

The distributed nature of the microservices architecture poses significant challenges to data management. The data consistency, query complexity, and distributed transactions are some significant challenges. However, research and practitioners should define the guidelines on how data consistency can be ensured in a microservices architecture, where multiple microservices may access and modify the same data.

From a practical perspective, practitioners can use the study's findings to develop strategies for improving microservices architecture design areas. The taxonomy of reported challenges provides an overview of critical areas that need to be considered by industry experts before initiating the design activities of microservices architecture.

4.1. Threats to Validity

Several threats could influence the validity of our study. The potential threats of this study are analyzed based on internal, external, and conclusion validity.

Internal validity: We used ST-GTLR to focus on key aspects of the topic, acknowledging the risk of missing some studies due to our search strategy and keyword selection. To mitigate this, we used an iterative approach for keyword definition and study selection, validated by experienced authors. Studies were chosen based on the criteria in Table 1 and discussed for quality assurance. Personal bias in data extraction was minimized through regular discussions with senior authors. We included non-reviewed studies from arXiv for comprehensive insights. Coding was done by the first author and validated iteratively by expert co-authors to ensure accuracy.

External validity: refers to the extent to which the results of a study can be generalized to other populations, settings, or times. To mitigate this validity, we followed rigorous protocols of ST-GTLR.

Conclusion validity: The degree to which the study's conclusions are credible or reasonable is referred to as conclusion validity. The authors conducted brainstorming sessions to discuss the findings of the study to construct a correct conclusion.

5. Conclusion

The concept of microservices architecture is booming and has become more common due to its significant benefits, such as agility and scalability. However, design areas of microservices architecture (design, development, and operations) pose various challenges. Seeking the significance of design, development, and operations, we formulated the research question: what are the challenges in the design area of microservices architecture reported by practitioners in state-of-the-art empirical studies? To address the mentioned research question, we conducted a socio-technical grounded theory literature review (ST-GTLR) by applying the framework of grounded theory literature review (GTLR) and rigorous steps of socio-technical grounded theory (STGT) for analyzing the data of 31 primary studies. We applied open coding and targeted coding in the finding section of each empirical study. The codes were developed by analyzing the raw data of the finding sections. Further, codes were mapped into core concepts, and finally, core concepts were mapped into categories. Through rigorous analysis, we found 24 challenges mapped into seven concepts, i.e., architecture, security, development, testing, storage, monitoring, and deployment. All the concepts were mapped into three categories, i.e., microservices design, development, and operation. The challenges and their mapping would provide a holistic view to practitioners and researchers.

References

[1] S. Newman, Monolith to microservices: evolutionary patterns to transform your monolith. O'Reilly Media, 2019.

[2] J. Lewis and M. Fowler, "Microservices: a definition of this new architectural term," MartinFowler. com, vol. 25, pp. 14–26, 2014.

[3] C. Richardson, "Who is using microservices?" [Online]. Available: https://microservices.io/articles/whoisusingmicroservices.html

[4] M. Waseem, P. Liang, M. Shahin, A. Ahmad, and A. R. Nassab, "On the nature of issues in five open source microservices systems: An empirical study," in Evaluation and Assessment in Software Engineering, 2021, pp. 201–210.

[5] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," IEEE Software, vol. 35, no. 3, pp. 24– 35, 2018.

[6] S. Haselböck, R. Weinreich, and G. Buchgeher, "An expert interview study on areas of microservice design," in 2018 IEEE 11th Conference on service-oriented computing and applications (SOCA), 2018, pp. 137–144.

[7] H. Knoche and W. Hasselbring, "Drivers and barriers for microservice adoption– a survey among professionals in Germany," Enterprise Modelling and Information Systems Architectures (EMISAJ)–International Journal of Conceptual Modeling: Vol. 14, Nr. 1, 2019.

[8] T. Yarygina and A. H. Bagge, "Overcoming security challenges in microservice architectures," in 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE), 2018, pp. 11–20.

[9] M. Viggiato, R. Terra, H. Rocha, M. T. Valente, and E. Figueiredo, "Microservices in practice: A survey study," arXiv preprint arXiv:1808.04836, 2018.

[10] S. Baškarada, V. Nguyen, and A. Koronios, "Architecting microservices: Practical opportunities and challenges," Journal of Computer Information Systems, vol. 60, no. 5, pp. 428–436, 2020.

[11] X. Zhou et al., "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," IEEE Transactions on Software Engineering, vol. 47, no. 2, pp. 243–260, 2018.

[12] Y. Wang, H. Kadiyala, and J. Rubin, "Promises and challenges of microservices: an exploratory study," Empirical Software Engineering, vol. 26, no. 4, pp. 1–44, 2021.

[13] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation," IEEE Cloud Computing, vol. 4, no. 5, pp. 22–32, 2017.

[14] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," Journal of Systems and Software, vol. 150, pp. 77–97, 2019.

[15] J. Soldani, D. A. Tamburri, and W.-J. Van Den Heuvel, "The pains and gains of microservices: A systematic grey literature review," Journal of Systems and Software, vol. 146, pp. 215–232, 2018.

[16] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," Journal of Systems and Software, vol. 150, pp. 77–97, 2019.

[17] R. Hoda, "Socio-technical grounded theory for software engineering," IEEE Transactions on Software Engineering, 2021.

[18] J. F. Wolfswinkel, E. Furtmueller, and C. P. Wilderom, "Using grounded theory as a method for rigorously reviewing literature," European journal of information systems, vol. 22, no. 1, pp. 45–55, 2013.

[19] F. Nunes, N. Verdezoto, G. Fitzpatrick, M. Kyng, E. Grönvall, and C. Storni, "Self-care technologies in HCI: Trends, tensions, and opportunities," ACM Transactions on Computer-Human Interaction (TOCHI), vol. 22, no. 6, pp. 1–45, 2015.

[20] C. Meuwly et al., "Definition and diagnosis of the trigeminocardiac reflex: a grounded theory approach for an update," Frontiers in neurology, vol. 8, p. 533, 2017.

[21] K.-I. Andersson, "Developing a theory of open access: a grounded theory based literature review," 2016.

[22] A. R. Montazemi and H. Qahri-Saremi, "Factors affecting adoption of online banking: A meta-analytic structural equation modeling study," Information & management, vol. 52, no. 2, pp. 210–226, 2015.

[23] S. Utulu, K. Sewchurran, and B. Dwolatzky, “Systematic and Grounded Theory Literature Reviews of Software Process Improvement Phenomena: Implications for IS Research,” in Proceedings of the Informing Science and Information Technology Education Conference, 2013, pp. 249–279.

[24] A. Pant, R. Hoda, C. Tantithamthavorn, and B. Turhan, “Ethics in AI through the Developer’s Prism: A Socio-Technical Grounded Theory Literature Review and Guidelines,” arXiv preprint arXiv:2206.09514, 2022.

[25] D. Hidellaarachchi, J. Grundy, R. Hoda, and K. Madampe, “The effects of human aspects on the requirements engineering process: A systematic literature review,” IEEE Transactions on Software Engineering, 2021.

[26] A. Sill, “The design and architecture of microservices,” IEEE Cloud Computing, vol. 3, no. 5, pp. 76–80, 2016.

[27] A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Microservices architecture enables devops: Migration to a cloud-native architecture,” Ieee Software, vol. 33, no. 3, pp. 42–52, 2016.

[28] D. Taibi, V. Lenarduzzi, C. Pahl, and A. Janes, “Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages,” in Proceedings of the XP2017 Scientific Workshops, 2017, pp. 1– 5.

[29] J.-P. Gouigoux and D. Tamzalit, “From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture,” in 2017 IEEE international conference on software architecture workshops (ICSAW), 2017, pp. 62–65.

[30] J. Ghofrani and D. Lübke, “Challenges of Microservices Architecture: A Survey on the State of the Practice.,” ZEUS, vol. 2018, pp. 1–8, 2018.

[31] P. Di Francesco, P. Lago, and I. Malavolta, “Migrating towards microservice architectures: an industrial survey,” in 2018 IEEE International Conference on Software Architecture (ICSA), 2018, pp. 29–2909.

[32] W. Luz, E. Agilar, M. C. de Oliveira, C. E. R. de Melo, G. Pinto, and R. Bonifácio, “An experience report on the adoption of microservices in three Brazilian government institutions,” in Proceedings of the XXXII Brazilian Symposium on Software Engineering, 2018, pp. 32–41.

[33] H. H. S. da Silva, G. de F Carneiro, and M. P. Monteiro, “An experience report from the migration of legacy software systems to microservice based architecture,” in 16th International Conference on Information Technology-New Generations (ITNG 2019), 2019, pp. 183–189.

[34] J. Fritzsch, J. Bogner, S. Wagner, and A. Zimmermann, “Microservices Migration in Industry: Intentions, Strategies, and Challenges,” in 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), Cleveland, OH, USA, Sep. 2019, pp. 481–490. doi: 10.1109/ICSME.2019.00081.

[35] J. Bogner, J. Fritzsch, S. Wagner, and A. Zimmermann, “Assuring the evolvability of microservices: insights into industry practices and challenges,” in 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2019, pp. 546–556.

[36] J. Ghofrani and A. Bozorgmehr, “Migration to microservices: Barriers and solutions,” in International Conference on Applied Informatics, 2019, pp. 269– 281.

[37] 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.

[38] Q. Xiang et al., “No free lunch: Microservice practices reconsidered in industry,” arXiv preprint arXiv:2106.07321, 2021.

[39] M. Wu et al., “On the Way to Microservices: Exploring Problems and Solutions from Online Q&A Community,” in 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2022, pp. 432–443.

[40] 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), 2019, pp. 51–60.

[41] T. Colanzi et al., “Are we speaking the industry language? The practice and literature of modernizing legacy systems with microservices,” in 15th Brazilian Symposium on Software Components, Architectures, and Reuse, 2021, pp. 61–70.

[42] P. Kruchten, R. L. Nord, and I. Ozkaya, “Technical debt: From metaphor to theory and practice,” Ieee software, vol. 29, no. 6, pp. 18–21, 2012.

[43] D. Taibi and V. Lenarduzzi, “On the definition of microservice bad smells,” IEEE software, vol. 35, no. 3, pp. 56–62, 2018.

[44] D. Taibi, V. Lenarduzzi, and C. Pahl, “Microservices anti-patterns: A taxonomy,” in Microservices, Springer, 2020, pp. 111–128.

[45] V. Lenarduzzi, F. Lomio, N. Saarimäki, and D. Taibi, “Does migrating a monolithic system to microservices decrease the technical debt?,” Journal of Systems and Software, vol. 169, p. 110710, 2020.

[46] 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.

[47] H. Michael Ayas, P. Leitner, and R. Hebig, “The Migration Journey Towards Microservices,” in International Conference on Product-Focused Software Process Improvement, 2021, pp. 20–35.

[48] J. Lotz, A. Vogelsang, O. Benderius, and C. Berger, “Microservice architectures for advanced driver assistance systems: A case-study,” in 2019 IEEE International Conference on Software Architecture Companion (ICSA-C), 2019, pp. 45–52.

[49] J. Carnell and I. H. Sánchez, Spring microservices in action. Simon and Schuster, 2021.

[50] A. R. Nasab, M. Shahin, S. A. H. Raviz, P. Liang, A. Mashmool, and V. Lenarduzzi, “An Empirical Study of Security Practices for Microservices

Systems," arXiv preprint arXiv:2112.14927, 2021.

[51] P. Billawa, A. B. Tukaram, N. E. D. Ferreyra, J.-P. Steghöfer, R. Scandariato, and G. Simhandl, "Security of Microservice Applications: A Practitioners' Perspective on Challenges and Best Practices," arXiv preprint arXiv:2202.01612, 2022.

[52] S. S. de Toledo, A. Martini, and D. I. Sjøberg, "Improving agility by managing shared libraries in microservices," in International Conference on Agile Software Development, 2020, pp. 195–202.

[53] K. Kong, "APIs & Microservices Connectivity Report." [Online]. Available: https://konghq.com/resources/reports/2022-api-microservices-connectivity-report

[54] N. Viennot, M. Lécuyer, J. Bell, R. Geambasu, and J. Nieh, "Synapse: a microservices architecture for heterogeneous-database web applications," in Proceedings of the Tenth European Conference on Computer Systems, 2015, pp. 1–16.

[55] R. Laigner, Y. Zhou, M. A. V. Salles, Y. Liu, and M. Kalinowski, "Data management in microservices: State of the practice, challenges, and research directions," arXiv preprint arXiv:2103.00170, 2021.

[56] X. Zhou, H. Huang, H. Zhang, X. Huang, D. Shao, and C. Zhong, "A cross- company ethnographic study on software teams for DevOps and microservices: organization, benefits, and issues," in Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice, 2022, pp. 1–10.

[57] B. BHANDA, "Comprehensive List of DevOps Tools 2023." [Online]. Available: https://www.qentelli.com/thought-leadership/insights/devops-tools

[58] Forge, "Best Microservices Tools." [Online]. Available: https://sourceforge.net/software/microservices/

[59] C. Pahl and P. Jamshidi, "Microservices: A Systematic Mapping Study.," CLOSER (1), pp. 137–146, 2016.

[60] D. Taibi, V. Lenarduzzi, and C. Pahl, "Architectural patterns for microservices: a systematic mapping study," in CLOSER 2018: Proceedings of the 8th International Conference on Cloud Computing and Services Science; Funchal, Madeira, Portugal, 19-21 March 2018, 2018.

[61] M. G. de Almeida and E. D. Canedo, "Authentication and Authorization in Microservices Architecture: A Systematic Literature Review," Applied Sciences, vol. 12, no. 6, p. 3023, 2022