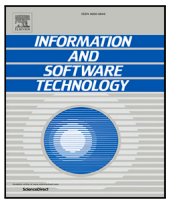
Information and Software Technology 183 (2025) 107723

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



A taxonomy of microservice integration techniques

Georg-Daniel Schwarz [a,*], Andreas Bauer [b], Dirk Riehle [a], Nikolay Harutyunyan [a]

[a] Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
[b] Blekinge Institute of Technology, Karlskrona, Sweden

ARTICLE INFO

Dataset link: https://zenodo.org/records/12740383

Keywords:
Microservices
Integration
Techniques
Taxonomy

ABSTRACT

Context: Microservices have become an important architectural style for building robust and scalable software systems. A system's functionality is split into independent units, the microservices, that communicate over a network and can be deployed independently. The shift of complexity into the integration layer necessitates enhanced collaboration among stakeholders, stressing the importance of effective communication.
Objective: We aim to streamline communication between stakeholders in microservice-based projects by constructing a framework for enhanced clarity, a taxonomy, by answering our research question: *"How can microservice integration techniques be classified?"*
Method: We conducted a thematic analysis of literature and six expert interviews to identify microservice integration techniques and construct a taxonomy.
Results: The results of this study are (i) a taxonomy for microservice integration techniques consisting of five main and ten refined categories, (ii) the classification of 121 found integration techniques, (iii) an illustration of the taxonomy usage based on three selected techniques to demonstrate the procedure in case of classification ambiguity, (iv) a comparison of data gathered from literature with the interviews, and (v) comprehensive supplementary materials.
Conclusion: The taxonomy offers a structured framework to classify microservice integration techniques and enhances the understanding of the diverse landscape of microservice integration techniques, including organizational ones that are often overlooked. Practitioners can discover integration techniques through the taxonomy and apply them with guidance provided in the supplementary materials.

1. Introduction

Microservices have become an essential architectural style to build robust and scalable software systems optimized to run in cloud environments [1]. Lewis and Fowler [2] define the microservice architectural style as *"an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and [are] independently deployable"*. If done right, development teams can work and deploy their microservice(s) independently of each other. The focused responsibility of microservices and the loose technological and organizational coupling enable large software projects with multiple teams to work in parallel [3].

However, microservice-based systems are distributed systems that come with their unique challenges. For instance, microservices cannot use in-process communication to integrate the deployed instances but communicate over the system's unreliable network layer. By shifting complexity into the integration layer, integration becomes a more predominant and explicit challenge [4]. This increased complexity necessitates enhanced collaboration among developers, architects, and stakeholders, making effective communication crucial. However, communication is challenging when the terms and concepts are not standardized.

To address this issue, we present a taxonomy for microservice integration techniques. A taxonomy's main purpose is to classify existing knowledge and promote a shared terminology and language [5]. This enhanced clarity improves decision-making, allowing teams to systematically evaluate and select the most appropriate integration techniques tailored to their specific needs. A well-structured taxonomy guides practitioners toward proven approaches and helps to avoid common pitfalls. Further, it serves as a powerful learning tool, providing a structured pathway for newcomers to swiftly grasp fundamental concepts and understand the relationships between different techniques. Collectively, these benefits support a more efficient, consistent, and high-quality microservice integration in the industry.

We employ a thematic analysis of two types of primary materials to construct our taxonomy: literature we collected in a systematic

* Corresponding author.
E-mail address: georg.schwarz@fau.de (G.-D. Schwarz).

https://doi.org/10.1016/j.infsof.2025.107723
Received 12 April 2024; Received in revised form 10 March 2025; Accepted 11 March 2025
Available online 24 March 2025

literature search, and six expert interviews we conducted in a qualitative survey. Thematic analysis is a suitable method for progressively introducing structure into a field. The method and its application are thoroughly documented, and the supplementary materials transparently present the traceability of results to their evidence. Thematic analysis allows us to meticulously ground our theory in empirical evidence. We implemented a full cycle of theory building by incorporating literature and expert interviews to gain diverse viewpoints on the topic. The interviews, centered around microservice techniques as a means to overcome integration challenges, enhance the practical relevance of the findings.

We grouped integration techniques into categories based on common characteristics. Categories are arranged into higher-level categories, resulting in a hierarchy of categories that classify microservice integration techniques.

Based on our results, we claim the following contributions:

- a novel taxonomy of microservice integration techniques with five main and ten refined categories as a structured framework for understanding the diverse landscape of microservice integration techniques;
- the incorporation of different viewpoints going beyond technical solutions by encompassing organizational aspects of integration, which are often overlooked;
- the classification of 121 found integration techniques;
- an illustration of the taxonomy usage, classifying two example techniques for demonstration;
- a synthesis of data in the literature with practitioner interviews, highlighting areas that require further research;
- comprehensive supplementary materials [6] that allow browsing the 121 integration techniques to support practitioners in applying our findings.

Researchers can use this taxonomy to review existing techniques and propose new ones. It allows identifying related techniques to evaluate against them, compare them, and delimit from them. The taxonomy offers practitioners a comprehensive overview of the multifaceted challenges inherent in microservice-based projects. This proposed structure guides practitioners in navigating their learning process, including architectural, technical, organizational, and operational aspects of microservice-based architectures.

The study focuses purely on theory building, presenting the findings in an actionable and extensive way. An empirical evaluation of the taxonomy and its 121 techniques with practitioners or within an industrial context is beyond the scope of this study.

First, Section 2 describes our holistic and broad perspective on microservice integration. Section 3 reviews the related work and situates this article within the research field. Section 4 outlines the applied research design. Section 5 introduces the taxonomy as the main result of this article. Section 6 demonstrates the utility of the taxonomy by classifying two example integration techniques. Section 7 compares the findings in the literature with practitioner interviews and outlines future work. Section 8 reflects on the limitations of the applied research methods, and Section 9 concludes the article.

2. Microservice integration

One of our interviews highlights the high knowledge barrier for adopting microservices, not only for the developers but also for operations, management, and the business level:

> *"[...] What I would like to impart is the awareness of the consequences [of doing microservices] and of the people's required education with regard to the challenges of distributed systems on management, business, and also IT level. In my opinion, this happens far too rarely".*
>
> [Interviewee D, translated from German]

The supposedly pure architectural decision of adopting the microservices architectural style may come with more challenges than one might expect, especially on the organizational level. Thus, we aim to give a more holistic perspective on the topic of microservices and their integration.

Integration in software systems is not a new phenomenon that affects software engineering in various forms. Nierstrasz and Dami [7] trace back the idea of component-oriented development back to the first developments of structured programming and modularity. They define a component as a "static abstraction with plugs" encapsulating a piece of software. Components can be interconnected and integrated to compose an application.

In software engineering, components can be composed during the different phases, such as design, deployment, and runtime [8]. While monolithic applications foster integration during design and deployment, distributed architectures like service-oriented architecture (SOA) and microservices rely on the composition at runtime.

In SOA, a predecessor of microservices, integration is a common theme throughout the whole architecture, reaching from the technical interconnection of services to integrating business workflow processes within the architecture. A central part enabling the integration within those systems is a pattern called the enterprise service bus (ESB) [9]. The ESB is responsible for message routing and message transformations for interoperability and, thus, becomes the central place for integration in SOA [10]. Microservices, in comparison, facilitate integration through lightweight messaging approaches.

Aside from these architectural perspectives, integration itself is a term that has manifold meanings. In the area of information systems, different perspectives on integration are considered. Barki and Pinsonneault [11] describe the technical interconnection between information systems talking to each other, the coupling of business processes of independent organizations by IT, and coordination and cooperation among project teams.

In this article, for the sake of clarification, we base our definition of integration on Mohamed et al. [12]: *"integration per se has been found to be a socio-technical phenomenon beyond a mere technological aspect such that it includes an assortment of economical, organizational, and even social facets of the phenomenon"*. In the context of microservices, **we define integration as an enabling factor for interactions among software components**. We chose this wide definition of integration to emphasize the openness that drove our research to incorporate various viewpoints on the topic. Even though our definition targets the interaction of software components, the enabling factors can go beyond the software component scope. For example, the use of specific types of software, design policies, and methodologies can be factors that enable and simplify component interaction. Further, the development of software is a social activity, and so is the integration of software components. Thus, we view organizational structure, coordination processes, and management activities between organizational entities within the scope of our research if they transitively enable the interaction of software components. This definition encompasses horizontal integration across various organizational units at different levels, aligning with the viewpoint proposed by Hasselbring [13] while also accommodating considerations of integration within a single organizational unit or with external entities. We deliberately chose such a broad viewpoint on the integration topic to draw a holistic picture of the field.

3. Related work

Taxonomies have been proposed in many software engineering areas to structure and better understand the body of knowledge. Usman et al. [5] provides a comprehensive overview of the current taxonomy literature in software engineering, analyzing 270 articles. The examined taxonomies can be categorized into knowledge areas (based on SWEBOK [14]), such as construction, design, requirements, and maintenance. Microservice integration falls under the software

design category. While software engineering taxonomies like Keshav and Gamble [15] or Hofmeister and Wirtz [16] cover general architecture and integration patterns, there is no dedicated taxonomy covering the multi-dimensional facets of microservice integration. In contrast, our taxonomy provides an orthogonal angle to the proposed dimensions in the context of microservice-based projects, as our holistic understanding of integration covers multiple areas, such as design, maintenance, processes, and configuration management. Further, most studies construct taxonomies in an ad-hoc manner [5], whereas we employ thematic analysis [17], an iterative method to uncover patterns of meaning.

Based on the broad definition of microservice integration used in this article, numerous studies have addressed the topic to some extent. For example, Bogner and Zimmermann [18] investigates mechanisms to integrate microservice-based architectures by using meta-models derived from enterprise architecture reference models. Petrasch [19] explore a UML-based approach to the use of enterprise integration patterns for inter-service communication. Shafabakhsh et al. [20] compare synchronous and asynchronous communication among microservices and their implementations with respect to efficiency and availability. In contrast, our study aims to synthesize different angles of integration aspects and construct a taxonomy of integration techniques instead of focusing on a specific approach.

In this regard, all microservice literature that presents patterns from different viewpoints is relevant to our research. In our understanding, patterns are often used in combination with pattern languages or relate to the many other things that the patterns community, separately from the scientific community, performs. Thus, we call the actionable parts of our theory techniques to avoid implying we created a pattern language or performed the pattern-community-specific procedures. Nevertheless, patterns aim to overcome challenges as we do as well and, thus, are related to our work.

Balalaie et al. [21], for example, present migration patterns, Harms et al. [22] present patterns related to front-ends. Osses et al. [23] present further architectural patterns and tactics. While this list goes on, many of these papers propose and evaluate concrete microservice techniques for a subset of the problems inherent to microservice architectures. In contrast, this article offers a broader perspective by presenting a structure that takes a larger context into account.

Previous research has explored the topic of introducing a certain structure, but most studies have focused on narrow or technical aspects. For example, Fritzsch et al. [24] proposed a classification of techniques to decompose a monolith. However, our study takes a broader approach by examining both operational and organizational aspects of introducing such a structure. Weerasinghe and Perera [25] categorize patterns into service decomposition patterns, data management patterns, deployment patterns, API-based patterns, service discovery patterns, and resilience patterns. Márquez and Astudillo [26] present categories for communication, orchestration, deployment, and backend patterns while evaluating their use in open source projects. Taibi et al. [27] presents a pattern catalog with the major categories of deployment patterns, data storage patterns, and patterns for orchestration and coordination. Söylemez et al. [28] propose a structure for the challenges facing microservice-based architectures. This structure covers important aspects such as service discovery, testing, communication, integration, performance prediction, measurement, optimization, service orchestration, monitoring, tracing, and logging. While integration is one of their categories, we provide a broader perspective on this topic by not focusing only on technical and architectural aspects but also viewing the integration among microservice teams as organizational aspects as a success factor for microservice-based projects.

Building on these efforts to structure the field of microservice techniques, several articles have incorporated operational and organizational aspects. Taibi et al. [29] present a taxonomy of anti-patterns that includes technical and organizational issues, which can be interpreted as techniques for addressing those issues. For example, the anti-pattern “No API Gateway” can be reframed to a technique “Use API Gateway”. They distinguish technical and organizational anti-patterns. Technical anti-patterns can either focus on a single microservice, the communication among them, or fall into the bucket “others”. Organizational anti-patterns are either categorized as team-oriented ones or as technology and tool-oriented ones. We extend their work by providing a broader and deeper structure that reflects their categories as well. While the anti-pattern format points out the pitfalls to avoid, we present techniques on how to approach challenges in a context.

Brown and Woolf [30] present patterns and order them by their origin into categories modern web architecture patterns, microservices architecture patterns, scalable store patterns, microservices DevOps patterns. Looking at the origins of a pattern reveals valuable insights, which we aim to further by offering a comprehensive structure for applying these patterns in practice.

Osses et al. [31] presents in a poster a taxonomy of microservice patterns. Their major categories, DevOps, migration, design, mitigation, IoT, frontend, deployment, backend, communication, behavior, and orchestration, cover a broad field of integration aspects. We provide a deeper and stricter structure from a different viewpoint.

Alshuqayran et al. [32] present challenges in microservice architectures in a structured way and connect them to solutions they found in the literature. Their distinction of requirements, design, implementation, testing, deployment, monitoring, organizational problems, and resource management problems provides a suitable structure to organize the field. While they take the DevOps perspective in microservice-based projects, we take the integration perspective, leading to a different structure. In our opinion, both perspectives are very valuable and actionable when it comes to finding a solution to a certain problem.

From a research design perspective, we position this article as an analysis of literature and expert interviews, while most presented related work focuses on academic literature and case studies. We implemented a full cycle of theory building by incorporating literature and expert interviews to gain a diverse sample of viewpoints on the topic. We performed a thematic analysis of the data, a method suited to iteratively introduce structure into a field.

4. Research design

The objective of this work is to organize the field of microservice integration by constructing a taxonomy for microservice integration techniques. From this objective, we derive the following research question:

RQ1: How can microservice integration techniques be classified?

To address the research question, we conducted a systematic literature search [33], conducted expert interviews in a qualitative survey [34], and analyzed both types of primary materials with a thematic analysis described by Clarke et al. [17].

In thematic analysis, the researcher annotates excerpts of the primary materials with codes to capture patterns of meaning within qualitative data. These codes are then arranged in a hierarchical structure. The method does not lead to taxonomies per se. To build a taxonomy, we deliberately built categories by employing clear heuristics per hierarchy level to assess to which category lower-order elements belong. Due to this procedure, a taxonomy with distinctive categories emerged that allows the classification of the found patterns of meaning, the microservice integration techniques.

Due to the study’s objective of providing an academic and an industrial contribution, we applied thematic analysis in multiple iterations on primary materials from literature and interviews with industrial practitioners. Fig. 1 gives an overview of the overall research design. In the continuation of this section, Section 4.1 describes the literature selection procedure, Section 4.2 elaborates on the design and execution of the interviews, and Section 4.3 details the thematic analysis we applied to construct the taxonomy.

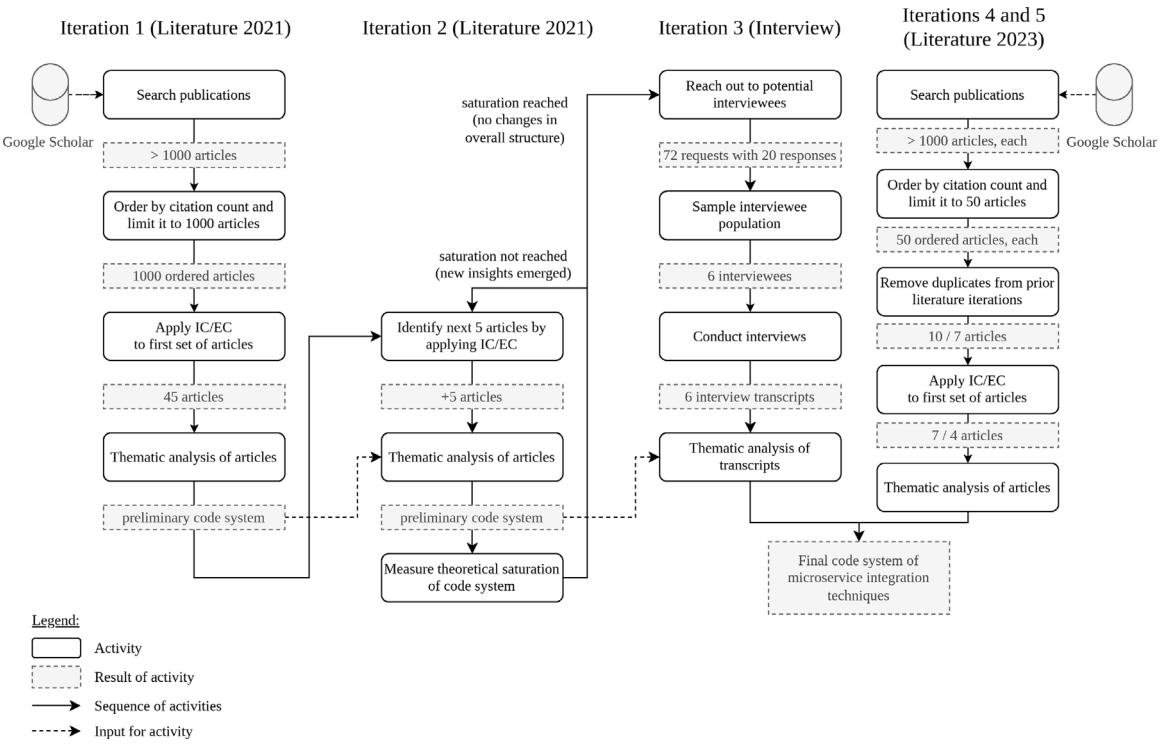


Fig. 1. Summary of the research approach.

4.1. Literature selection

We selected white literature as the primary material for the thematic analysis. We partially followed the guidelines of Kitchenham [33] to systematically select the literature. As our goal is to construct a taxonomy of microservice integration techniques and not to give an overview of the research area, we deviated from the selection process of a systematic literature review where it made sense. We will rigorously lay out these deviations in the following paragraphs.

We planned the literature selection before executing the data collection and created a research protocol for the data collection. The following paragraphs summarize the contents of this research protocol with the main points of the search strategy and the study selection process. On the path to the final procedure, we conducted an ad-hoc pilot search, allowing us to learn how the search terms and search engines influence the literature selection and simplify the process.

Search strategy

In an ad-hoc pilot search, we used Google Scholar, ACM Digital Library, and IEEE Xplore as academic search engines to evaluate the search term, the inclusion, and the exclusion criteria before starting the actual literature selection. Google Scholar covered the results of the other search engines together with additional published venues. Thus, we decided to simplify our final search process to solely use Google Scholar.[1] As Google Scholar includes articles in academic magazines and practitioner books, we applied additional filter steps to stick to academic peer-reviewed articles (see study selection).

We defined our search term as a combination of "microservice" and "integration" in different writing styles. Further, we considered using similar terms like "interoperability" but decided against it. While integration is a broader concept, interoperability has a technical connotation that could bias the results favoring technical-focused topics. To include relevant articles, we chose the title to match our search query. The logical search query is as follows:

("microservice" OR "microservices" OR "micro-service" OR "micro-services" OR "micro service" OR "micro services")[title] AND ("integration" OR "integrate")

Selection strategy

Due to the expected number of articles, we ordered them by citation quantity. This order criteria favors well-established and adopted literature capturing the common sense of the broader population of interest in the domain.

To ensure that each analyzed article was relevant, we only considered articles that met all the inclusion criteria (IC) and none of the exclusion criteria (EC):

- IC1: The article must have been published between 2014, when the term "microservices" first emerged [35], and March 2023, when we conducted the literature selection.
- IC2: The article must be peer-reviewed academic literature published at a journal, conference, or workshop.
- IC3: The article must address microservice integration challenges or present microservice integration techniques (implicitly or explicitly).
- IC4: The article must be accessible in full-text.
- IC5: The article must be available in English.
- EC1: The article is a talk abstract.

Defining a stopping criterion rather than exhaustively analyzing all articles is suggested by Garousi et al. [36] when the relevant pool of literature is too large. This method is appropriate in our case as well because the amount of considered literature due to Google Scholar was too large, similar to their use-case on multi-vocal literature reviews where the amount of gray literature is overwhelming. Thus, we combined an effort-bounded stopping criterion (starting with 45 articles) with theoretical saturation (measure changes in sets of five articles).

While this approach might not reveal all existing microservice integration techniques, it allows us to address the research question

[1] We used the tool *Publish or Perish* to automate the search on Google Scholar.

by constructing a taxonomy of integration techniques with an adequate trade-off between effort and accuracy. Using only peer-reviewed literature ordered by citation count ensures that only literature approved by the microservice research community is included, favoring well-established literature. This metric ensures an adequate trade-off between relevance and rigor.

Literature selection execution

We iteratively added articles to our literature pool and started analyzing the data in parallel. In the first iteration, we started with 45 articles that met the inclusion and exclusion criteria. Then, we added sets of five articles that passed the inclusion and exclusion criteria. We tracked the changes in the code system of the set and used it to measure saturation.

The portrayed literature selection was conducted in January 2021 (literature ids prefixed with "L"). In order to ensure that the analysis is based on the most recent literature, the literature was repeated in March 2023 (literature ids prefixed with "LN") and in July 2024 (literature ids prefixed with "LM"). We considered the first 50 articles, ordered by citation count, leading to 10 newly considered articles, where 7 passed the inclusion and exclusion criteria. Since we analyzed these articles after analyzing the interviews, we used this opportunity to evaluate the code system by tracking all changes. We found two new techniques, refactored four techniques, and tracked six further changes related to loose collections of codings, e.g., where evidence was missing to turn them into a full-blown technique. In the process of writing this paper, we renamed two refined categories to clarify their intent. The overall structure of the taxonomy did not change, confirming the theoretical saturation measured in previous iterations.

See the supplementary materials for details on the selected literature.

Data analysis

To construct a preliminary taxonomy based on the existing literature, we employed thematic analysis as defined by Clarke et al. [17]. We then complemented our findings from the literature with expert interviews. A detailed description of the analysis process using thematic analysis of both literature and interviews can be found in Section 4.3.

Microservices have emerged in the industry and are still a fast-moving topic. The interviews allow us to capture the state-of-the-practice challenges and latest techniques of microservice integration. Performing thematic analysis on these additional primary materials led to an extension: the final taxonomy. We use this separation of the preliminary and final taxonomy in Section 7 to compare the findings in literature with the interview data to identify gaps in research and deviating emphases in practice.

4.2. Expert interviews

In addition to the literature review, we conducted six expert interviews to triangulate our findings from the literature. With data triangulation, different sources of information are used to increase the validity of the study's results [37]. This triangulation allows us to improve our taxonomy's quality and emphasize the latest insights from the industry. For these interviews, we followed the qualitative survey approach defined by Jansen [34].

Interview preparation and guide

We followed the five phases presented by Kallio et al. [38] to prepare for the interviews, leading to our interview guide as an artifact.

Phase 1 Identifying prerequisites. We first evaluated the appropriateness of semi-structured interviews according to our research questions. Semi-structured interviews allow us to study different organizational contexts and different angles on the topic for a diverse perception and to discover topics that are especially relevant to practitioners.

Phase 2 Previous knowledge. The preceding systematic literature review resulting in a preliminary taxonomy led us to a comprehensive understanding of the domain to prepare and conduct semi-structured interviews. We utilized our insights to construct the interview guide.

Phase 3 Preliminary interview guide. We used previous knowledge to structure the interview into multiple phases. Each phase consists of questions that allow steering the interview in the direction of our area of interest but are flexible and loose enough to allow open conversation. We adopted questions for main themes and follow-up questions in the different phases of the interview. We started with the main themes as a warm-up to break the ice. Afterward, we used a mix of generic follow-up questions adapted to the interviewee's answers, prepared questions, and spontaneous ones to go into depth. We applied verbal and non-verbal probing techniques during the interviews but did not make them an explicit part of the interview guide.

Phase 4 Pilot testing. The interview guide was reviewed internally by members of our research group to avoid ambiguous or leading questions. We applied live field testing by reviewing the interview guide after the first interview, allowing for incremental improvements.

Phase 5 Presenting the interview guide. The supplementary materials contain the complete interview guide. The rough structure was as follows:

1. Preamble: Before starting the interview, we alleviate the tension the interviewee is potentially experiencing with some small talk. We agree on the language used during the interview according to their preference to provide the interviewee with a comfortable and natural environment. We explain the interview procedure and how we will handle the recording of the interview and all its information. If the interview partners consent, we begin the interview.
2. Warm-up questions: In order to warm up and overcome the last potential tension of the interviewee, we let them present themselves. We ask about when they started using microservices and what a microservice is from their perspective.
3. Definition microservice integration: We use open questions to explore what microservice integration means to the interviewees, why it is important to them, and what aspects of microservice integration are relevant for them in their projects. The goal is to reveal the main categories of integration techniques by speaking about integration in general.
4. Microservice integration techniques: We detail each aspect they mentioned in the previous part in an ad-hoc manner. We added semi-open and closed questions on integration aspects we had already discovered in literature and previous interviews to evaluate their relevance and further insights. The goal is to reveal integration techniques, to fill in the details for the ones we already discovered, and to resolve conflicts in the gathered data.
5. Cool-down: The interviewee can freely point to interesting topics in the microservice field we did not cover during the interview but should get more attention in research. Finally, we ask for recommendations for future potential interview partners for snowballing.

We sent the interview guide with additional notes to our interviewees before the interview. Understanding the context and scope of the interview allowed them to prepare thematically and mentally. We included the following information:

- The context of our research.
- The process of an interview (time frame, the way we ask questions).
- The data assessment process (audio recording, interview transcription).
- The approval process: we send out each interview transcription to the interviewee to correct errors and misunderstandings. Only after approval we use the interview transcription for further analysis.

- Data confidentiality and privacy (data pseudonymization for data analysis and anonymization for publications).

Sampling model

To select suitable interviewees, we first created a sampling model. It contains fine-grained categories towards factors we believe might have an influence on how microservice integration is facilitated: the interviewee's company, project, role, and experience.

As a quality measure, we asked an established expert in the field to provide feedback on our sampling model and find uncovered categories. We employed generic roles for the expert's role as each company might define its specific roles with more fine-grained responsibilities that would be hard to match against each other. On a higher level, we distinguish between in-house employees and consultants, as the latter tend to experience many different project contexts.

We translated our sampling model into a form to be sent out to potential interviewees. Using such a form allows efficient classification of each potential interviewee within the sampling model.

For interviewee selection, we aimed to cover each category within the sampling model adequately to achieve a diverse sample. The goal is to create a diverse selection of participants with multiple points of view, strengthening the breadth of the taxonomy. Additionally, we followed Taibi et al. [29] to only consider interviewees with at least two years of experience with microservices.

We followed the guidelines of Francis et al. [39] for an adequate sampling size:

1. Specify initial sampling size a priori: six interviews to cover each of the six expert roles at least once.
2. Specify stopping criteria a priori: add one interview until we did not discover new themes and did not make changes to the taxonomy.
3. Multiple researchers for analysis: Section 4.3, inter-coder reliability, peer debriefings.
4. Report data saturation methods: theoretical saturation by adding interviews until no changes to the structure of the taxonomy were made.

Interviewee sampling

We utilized our group's network, the mailing list of the working group for microservices and DevOps by the German Informatics Society. We contacted over 50 speakers at practitioner conferences like microxchg[2] or Microservice Summit.[3] We received 20 answers from willing interviewees, arranging them into our sampling model by filling out the form.

Table 1 presents the sampled population arranged in the major categories of the sampling model, showing the diversity of our sample. Please note that we did not receive answers from all participants for the project-related questions since one consultant felt uncomfortable limiting the focus of the interview to one specific project but instead elaborated on their experience regarding multiple project contexts.

Interview execution

We found the interview guide especially useful in the first phases to streamline our interviews. It supported us to stick to the semi-structured frame and avoid deviations from the topic of interest. While detailing microservice integration techniques, we used the guide as a checklist rather than sticking to it strictly, as interviewees tended to be very active and speak freely. In the cool-down phase, we experienced the open question on further interesting topics as especially valuable. Some answers led us back to phase 4 to investigate further integration topics.

After the interview, we transcribed the audio recording. The transcript was sent to the interviewee for review to detect misunderstandings and consider second thoughts on some of the insights they gave us. After their final approval, we added the interview transcripts as primary materials for analysis. As the code system had already been populated by the preceding literature analysis, we reached saturation after six interviews.

Data analysis

We extended the preliminary taxonomy based on the literature (Section 4.1) for data analysis. We added the interview transcripts to the primary materials for the thematic analysis (Section 4.3).

Although we did uncover new discussions and techniques, we did not make any changes to the overall structure of the code system. This confirms the findings from the preceding literature.

4.3. Thematic analysis for taxonomy construction

We applied thematic analysis, as described by Clarke et al. [17], to construct the taxonomy. Thematic analysis is an accessible and systematic research method and procedure to discover, analyze, and interpret patterns of meaning within qualitative data. The researcher takes an active role in generating codes from the qualitative data guided by the research question. Codes capture interesting features of the data that are relevant to the research question. Codes are aggregated into themes, representing patterns of meaning. Underlying is a central organizing concept for analytic observations.

In this study, codes are specific integration techniques, and themes are categories of integration techniques. Codes and themes together in a hierarchy build the code system.

Method choice

Our qualitative data analysis builds on a diverse set of data, like academic literature and expert interview transcripts. We considered grounded theory and thematic analysis as competing methodologies.

Grounded theory approaches, as described by Strauss and Corbin [40], act as a framework for generating theories from qualitative data. The approach is predominantly inductive with the goal of creating a theory purely from the data; prior in-depth familiarization with the topic is discouraged to avoid the researchers' prior knowledge influencing the results. Grounded theory follows a structured coding process. First, the researchers break down data into the initial codes in the *open coding* phase. Afterwards, they identify relationships among codes in the *axial coding* phase. Eventually, structured codes are refined into a central category that becomes the foundation of the theory in the *selective coding* phase.

In contrast, thematic analysis, as described by Clarke et al. [17], is a flexible data analysis method that focuses on identifying patterns of meaning within data rather than on generating a hierarchical theory. Thematic analysis also employs a more adaptable coding process. First, researchers generate initial codes, then search for themes, and then finally review and refine the themes; the full process is detailed below. This method supports both inductive and deductive approaches, either deriving themes directly from the data in a bottom-up fashion, similar to grounded theory, or in a top-down fashion, guided by research questions and prior knowledge. The output of thematic analysis is not necessarily a hierarchical, structured theory as in grounded theory; rather, it is a collection of themes that can nonetheless support theory building, offering the researcher considerable flexibility in interpretation and presentation.

The tradeoff described makes thematic analysis a more flexible option; however, it offers less guidance on other aspects of the research process, such as data collection and the development or presentation of a theory. We chose thematic analysis because it aligns better with our research question. Building on our previous research in the field of microservices allowed us to effectively combine deductive and inductive approaches rather than relying solely on grounded theory, which would require all theoretical constructs to emerge inductively from the data.

[2] https://microxchg.io/2020/index.html
[3] https://microservices-summit.de/

Table 1
Interview sampling.

Category	Feature	Interviewee					
		A	B	C	D	E	F
Expert role	High-level consultant		x			x	
	Detail-oriented consult				x		
	Architect						x
	Project manager			x			
	Developer						x
	Operator/DevOps	x					
	Other				x		
Project phase	Research & innovation			x			
	New software					x	
	Rewrite	x					
	Evolution		x				x
Project size	1 team						
	2-10 teams		x	x			x
	10+ teams	x				x	
Microservices	1-10 services			x			x
	11-50 services		x			x	
	50+ services	x					
Deployment	Customer-managed	x				x	x
	In-house	x					
	Cloud		x	x			

Analysis procedure

For the thematic analysis, we followed the six-step process defined by Braun and Clarke [41]:

Phase 1: Familiarize with the data. We read the primary material actively and noted the first coding ideas. We used the transcription process for the interviews as an excellent way to familiarize ourselves with the interview data.

Phase 2: Generate initial codes. We worked through the primary materials and annotated data segments with preliminary names. We coded as detailed as possible as time permitted and included the context in the coded text segments. For example, we used the codes "Data replication" and "Decentralize conceptual models" to annotate the following text snippet in Cerny et al. [42]: *"μServices usually do share the same database schema as it would predetermine a bottleneck as well as coupling. Each μService is in charge of its own data model, which possibly leads to replication".*

Phase 3: Search for themes. We took the long list of codes and considered how differently the codes may be combined. We created the potential themes by aggregating codes that seemed cohesive to us. We thought of relationships between codes and themes and arranged them in a hierarchy. For example, we categorized the code "Data replication" together with the code "Avoid transactions over multiple microservices" under a newly created theme "Dataflows" based on their commonality of describing data flow-related techniques among microservices. The theme "Dataflows" itself is further categorized in the hierarchy under the theme "Conceptual integration" because its techniques describe architectural and conceptual aspects.

Phase 4: Review the themes. We revisited the created themes and codes to reflect on how the individual themes represent the data set. We paid attention to clear distinction criteria of themes and discussed ambiguous ones. For example, we renamed the code "Orchestration vs. choreography" to "Choreography over orchestration" after coding it in multiple primary materials. Although the code first captured discussions about the topic, the code later represented a clear choice of one over the other. We were able to refine this code by bringing together multiple voices and narrowing the context in which this technique can be applied.

Phase 5: Define and name themes. Until now, themes had a working title. We went over each theme individually and identified what is of interest about them concerning the research question and why. We ensured that the themes were not too complex and too broad by utilizing sub-themes. Additionally, we explicitly put down the definition of each theme and criteria on when and when not to apply them. For example, we renamed the final theme "Among services" from a prior version "Between Services/Teams / Bounded Contexts (horizontal)" while transferring the results into this article. Further, we added a memo to the theme to document when it is used and when not. Here, we coded text segments expressing the general need to bridge the gap between bounded contexts and the need of microservices to cooperate (not necessarily communicate) to form the whole system behavior. We did not apply the code to where the text was related to integration within one bounded context or microservice or when specific sub-codes fit better or more precisely.

Moving back and forth between the phases may be necessary as researchers gain new insights during coding and building themes [41].

Table 2
Iterations of analysis.

#	Analyzed materials
IA1	45 articles from literature selection 2021
IA2	5 articles from literature selection 2021
IA3	6 interview transcripts
IA4	7 articles from literature selection 2023
IA5	4 articles from literature selection 2024

Execution of analysis

As primary material for thematic analysis, we used white literature and interviewed practitioners. Table 2 gives an overview of the conducted iterations. First, we conducted two iterations of analysis (IA) using selected literature as primary materials. The first iteration (IA1) reflects the analysis of the effort-bounded stopping criterion of the literature selection by starting with 45 articles. The second iteration (IA2) added a set of five articles. There were no major changes to the structure of the code system within this iteration, so we reached theoretical saturation and decided not to add further literature-based iterations. Instead, we analyzed the interview transcripts in the third iteration (IA3) to triangulate our findings by using a different type of primary materials. The last two iterations (IA4, IA5) were conducted to update the literature selection over the course of two years. They did not yield new insights but were necessary to incorporate the latest relevant literature.

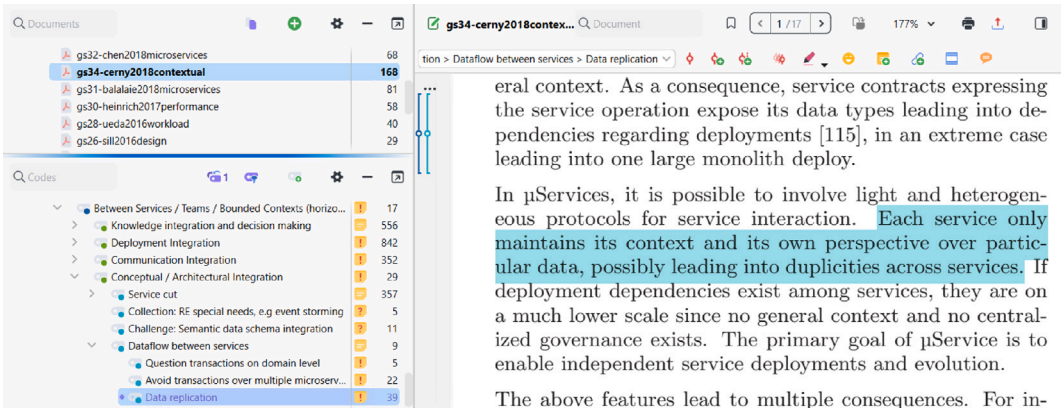


Fig. 2. MaxQDA example: Text segment coded as "Data replication".

Table 3
Inter-coder reliability feedback sessions with the number of requested changes for each session.

Feedback	Session		
	#1	#2	#3
Missing themes	3	4	0
Themes to refine	3	3	0
Themes to re-categorize	5	0	0
Other comments	13	5	3
Sum	**24**	**12**	**3**

To ensure the traceability of emerging codes and themes to their sources, we used the software MaxQDA[4] to support our coding and theme-building process. A screenshot demonstrating our use of MaxQDA is shown in Fig. 2. We created a hierarchy with over 199 themes and codes and over 2400 coded segments. The high-level themes of this hierarchy represent a taxonomy that addresses RQ1, while low-level themes represent the integration techniques we found. We attached a memo to each code summarizing its theme in prose and specifying when the code is not applied. The codes, together with these memos, are called the codebook.

Inter-coder reliability

We conducted three inter-coder reliability sessions to improve the quality and validity of our emerging codebook. Depending on the session, one or two fellow researchers applied the existing themes and codes to parts of the uncoded primary materials supported by the codebook entries. The themes, codes, and codebook quality were evaluated qualitatively by comparing the result with the original coding. The inter-coders took notes of missing themes, themes that need refinement (renaming, redefinition), re-categorization within the themes, and other comments. We discussed the notes jointly afterward to define improvements to the themes and the codebook. Table 3 summarizes the inter-coder reliability sessions. We see a reduction of change proposals indicating the maturity of the results. The same trend manifested in the qualitative comparison of the coding with the original. On the one hand, we chose inter-coders experienced in the software architecture field to perform investigator triangulation. On the other hand, we chose inter-coders from outside the field to perform theory triangulation by having multiple perspectives from different disciplines interpret the data [37].

5. Results

We present a hierarchical taxonomy that expresses the diversity of integration aspects covered by microservices and their interrelations. Following our definition of integration, we define an integration technique as an abstract and reusable solution to a recurring integration problem in the microservice domain. These techniques may involve architectural, technical, operational, organizational, cultural, economic, procedural, or cross-contextual implementation. We have categorized these techniques into different categories for easier comprehension.

A taxonomy systematically assigns subject matter instances to categories [5]. Categories can further be grouped into higher-level categories, resulting in a hierarchy of categories.

Fig. 3 presents the taxonomy of microservice integration techniques. The left part of the figure shows the main categories as a tree with relations between higher-level parent nodes and lower-level child nodes expressing a specialization relationship. Child categories of a parent are mutually exclusive (ME), meaning an integration technique can only be categorized as one of the child categories. Child categories are collectively exhaustive (CE), meaning every technique is assignable to at least one of the child categories. Combining both characteristics (MECE) implies that each integration technique is classifiable by exactly one leaf main category (a category without child categories). We can apply a top-down process for taxonomy usage to classify a concrete integration technique.

Inside each main category, we further present refined categories of techniques. We cannot claim collective exhaustiveness and mutual exclusiveness for those refined categories as undiscovered or new techniques could exist that do not fit in one of these categories. Still, the refined categories introduce a particular structure for the integration techniques that we do not want to withhold from the reader. Each technique is classifiable as none, one, or more refined categories within its main category.

In the remainder of this section, we will discuss the taxonomy categories in detail, beginning with the main categories and their criteria of separation, followed by the refined categories. As an example of using the taxonomy, we mention the identified techniques in each refined category. A detailed discussion of each identified technique does not fit the scope of this study and could be the subject of further studies.

[4] https://www.maxqda.com/

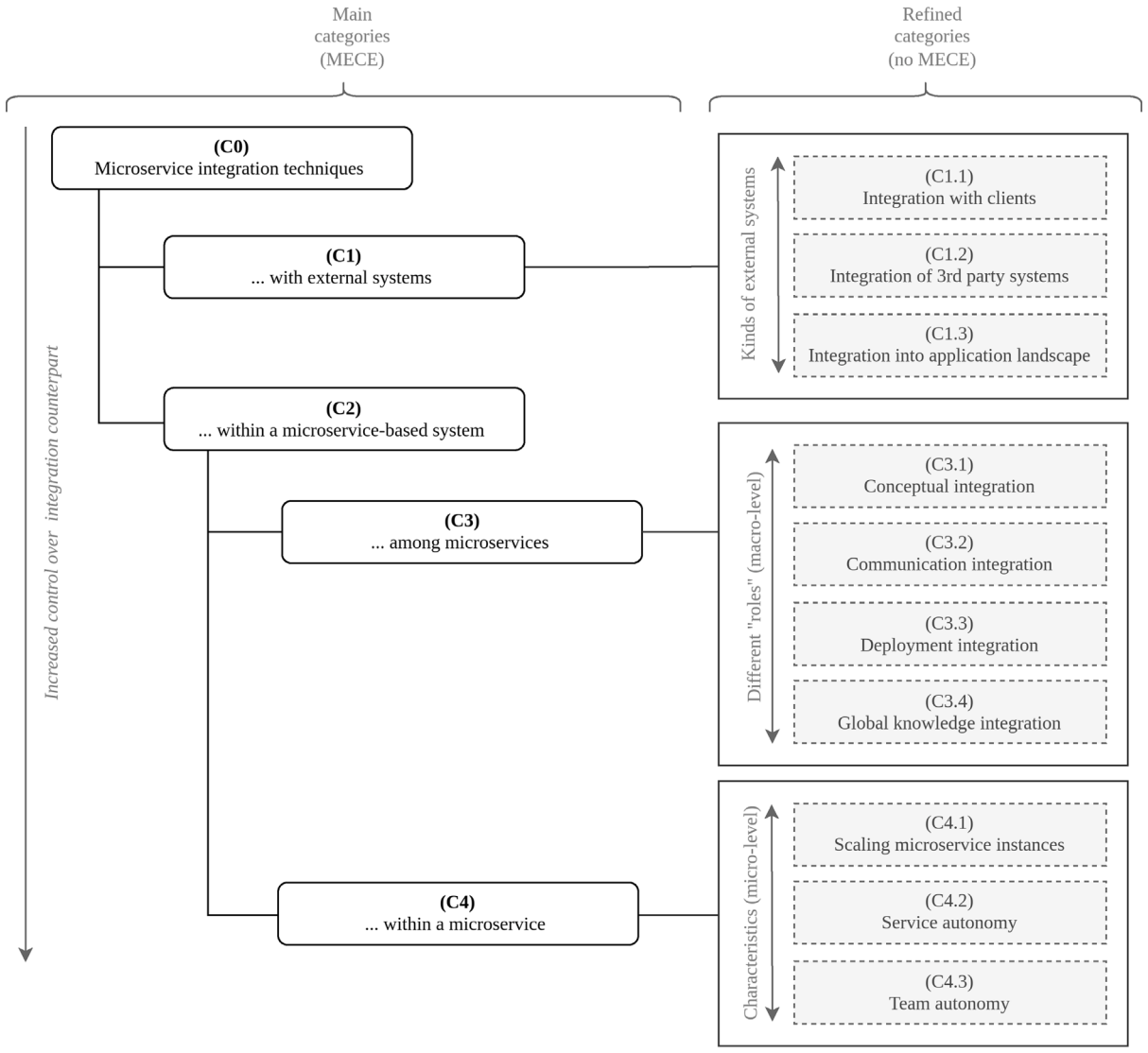


Fig. 3. The taxonomy of current microservice integration techniques. (MECE = mutually exclusive and collectively exhaustive).

5.1. Main categories

The main categories of the taxonomy are C0 to C4. The level of control over the integration counterpart emerged as the distinguishing factor of the main categories in the taxonomy during the coding procedure. The level of control over the integration counterpart is transitively linked with the innovation speed of a project, which is one of the main motivations for using the microservice architectural style [43,44].

Certain types of interactions between computer systems bear a resemblance to interactions and coordination among people [13]. The higher the control over the integration counterpart, the lower the effort to coordinate, facilitate, and maintain integration. The lower these efforts, the more time can be spent on feature development, leading to a higher innovation speed and agility of the overall project. Organizing techniques into categories by the control over the integration counterpart as the distinguishing factor allows one to quickly navigate in the taxonomy since this criterion is assessable at first glance.

In this subsection, we present all main categories with a description of the category and a rationale outlining the distinction from other categories by the rating of the control over the integration counterpart (low, medium, high). The subsections after will detail the refined categories of the leaf main categories (C1, C3, C4).

(C0) Microservice Integration Techniques
Description: Integration techniques offer practical solutions to recurring problems in specific contexts. They encompass not only architectural and technical issues but also operational and organizational challenges since integration is a socio-technical phenomenon that extends beyond technicalities [12].
Rationale: This is the root category of the taxonomy. The level of control over the integration counterpart is inherited (low to high) from the child categories C1 and C2.

(C1) Integration with External Systems
Description: These techniques target integration with external systems outside of the microservice-based system. Examples are third-party

systems or their APIs used by the application or clients that use the API of the microservice-based system.
Rationale: The control over the integration counterpart is low since external systems are not under the control of the project teams. Changes in external systems come with a significant communication overhead, if possible at all. *"[Many third-party systems] are quite closed off. At best, I may be able to gain access to some sort of data schema. If I am fortunate, there may be a rudimentary input–output table or some mechanism through which I can manipulate the data. However, what is the level of availability of these mechanisms? Most are not available 24/7 or any sort of guarantee"* [Interview D, translated from German]. Special measures might be required to ensure successful integration between the microservice-based system and the external systems since they do not necessarily keep up with the level of fault tolerance, scaling, and knowledge about the internal structure of the components within the system.

(C2) Integration Within a Microservice-based System
Description: These techniques are applied for integration within the system. Concerned software components within a microservice-based system are usually maintained by the teams of the project.
Rationale: The control over the integration counterpart is inherited (medium to high) from the child categories C3 and C4. We logically infer that the coordination effort within a project is significantly lower than with an external system.

(C3) Integration Among Microservices
Description: These techniques apply to the integration among microservices within the system and their responsible teams. Each microservice is owned by its responsible team [45]. Thus, integration among microservices requires coordination among their responsible teams.
Rationale: The control over the integration counterpart is medium since it depends on another team that owns the microservice. Coordination and negotiation are required to change or implement some functionality [1,46].

There is a potential difference in the amount of coordination overhead among teams depending on whether the microservice is under the control of an in-house team or a supplier. Working with subcontractors might pose an additional coordination overhead because *"[...] you need to adhere to certain communication protocols [...]. This means that it is not always possible to take the most direct route in communication"* [Interview F, translated from German].

(C4) Integration Within a Microservice
Description: These techniques apply to the integration within a microservice and its responsible team. Each microservice is a silo from top to bottom, potentially internally integrating with database(s), user interface(s), and further components [47].

Scaling is facilitated horizontally by deploying multiple instances, introducing the challenge to coordinate these instances [48].
Rationale: The control over the integration counterpart is high since the microservice's team is fully responsible for all its components. Integration within a microservice benefits from the considerably lower communication effort within a single team than coordinating with other microservices teams [1,45,49]. Communication channels within a team can be more informal and adhoc, and are well exercised: *"[...] I can't imagine how we'd organize ourselves without a chat system. The cross-functionality and the inherently fast, efficient, and constructive communication is key. Email distribution would have slowed us down during implementation; it's an outdated method of communication"* [Interview A, translated from German].

5.2. (C1.1–C1.3) Refined categories of integration techniques with external systems

It is common in practice that microservice-based systems integrate with external systems that are not under the project team's control. Some might offer configuration options or even plugin systems. Others, however, might not be extensible. Changes in external systems are usually very costly if bought in, if possible in the first place.

In this main category of integration, we present refined categories of integration techniques distinguished by the usage types of or by the integration counterpart.

(C1.1) Integration With Clients
Description: These techniques foster integration with clients running outside of the system.

We distinguish clients developed by a third party and in-house ones. By definition, third-party clients are not under the control of the project's teams at design and run time. In-house clients might be under control at design time but run on machines outside of the control of the project's teams, e.g., in the browser of a user. Thus, the control at runtime is limited.
Rationale: The distinguishing usage type is service provision to the external system. The client application consumes the APIs of microservice-based applications to provide its service.
Techniques (5): *(T1.1.1)* API facade, *(T1.1.2)* Edge server facade, *(T1.1.3)* API gateway facade, *(T1.1.4)* API facade per client type, *(T1.1.5)* Independent choice of communication technology

(C1.2) Integration of 3rd-party Systems Into the Application
Description: These techniques foster integration with third-party systems running outside of the system, e.g., by using their API (see [50,51] for two example systems).

We also classify the legacy system in a migration scenario as such a third-party system as we noticed that the techniques for both highly correlate. The reason might be the reduced disposition to introduce significant changes for integration to a system that will vanish over the course of the migration.
Rationale: The distinguishing usage type is the service consumption of the external system. The microservice-based application majorly uses the third-party or legacy system to provide its service.
Techniques (6): *(T1.2.1)* Proxy microservice, *(T1.2.2)* Data replication proxy, *(T1.2.3)* CQRS proxy, *(T1.2.4)* Gradually replace the legacy system, *(T1.2.5)* Treat legacy system like a microservice, *(T1.2.6)* ESB to decouple from legacy system

(C1.3) Integration Into an Application Landscape
Description: These techniques concern integrating the whole microservice-based system or parts of it into a wider organizational context.

The surrounding application landscape might, for example, reuse and combine certain microservices in other projects [52].
Rationale: The distinguishing usage type is service reuse within an organization. Other applications might use parts of the microservice-based system in different projects. Lu et al. [53], for example, describe their vision of a "supermarket" of microservice in their IoT context allowing to simplify this reuse.
Techniques (4): *(T1.3.1)* Service/API registry, *(T1.3.2)* Document microservice metadata, *(T1.3.3)* Enterprise-wide standardization, *(T1.3.4)* Enterprise service wrapper

While the refined categories may sound like they are related to architecture and technical aspects, they also encompass operational and organizational aspects. For example, we need to make services discoverable for the rest of the organization with a service or API registry to allow the reuse of existing microservices in other projects, a mainly organizational challenge [53,54].

5.3. (C3.1–C3.4) Refined categories of integration techniques among microservices

Within a microservice-based system, collaboration is needed on the overall project level. Since each microservice is usually managed by one dedicated team, collaboration on an overall project level means collaboration between different teams. Integration with microservices managed by other teams leads to additional coordination efforts.

In this main category, we present refined categories of integration techniques distinguished by the stakeholder roles covering technical, architectural, operational, and organizational integration aspects. For better readability, we use sub-categories that organize the techniques by the rough topic they address.

(C3.1) Conceptual Integration

Description: These techniques cover the conceptual and architectural integration design among microservices.

Soldani et al. [48] describes the service cut, splitting the overall application into the microservices, as the primary pain of microservices at design time. Inferring from our experience, the service cut shapes the whole architectural landscape that needs to cope with the trade-offs made by the service cut.

Rationale: The distinguishing stakeholder role of this category is the software architect. The main focus is planning from a macro-level perspective and incorporating cross-cutting concerns like user authentication.

Techniques (30):

Service cut: *(T3.1.1)* Evaluate cut with proof of concepts, *(T3.1.2)* Avoid LoC metric for evaluation, *(T3.1.3)* Decentralize the service cut, *(T3.1.4)* Cut by non-functional characteristics, *(T3.1.5)* Cut by functional proximity, *(T3.1.6)* Cut by Domain-Driven Design, *(T3.1.7)* Cut by data entities and consistency needs, *(T3.1.8)* Cut by use-case, *(T3.1.9)* Cut by data-flow;

Dataflows: *(T3.1.10)* Question transactions on domain level, *(T3.1.11)* Avoid transactions over multiple microservices, *(T3.1.12)* Data replication;

Workflows: *(T3.1.13)* Choreography over orchestration, *(T3.1.14)* Align synchronicity to business flow;

Storage management: *(T3.1.15)* Decentralize conceptual models, *(T3.1.16)* Clear responsibilities for parts of the data;

Location of business logic: *(T3.1.17)* No domain logic into infrastructure, *(T3.1.18)* No sharing of domain-specific code;

User auth: *(T3.1.19)* Centralized SSO, *(T3.1.20)* Token-based authentication, *(T3.1.21)* Propagate security context via headers, *(T3.1.22)* Propagate security context via tokens;

UI integration: *(T3.1.23)* Only share context information between UIs, *(T3.1.24)* UI as part of each microservice, *(T3.1.25)* UI suites, *(T3.1.26)* Micro-frontends;

Conceptual error handling: *(T3.1.27)* Design for failure, *(T3.1.28)* Compensations in workflows, *(T3.1.29)* Degradation of functionality, *(T3.1.30)* Domain-motivated alternatives;

(C3.2) Communication Integration

Description: These techniques cover integration among microservices on the technical level.

Microservices interact with each other solely through their published APIs [55]. In general, integration counterparts need to share an understanding of the syntax and semantics of the exchanged messages to avoid data representation and schema mismatches [35].

Rationale: The distinguishing stakeholder role of this category is the software developer. The main focus is successfully and securely facilitating communication using APIs.

Techniques (15):

General: *(T3.2.1)* Align technical communication style to the nature of the business process

Communication security: *(T3.2.2)* Service-to-service authentication, *(T3.2.3)* Encrypt service-to-service communication;

API contracts: *(T3.2.4)* Use APIs to decouple from implementation details, *(T3.2.5)* Resilient consumers, *(T3.2.6)* Backward-compatible APIs, *(T3.2.7)* Hypermedia to reduce coupling, *(T3.2.8)* API versioning, *(T3.2.9)* Consumer-driven contract testing;

Communication error handling: *(T3.2.10)* Circuit breaker and fail fast, *(T3.2.11)* Dead letter queue, *(T3.2.12)* Bulkheads, *(T3.2.13)* Timeouts, *(T3.2.14)* Bounded retries, *(T3.2.15)* Domain-motivated implementation details;

(C3.3) Deployment Integration

Description: These techniques address the integration of deployed microservice instances into their runtime environment on the operational level.

Deployment automation is the driving key success factor to achieve innovation agility and reliability of the overall system in the face of the multitude of microservices and their instances at runtime [56].

Rationale: The distinguishing stakeholder role of this category is the software operator. The main focus is automating and optimizing the deployment.

Techniques (24):

General: *(T3.3.1)* CI/CD for automated deployment, *(T3.3.2)* Immutable deployments, *(T3.3.3)* Reduce deployment coordination, *(T3.3.4)* Sidecars/service meshes;

Service configuration: *(T3.3.5)* Avoid hardcoded configurations, *(T3.3.6)* Avoid default values, *(T3.3.7)* Environment variables for configuration, *(T3.3.8)* Configuration server for configuration, *(T3.3.9)* Configuration/deployment as code, *(T3.3.10)* Internal integration proxy to reduce coupling, *(T3.3.11)* DNS for routing, *(T3.3.12)* Service instance discovery, *(T3.3.13)* Service instance discovery by message broker;

Deployment environments: *(T3.3.14)* Virtualize the network, *(T3.3.15)* Offer single-node deployment, *(T3.3.16)* Provide resources as a service (Cloud), *(T3.3.17)* FaaS/serverless platform to abstract infrastructure, *(T3.3.18)* Cluster management by container orchestrator;

Zero-downtime deployment: *(T3.3.19)* Rollbacks, *(T3.3.20)* Rolling updates, *(T3.3.21)* Canary releases, *(T3.3.22)* Blue–green deployments,

Deployment artifacts: *(T3.3.23)* Containers as portable deployment artifacts, *(T3.3.24)* Artifact registry;

(C3.4) Global Knowledge Integration

Description: These techniques address integrating information between all roles and teams of the microservice-based project to foster global decision-making.

Di Francesco et al. [57] present sharing knowledge and effective communication as one of the core challenges during the phase of finalizing, implementing, and deploying the microservice design in a migration scenario.

Rationale: The distinguishing stakeholder role of this category is the project manager. The main focus is adopting tools and processes to efficiently communicate and make decisions.

Techniques (23):

Understanding the system: *(T3.4.1)* Standardize location of microservice documentation, *(T3.4.2)* Responsibility documentation, *(T3.4.3)* Standardize API documentation;

Organizational structure: *(T3.4.4)* Microservice managed by one team, *(T3.4.5)* Align architecture with org structure, *(T3.4.6)* Overarching organizational framework, *(T3.4.7)* Push more responsibility to teams, *(T3.4.8)* Group services based on domain proximity;

Coordination between teams: *(T3.4.9)* Establish a common vocabulary, *(T3.4.10)* Establish common cultural values, *(T3.4.11)* Standardization, *(T3.4.12)* Adhoc over formal communication, *(T3.4.13)* Regular cross-team discussions, *(T3.4.14)* Thematic boards for decision making, *(T3.4.15)* Service templates, *(T3.4.16)* Collaborate on libraries, *(T3.4.17)* Communicate API changes;

Understanding the system's behavior: *(T3.4.18)* Standardize logging/monitoring / tracing, *(T3.4.19)* Aggregate logging/monitoring information in a central place, *(T3.4.20)* Monitor metrics at different levels, *(T3.4.21)* Use dashboards and visualizations, *(T3.4.22)* Use a tracing mechanism, *(T3.4.23)* Automate anomaly detection and alerting;

By separating refined categories by role, the architectural, technical, operational, and organizational topics are easy to spot. The MECE criteria do not apply here as challenges often cross-cut the boundaries of the roles. We recommend assigning a major refined category representing the most prominent perspective taken in the technique.

5.4. (C4.1–C4.3) refined categories of integration techniques within a microservice

There is a need for integration within a microservice and its team. Integrating on this level is required to ensure the quality attributes of the specific microservice, such as scalability or service autonomy. The integration counterpart is usually under the team's direct control, so the integration effort in terms of coordination is the lowest.

In this main integration category, we identified refined categories of integration techniques by the characteristics of a microservice as a distinguishing factor.

(C4.1) Scaling Microservice Instances
Description: These techniques address scaling a single microservice and integrating its instances.

Microservices allow deploying instances in a replicated way in order to scale horizontally to individually cope with varying levels of load on each microservice [48,52,55].
Rationale: The distinguishing microservice characteristic is the (horizontal) scalability. The main focus is designing a microservice for scalability and integrating the instances at runtime.
Techniques (5): *(T4.1.1)* Stateless design, *(T4.1.2)* Auto-scale instances based on metrics, *(T4.1.3)* Load balancing between instances, *(T4.1.4)* Load balancing by message broker, *(T4.1.5)* Database clustering and sharding

(C4.2) Service Autonomy
Description: These techniques contribute to the autonomy of the microservice as a technical artifact.

A microservice is a self-contained silo from top to bottom, including its database tables and message queue topics. This self-containment allows autonomous development by the microservice's team, including independent deployment [50,54].
Rationale: The distinguishing microservice characteristic is the autonomy of the microservice. The main focus is supporting the microservice's autonomous life cycle and independence of other microservices.
Techniques (2): *(T4.2.1)* Self-contained design, *(T4.2.2)* Storage area isolation per microservice

(C4.3) Team Autonomy
Description: These techniques contribute to the autonomy of the microservice team.

A microservice team is responsible for the whole life cycle of a microservice, including deployment and (parts of the) operation [1]. Compared to classical monoliths, this requires more knowledge within a microservice team [58,59].
Rationale: The distinguishing microservice characteristic is the autonomy of the microservice team. The main focus is fostering the autonomy of the microservice team by aggregating the knowledge in the team to foster an autonomous life cycle of the microservice.
Techniques (7): *(T4.3.1)* Cross-functional teams, *(T4.3.2)* Experiments, *(T4.3.3)* Education programs, *(T4.3.4)* Support by a task force team, *(T4.3.5)* Use of established patterns, *(T4.3.6)* Proximity to domain-knowledge holders, *(T4.3.7)* Local proximity of team members

We find technical, architectural, and operational techniques mainly in the refined categories C4.1 and C4.2. Category 4.3 deals with the overarching organizational challenge to overcome the knowledge hurdle to build microservices within each team.

6. Illustration of the taxonomy usage

In this section, we present an illustration of how our taxonomy can be utilized to categorize a given microservice integration technique. Illustrations are a common approach to showcase the usefulness of a taxonomy [5].

The following examples of integration techniques are structured as a set of context, problem, and solution. We choose the context-problem-solution presentation format to effectively link a specific problem situation to our theory and provide an explanation on how to solve the given problem [60].

For the taxonomy illustration, we select two complex techniques that are difficult to classify due to multiple valid implementation approaches. These challenging integration scenarios provide better insights into how the taxonomy addresses decision support when classifying them.

The illustrating examples will be the techniques *(T3.3.12)* Service instance discovery and *(T4.1.3)* Load balancing between instances.

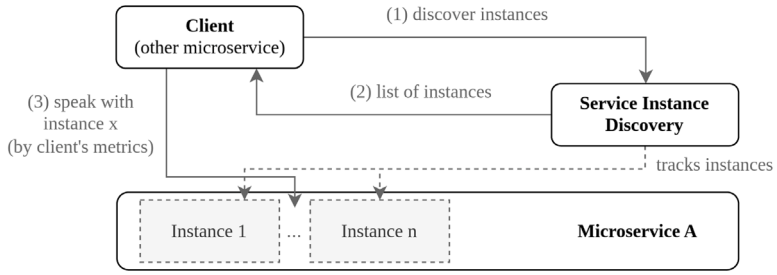
Ambiguities from literature. Service discovery is a concept that is not tied to microservice-based architectures. The term is also used to search, find, and reuse existing software services within an enterprise context [54]. To avoid ambiguities, we use the terminology service instance discovery in this article to express its purpose of discovering and tracking the locations of instances of a microservice.

Implementations of this technique are distinguished in literature as server-side and client-side discovery mechanisms [27]. Client-side discovery requires the users of the discovery mechanism (clients) to be aware of multiple instances of the same microservice. In return, this added complexity on the client side allows the client to select the microservice instance based on client-specific metrics, such as geographical proximity. Server-side discovery is transparent to the client and does not require any knowledge about available instances. Its implementation utilizes a load balancing mechanism that handles the selection of the instance by returning the address of the instance or by acting as a proxy. In our understanding, the service instance discovery technique intends to cope with the dynamic deployment environment by moving configuration from build- to run-time. Instead of communicating with a microservice instance directly, a client has to resolve the network location of an instance first.

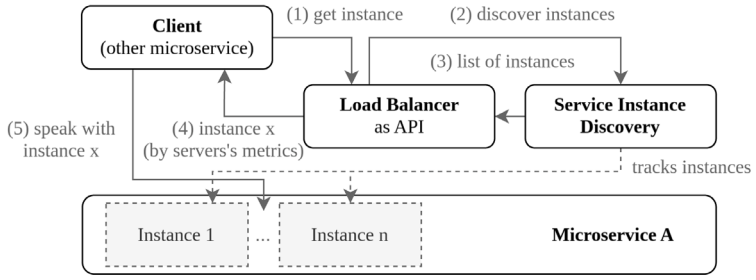
The load balancing technique aims to evenly distribute the load among the instances of a microservice. The literature distinguishes client-side (or internal) and server-side (or external) load balancing mechanisms [21] in the same style as for service instance discovery.

What makes classifying both integration techniques challenging is that there are no clear boundaries regarding the differences between client-side service instance discovery and load balancing. Additionally, they are ambiguous to classify as they act in integration scenarios among microservices and deal with microservice instances. Thus, before making a classification, we examine different technical implementations of both techniques and how they are interconnected (Fig. 4).

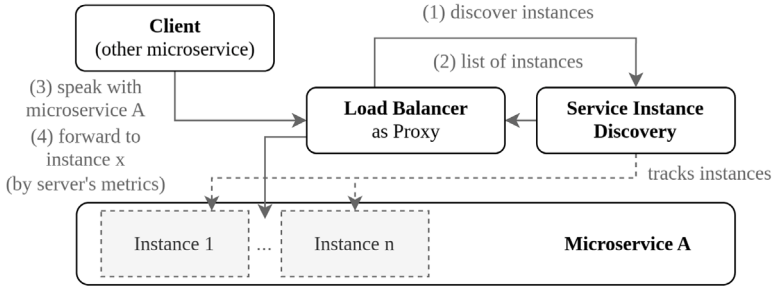
Introspection of technical implementations. A service instance discovery mechanism can have different types of clients: (i) a "real" client in the form of a microservice or a client application, or (ii) a load balancing mechanism. In the case of a real client, the client receives a list of available instances and chooses based on custom metrics. This may or may not incorporate considerations about load balancing. Thus, we do not categorize it as a load balancing technique but rather as an instance discovery one with potential load balancing effects caused by emergence (Fig. 4(a)). The second type of clients are load balancing components. They use the service instance discovery to choose the communication partner for their clients. The load balancing mechanism can either propagate the network location of the chosen service instance to the client (Fig. 4(b)) or serve as a proxy (Fig. 4(c)). These considerations allow us to conclusively define and classify both techniques.



(a) Client-side service discovery



(b) Server-side load balancing



(c) Server-side proxy load balancing

Fig. 4. Load balancing and service instance discovery.

(T3.3.12) Service Instance Discovery

Context:
Microservice instances are deployed to a dynamic environment where fixed network addresses and ports are not guaranteed.

Problem:
Changing network addresses and ports of microservice instances break communication with clients.

Solution:
Introduce a service instance discovery mechanism to keep track of all running microservice instances and their network location.

Main Category: (C3) Technique for Integration among Microservices
Refined Categories: Deployment Integration, Communication Integration, Conceptual Integration
Literature: L3, L5, L6, L7, L8, L9, L12, L18, L19, L20, L21, L24, L25, L31, L34, L37, L40, L42, L45, L52, L53, L54, L55, L59, L61, L63, LN21, LN43, LN44, LN48, LM43, LM47, LM48
Interviews: B

Classification example: service instance discovery. We go top-down to classify the service instance discovery technique: *[Integration technique]* → *[in a microservice-based system]* → *[among microservices]*

A specific implementation might serve external systems, as well. That implementation would additionally satisfy a second different technique.

Within its main category, we assign *[deployment integration]*, *[communication integration]*, and *[conceptual integration]* as refined categories. This is grounded on the solution mainly coping with microservice configuration, an operational topic. The technique impacts the communication behavior of a microservice. It requires the usage of a service instance discovery to determine the network location of its communication partners. Additionally, the service instance discovery technique includes architectural aspects, but it belongs more on the deployment side.

***(T4.1.3)* Load Balancing Between Instances**

Context:
Microservices are deployed with multiple instances each. There is a mechanism to locate service instances in the dynamic deployment environment.

Problem:
There is an uneven load distribution among the instances of a microservice. Relocation and tear-downs of microservice instances due to the dynamic deployment environment lead to communication errors.

Solution:
Introduce a (server-side) load balancing mechanism to automatically distribute the load among (healthy) microservice instances.

Main Category: (C4) Technique for Integration within a Microservice
Refined Categories: Scaling and Integrating Microservice Instances
Literature: L3, L5, L8, L9, L12, L13, L15, L16, L21, L23, L25, L30, L31, L34, L43, L44, L45, L49, L52, L53, L55, L58, L59, L61, L63, LN21, LN43, LN48, LM43, LM47, LM48
Interviews: A, B

Classification example: load balancing between instances. We go top-down to classify the load balancer technique: *[Integration technique]* → *[in a microservice-based system]* → *[within a microservice]*

This technique targets an integration challenge among the instances of a microservice, so integration within a microservice. The load balancing mechanism becomes a facade to the microservice encapsulating all its running instances.

Summary. To summarize, we were able to classify both techniques with the taxonomy. To do so, we used an in-depth examination of the existing technical implementations of both techniques to identify their similarities, differences, and interconnections. Tools often combine different abstract techniques into one single implementation. On the one hand, combining integration techniques into one implementation solution reduces the complexity of the many moving parts in a microservice architecture and supports developers who do not need to think about the specifics of the techniques. On the other hand, it makes reasoning about the underlying techniques considerably harder by disguising them but not removing the complexity of the distributed system.

Our taxonomy of integration techniques enforces being more specific on the abstract principles used in tools and implementations. Using the taxonomy will raise awareness of the complexity of the distributed systems that are built with microservices while supporting practitioners to build new tools based on the combinations of techniques.

7. Discussion

In this section, we contextualize the constructed taxonomy. Section 7.1 introduces an alternative categorization of techniques by their "type", demonstrating the taxonomy's adaptability to various perspectives. Section 7.2 examines the differences between findings from literature and interviews, highlighting trends and gaps in academic research.

To enrich our discussion, we refer to categorized techniques as illustrative examples. A comprehensive list of techniques, including their context-problem-solution descriptions and sources from both literature and interviews, is available in the supplementary materials [6].

7.1. A holistic view on integration

In the previous sections, we presented a taxonomy categorizing microservice integration techniques by the level of control over the integration counterpart. This criterion was selected for its intuitive and practical relevance, as practitioners can readily assess the level of control, making it a useful primary metric for organizing integration techniques.

As we describe in Section 2, microservice integration is a multi-faceted topic that extends beyond purely technical aspects. To illustrate our taxonomy's comprehensiveness, we also classified the techniques into other categories based on their "type", such as architectural, implementation, operational, organizational, and process-oriented focuses. While we consider this classification to be more of an academic exercise, especially with the blending of responsibilities in cross-functional teams and the influence of the DevOps movement, it effectively demonstrates our taxonomy's capacity to address a holistic view of integration.

Architecture-related techniques. They describe the high-level system design, including components and their relationships with each other. These techniques directly design the interaction between software components, either by introducing architectural elements or by using design policies that represent general design primitives. Examples of architectural elements are the API facade (T1.1.1), the proxy microservice to wrap a third-party system (T1.2.1), and the internal integration proxy as a central point of integration within the system (T3.3.10), among many others. Examples of design policies are to gradually replace the legacy system (T1.2.4), question transactions on the domain level (T3.1.10), or use APIs to decouple from implementation details (T3.2.4).

The listing of service-cutting techniques (T3.1.1–T3.1.9) might be surprising since decomposition is the opposite of integration. However, the service decomposition has a significant impact on the later integration. An appropriate alignment of functionalities within and across microservices should always be viewed in combination with their resulting need for interaction. Rather than seeing decomposition and integration as two separate activities, integration design should consider evolving the decomposition as one, making it one joint architectural design activity and an enabling factor for component interaction.

Implementation-related techniques. They describe low-level implementation decisions that are closer to the technology than to an abstract design. These techniques implement the details of the interaction between software components. Examples are resilient consumers that can deal with minor API changes (T3.2.5), bounded retries on communication failure (T3.2.14), or using environment variables for microservice configuration (T3.3.5).

Operation-related techniques. They describe strategies for deploying, maintaining, and monitoring systems in a deployment environment. These techniques support the interaction of components in a deployment environment and contribute to the maintenance of successful component interactions. Examples of these techniques are deploying to a cluster manager like Kubernetes (T3.3.18) or collecting monitoring metrics at different levels (T3.4.20).

Organization-related techniques. They describe the structure, management, and responsibilities of teams in a project. These techniques facilitate organizational structures that enable efficient coordination between the responsible parties of software components, transitively contributing to the interaction of software components. Examples are cross-functional teams (T4.3.1), introducing a task force team to support microservice teams catching up with new technology (T4.3.4), or assigning each microservice to one team (T3.4.4).

Process-related techniques. They describe the coordination between organizational units in terms of single activities, whole methodologies, or tooling support. These techniques implement coordination and management processes within the organizational structure. Similar to organization-related techniques, they contribute to the interaction of software components transitively. Examples are conducting experiments to overcome knowledge hurdles (T4.3.2), introducing standardization across microservices (T3.4.11), or conducting regular cross-team discussions (T3.4.13).

Table 4
Most discussed techniques in literature (more than 20 literature sources);
#L = number of coded literature, #I = number of coded interviews.

ID	Category	Technique	#L	#I
T3.3.23	C3.3	Containers as portable deployment artifacts	47	2
T3.3.1	C3.3	CI/CD for automated deployment	35	4
T4.1.3	C4.1	Load balancing between instances	31	2
T3.3.12	C3.3	Service instance discovery	33	1
T3.3.18	C3.3	Cluster management by container orchestrator	28	2
T4.2.2	C4.2	Storage area isolation per microservice	24	4

Table 5
Most discussed techniques in interviews (covered by at least four interviews);
#L = number of coded literature, #I = number of coded interviews.

ID	Category	Technique	#L	#I
T3.4.10	C3.4	Establish common cultural values	16	5
T3.2.8	C3.2	API versioning	13	5
T3.1.30	C3.1	Domain-motivated alternatives	3	5
T3.4.14	C3.4	Thematic boards for decision making	3	5
T3.4.11	C3.4	Standardization	11	5
T4.3.1	C4.3	Cross-functional teams	15	5

Summary. All presented types of techniques contribute to a well-integrated system in the end by enabling a successful interaction of software components. These types do not map directly to the categories of the taxonomy presented in Section 5 but rather to the individual techniques. For example, the category deployment integration (C3.3) consists of 16 operation-related, four architecture-related, three implementation-related, and one process-related technique; for many techniques, the classification is not fully distinct. However, this showcases the holistic viewpoint on integration the found techniques reflect.

7.2. Focus of literature and interviews

In this section, we will discuss the distinctions between the findings of literature and interviews in terms of most discussed techniques.

We retrospectively analyzed our code system and filtered the techniques most discussed in the literature (Table 4) and in the interviews (Table 5). The sources for a technique resemble simple references, implicit and explicit usage, and detailed discussion of the technique.

Four of the six most discussed techniques in literature (Table 4) are categorized as (C3.3) deployment integration techniques among microservices. The remaining two are categorized as integration techniques (C4.1) for scaling microservice instances and (C4.2) for service autonomy within a microservice. We pin these techniques down as the basics that most microservice-based systems share. They enable automation and cope with the runtime complexity of microservices.

The most discussed techniques in the interviews (Table 5) reveal a different focus of the efforts taken in microservice-based projects. Three of the six techniques are categorized as (C3.4) global knowledge integration techniques. The remaining three techniques are of categories (C3.1) conceptual integration and (C3.2) communication integration among microservices, and (C4.3) team autonomy within a microservice. In general, we see a shift to organizational topics for coordination among teams (T3.4.10, T3.4.14, T3.4.11) and building up knowledge within the teams (T4.3.1).

Although we cannot draw general conclusions solely based on the collected data, the findings suggest that all the essential tools to achieve success in microservice-based architectures are present to tackle the core problems. This highlights the importance of addressing related challenges, such as service cutting, building competencies within teams, and optimizing communication and coordination among teams. In one of our interviews, the interviewee emphasized that tooling is not an obstacle to successful microservice adoption by saying:

"Today, not two to three years ago, not a single microservice project has to fail because of technology. Technology is available, it's good, it's established, and it has been tested in large projects to maturity. Usually, the projects fail because of a wrong [service] cut. That is because you look at your monolith and analyze how it is cut, i.e., in the Java environment, the package structure. The whole thing is often entity-based, and you end up with a wrong cut". [Interview B, translated from German].

The non-organizational techniques T3.2.8 and T3.1.30 go beyond the classic skill set of developers coming from monolithic backend development. APIs of monoliths are not versioned at all or not in the same high frequency as in microservices-based systems. Due to the communication among microservices over a network, failures are much more likely to happen, and coping with them by using domain knowledge needs to become a first-class citizen. These two techniques are representatives of the knowledge required to implement microservices successfully. Even though the emerging tools simplify building microservice architectures, they do not solve the underlying issue: *"Many people don't realize that a microservice application [...] is a highly complex distributed application with all the problems that distributions and automatic scaling entail; both of which are very complex issues. And people often pretend that frameworks, libraries, or platforms can abstract the problem away. But that is not the case, the problem remains. I have to understand the problem well and have a good grip on it. Platforms can provide support, but I have to understand what they do".* [Interview C, translated from German]

Companies starting with microservices often struggle with these high knowledge barriers to successfully build a distributed system. The learning process to overcome these challenges is often driven by pain rather than a plan. It is very experience-based. While literature acknowledges this challenge, it presents only a few superficial solutions, such as establishing cross-functional teams (T4.3.1) or using established patterns in general (T4.3.5). There is a need to even out the steep learning curve with microservice projects and give more direction to the learning process.

In future work, we aim to research and discover further techniques in literature and interviews that are less known. Particularly, we aim to address the needs of the industry elaborated in this section. We will expand on organizational techniques to build a more holistic view of the field of microservices following our insights from the interviews.

8. Limitations

As our research design targets qualitative findings and not quantitative ones, we use the trustworthiness criteria proposed by Lincoln and Guba [61] to discuss the limitations of qualitative studies. The following subsections will discuss the credibility, transferability, dependability, and confirmability of this study.

8.1. Credibility

We acknowledge several inherent limitations to collecting the primary materials regarding credibility, the extent to which the findings accurately reflect the reality being studied and are believable.

First, we reviewed only the literature with the search terms specified above. This selection might exclude suitable articles, e.g., originally not within the microservice area but applicable to it. Furthermore, we cannot claim an exhaustive search due to our iterative approach to a select literature, and thus sources with additional integration techniques may have been overlooked. Although this could affect the resulting taxonomy, it is unlikely due to the level of abstraction in which categories of techniques are constructed. We used theoretical saturation as a stopping criterion to keep effort at an adequate level. By refraining from an exhaustive literature sampling, we might have excluded suitable articles. Second, the acquisition and sampling of interviewees might have excluded suitable interview partners by the limitation to English and German speakers. We believe that choosing English as the lingua franca of software engineering lessens the impact of this limitation.

To mitigate the limitations to credibility, we conducted member checking with the interviewees. We sent them the findings for review before publishing to curate misunderstandings, conflicting opinions, and capture valuable additions. To gain a holistic picture of the field, we used a sampling model to select the interviewees. We considered different roles reflecting different viewpoints on the topic to capture the diversity of the phenomenon. We refined the sampling model with feedback from an expert in the field to capture hidden categories. We spent sufficient time with the data (prolonged engagement) to avoid misinterpretations caused by a superficial immersion into the data. We transcribed the interviews by hand, read the primary materials several times, and immersed ourselves deeply in the thematic analysis over a period of two years.

Finally, it is important to note that this study does not encompass the evaluation of the taxonomy's practical application. It is common in software engineering that studies demonstrate the taxonomy's utility by an illustration [5]. However, we see merit in future work for further empirical evaluation and usage observations involving practitioners to validate the effectiveness of the taxonomy.

8.2. Transferability

We acknowledge several inherent limitations to the collection of the primary materials in terms of transferability, and the extent of generalizability of the findings.

First, we did not conduct an exhaustive sampling of the literature and interviewees. To mitigate this shortcoming, we measured theoretical saturation as changes to the structure of the code system before stopping adding primary materials. Additionally, we applied data triangulation by using white literature and interview transcripts as different types of data.

Second, the selection of literature and interviewees might not be representative of the broader population of interest. We addressed this concern by favoring established and well-adopted literature by ordering by citation count. For the interview selection, we applied purposive sampling. We used a sampling model to include different views on the topic but also included consultants that represent the knowledge aggregated over multiple project contexts instead of narrowly focusing on one specific context.

8.3. Dependability

We acknowledge several inherent limitations to the dependability of the study, the extent to how comprehensible and replicable the research design and execution is.

Table 6
Peer debriefing sessions.

#	Focus
1	Coding ideas (familiarizing with literature)
2	Sampling of literature
3	Research process overview, thematic analysis (results)
4	Thematic analysis (results + process), interviews
5	Presentation of research process

First, the search for literature on Google Scholar might make the study less replicable due to geospatial differences in the search engine. We ordered the results by citation count to mitigate different prioritization of results in different regions. The supplementary materials of the study document the search results to enable reproducibility of later steps.

Second, coding involves identifying and categorizing data into themes, which is a subjective process involving the knowledge and perspective of the researcher. We conducted three inter-coder reliability sessions with fellow researchers using the code system. Their feedback and the indicated maturity by decreasing change proposals over time increase the dependability of the analysis.

Third, the transparency of the data analysis process can affect the dependability of the findings. We mitigate this potential limitation by devoting this whole article to explaining the criteria used to identify different themes, e.g., the level of control over the integration counterpart. Further, we rigorously describe the data selection procedures to select the primary materials, increasing the reproducibility of the data gathering process.

8.4. Confirmability

We acknowledge several inherent limitations to the confirmability of the study, the extent to which the biases and perspectives of the researcher shaped the results.

Due to the subjective nature of the thematic analysis, the findings are at risk of researcher bias. To mitigate introducing biases, we complemented the continuous professional exchange among all co-authors and further members of our research group with regular peer debriefings to "[...] confirming that the findings and the interpretations are worthy, honest, and believable" [62]. We chose debriefers who either had a good understanding of the studied domain, a similar background in the qualitative methodology, or both. They continuously questioned the overall methodology and the analyzed and interpreted data, checking for potential bias. Table 6 gives an overview of the peer debriefing sessions we conducted and their focus. Additionally, the inter-coder reliability sessions required the maintenance of a code book, a supplemental material explaining the rationale of each theme and code, when they are applied, and when they are not applied. Formulating these criteria, as well as the qualitative feedback further improves the confirmability of the findings. Data triangulation by using literature and interviews as different types of primary materials, as well as discussing our thoughts with the interviewees in the interviews directly, but also in the member checking procedure, supported us in minimizing biases.

9. Conclusion

In this article, we have presented a comprehensive hierarchical taxonomy for microservice integration techniques consisting of five main categories. The clear separation of the main categories is guided by the criteria of the degree of control over the integration counterpart. Further, we present ten refined categories categorizing 121 techniques. Section 6 presents two of these techniques in detail to illustrate the taxonomy usage with an exemplary classification rationale. The comprehensive supplementary materials [6] present the taxonomy

populated with 121 integration techniques. Presenting and discussing all those techniques in detail is subject to future work as the scope of this article cannot do them justice.

In the broader context of software taxonomies, the proposed taxonomy follows the most prevalent taxonomy features [5]: A solution-focused research type, a combination of graphical and textual taxonomy notation, a hierarchical taxonomy structure, a clear descriptive basis of categories, a qualitative classification procedure that is explicitly described, and an illustration for utility demonstration. In contrast to most other studies, we did not construct the taxonomy in an ad-hoc manner but employed a structured research design for theory building.

The taxonomy provides common terminology to ease sharing knowledge among researchers, but also among participants and stakeholders in projects using microservices. The enhanced clarity improves decision-making by enabling teams to systematically select the most appropriate integration techniques in their current situation. Further, the structure can serve as a guided learning path for newcomers, also highlighting the socio-technical challenges of microservice architectures. The amount of organizational challenges and techniques we found in our study emphasizes the need to investigate microservice techniques not in a silo-style way but from a topic-centered viewpoint, including socio-technical and organizational aspects next to architectural, operational, and technical ones.

However, it is important to mention that the listed techniques cannot be considered complete or comprehensive for the practice of microservice integration. We emphasize the importance of conducting future research to empirically evaluate the proposed taxonomy and its associated integration techniques. The taxonomy's usefulness in finding solutions to integration problems and its effectiveness in saving time and resources through a guided process should be empirically evaluated in industrial contexts. Future work will conduct such an evaluation study across multiple projects.

In conclusion, our taxonomy represents a significant step towards a more comprehensive and holistic understanding of microservice integration techniques. By providing a structure for future research and practice, we hope to inspire further investigation and facilitate the successful adoption of microservices.

CRediT authorship contribution statement

Georg-Daniel Schwarz: Writing – original draft, Software, Project administration, Methodology, Investigation, Funding acquisition, Data curation, Conceptualization. **Andreas Bauer:** Writing – review & editing, Visualization, Validation, Software, Conceptualization. **Dirk Riehle:** Writing – review & editing, Methodology, Conceptualization. **Nikolay Harutyunyan:** Writing – review & editing, Validation, Methodology.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used ChatGPT in order to improve and rephrase written paragraphs. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Funding

This work was supported by BMBFs (Federal Ministry of Education and Research) Software Campus 2.0 project (BePra-MSI, 01IS17045), and by DFGs (German Research Foundation) Research Grants Programme (Industry Best Practices for Microservice Integration, RI 2147/9-1).

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Dirk Riehle reports financial support was provided by German Research Foundation. Georg-Daniel Schwarz reports financial support was provided by German Federal Ministry of Education and Research. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The research presented in this paper was conducted in partial fulfillment of the requirements for a cumulative dissertation. We extend our sincere thanks to our colleagues for their constructive feedback and careful proofreading. We are deeply grateful to the industry, and especially to the interviewees, whose generous contribution of time was invaluable to this study.

Data availability

The datasets generated during the current study are available at the following URL: https://zenodo.org/records/12740383 [6].

References

[1] P. Jamshidi, C. Pahl, N.C. Mendonça, J. Lewis, S. Tilkov, Microservices: The journey so far and challenges ahead, IEEE Softw. 35 (3) (2018) 24–35.
[2] J. Lewis, M. Fowler, Microservices: a definition of this new architectural term, 2014, URL https://martinfowler.com/articles/microservices.html.
[3] S. Newman, Building microservices, O'Reilly Media, 2021.
[4] S. Baškarada, V. Nguyen, A. Koronios, Architecting microservices: Practical opportunities and challenges, J. Comput. Inf. Syst. 60 (5) (2020) 428–436.
[5] M. Usman, R. Britto, J. Börstler, E. Mendes, Taxonomies in software engineering: A Systematic mapping study and a revised taxonomy development method, Inf. Softw. Technol. 85 (2017) 43–59, http://dx.doi.org/10.1016/j.infsof.2017.01.006.
[6] G. Schwarz, A. Bauer, Supplementary materials for the study "A taxonomy of microservice integration techniques", 2024, http://dx.doi.org/10.5281/zenodo.12740383.
[7] O. Nierstrasz, L. Dami, Component-oriented software technology, Object-Oriented Softw. Compos. 1 (1995) 3–28.
[8] K.-K. Lau, Software component models, in: Proceedings of the 28th International Conference on Software Engineering, 2006, pp. 1081–1082.
[9] M.-T. Schmidt, B. Hutchison, P. Lambros, R. Phippen, The enterprise service bus: making service-oriented architecture real, IBM Syst. J. 44 (4) (2005) 781–797.
[10] D. Shadija, M. Rezai, R. Hill, Towards an understanding of microservices, in: 2017 23rd International Conference on Automation and Computing, ICAC, IEEE, 2017, pp. 1–6.
[11] H. Barki, A. Pinsonneault, Explaining ERP implementation effort and benefits with organizational integration, 2002, Cahier du GReSI no 2.
[12] N. Mohamed, B. Mahadi, S. Miskon, H. Haghshenas, H.M. Adnan, Information system integration: A review of literature and a case analysis, in: Mathematics and computers in contemporary science, Wseas LLC, 2013, pp. 68–77.
[13] W. Hasselbring, Information system integration, Commun. ACM 43 (6) (2000) 32–38.
[14] P. Bourque, R.E. Fairley (Eds.), SWEBOK: Guide to the Software Engineering Body of Knowledge, Version 3.0, IEEE Computer Society, Los Alamitos, CA, 2014.
[15] R. Keshav, R. Gamble, Towards a taxonomy of architecture integration strategies, in: Proceedings of the Third International Workshop on Software Architecture, 1998, pp. 89–92.
[16] H. Hofmeister, G. Wirtz, A pattern taxonomy for business process integration oriented application integration., in: SEKE, 2006, pp. 114–119.
[17] V. Clarke, V. Braun, N. Hayfield, Thematic analysis, Qual. Psychology: A Pr. Guid. Res. Methods 222 (2015) 248.
[18] J. Bogner, A. Zimmermann, Towards integrating microservices with adaptable enterprise architecture, in: 2016 IEEE 20th International Enterprise Distributed Object Computing Workshop, EDOCW, 2016, pp. 1–6.
[19] R. Petrasch, Model-based engineering for microservice architectures using enterprise integration patterns for inter-service communication, in: 2017 14th International Joint Conference on Computer Science and Software Engineering, JCSSE, IEEE, 2017, pp. 1–4.

[20] B. Shafabakhsh, R. Lagerström, S. Hacks, Evaluating the impact of inter process communication in microservice architectures., in: QuASoQ@ APSEC, 2020, pp. 55–63.
[21] A. Balalaie, A. Heydarnoori, P. Jamshidi, D.A. Tamburri, T. Lynn, Microservices migration patterns, Software: Pr. Exp. 48 (11) (2018) 2019–2042.
[22] H. Harms, C. Rogowski, L. Lo Iacono, Guidelines for adopting frontend architectures and patterns in microservices-based systems, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, 2017, pp. 902–907.
[23] F. Osses, G. Márquez, H. Astudillo, An exploratory study of academic architectural tactics and patterns in microservices: A systematic literature review, in: Avances en Ingenieria de Software a Nivel Iberoamericano, CIbSE, Vol. 2018, 2018, pp. 71–84.
[24] J. Fritzsch, J. Bogner, A. Zimmermann, S. Wagner, From monolith to microservices: A classification of refactoring approaches, in: Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment: First International Workshop, DEVOPS 2018, Chateau de Villebrumier, France, March 5-6, 2018, Revised Selected Papers 1, Springer, 2019, pp. 128–141.
[25] S. Weerasinghe, I. Perera, Taxonomical classification and systematic review on microservices, Int. J. Eng. Trends Technol. 70 (3) (2022) 222–233.
[26] G. Márquez, H. Astudillo, Actual use of architectural patterns in microservices-based open source projects, in: 2018 25th Asia-Pacific Software Engineering Conference, APSEC, Ieee, 2018, pp. 31–40.
[27] D. Taibi, V. Lenarduzzi, C. Pahl, Architectural patterns for microservices: A systematic mapping study, in: V. ctor Méndez Muñoz, D. Ferguson, M. Helfert, C. Pahl (Eds.), Proceedings of the 8th International Conference on Cloud Computing and Services Science, CLOSER 2018, Funchal, Madeira, Portugal, March 19-21, 2018, SciTePress, 2018, pp. 221–232.
[28] M. Söylemez, B. Tekinerdogan, A. Kolukısa Tarhan, Challenges and solution directions of microservice architectures: A systematic literature review, Appl. Sci. 12 (11) (2022) 5507.
[29] D. Taibi, V. Lenarduzzi, C. Pahl, Microservices anti-patterns: A taxonomy, in: Microservices, Springer, 2020, pp. 111–128.
[30] K. Brown, B. Woolf, Implementation patterns for microservices architectures, in: Proceedings of the 23rd Conference on Pattern Languages of Programs, 2016, pp. 1–35.
[31] F. Osses, G. Márquez, H. Astudillo, Exploration of academic and industrial evidence about architectural tactics and patterns in microservices, in: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, 2018, pp. 256–257.
[32] N. Alshuqayran, N. Ali, R. Evans, A systematic mapping study in microservice architecture, in: 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications, SOCA, IEEE, 2016, pp. 44–51.
[33] B. Kitchenham, Procedures for performing systematic reviews, vol. 33, (no. 2004) Keele University, Keele, UK, 2004, pp. 1–26.
[34] H. Jansen, The logic of qualitative survey research and its position in the field of social research methods, Forum Qualitative Sozialforschung (Forum: Qualitative Soc. Res.) 11 (2) (2010).
[35] O. Zimmermann, Microservices tenets, Comput. Sci.- Res. Dev. 32 (3) (2017) 301–310.
[36] V. Garousi, M. Felderer, M.V. Mäntylä, Guidelines for including grey literature and conducting multivocal literature reviews in software engineering, Inf. Softw. Technol. 106 (2019) 101–121.
[37] L.A. Guion, D.C. Diehl, D. McDonald, Triangulation: establishing the validity of qualitative studies: FCS6014/FY394, rev. 8/2011, Edis 2011 (8) (2011) 3.
[38] H. Kallio, A.-M. Pietilä, M. Johnson, M. Kangasniemi, Systematic methodological review: developing a framework for a qualitative semi-structured interview guide, J. Adv. Nurs. 72 (12) (2016) 2954–2965.
[39] J.J. Francis, M. Johnston, C. Robertson, L. Glidewell, V. Entwistle, M.P. Eccles, J.M. Grimshaw, What is an adequate sample size? Operationalising data saturation for theory-based interview studies, Psychol. Heal. 25 (10) (2010) 1229–1245.
[40] A. Strauss, J. Corbin, Basics of qualitative research techniques, Citeseer, 1998.
[41] V. Braun, V. Clarke, Using thematic analysis in psychology, Qual. Res. Psychol. 3 (2) (2006) 77–101.
[42] T. Cerny, M.J. Donahoo, M. Trnka, Contextual understanding of microservice architecture: current and future directions, ACM SIGAPP Appl. Comput. Rev. 17 (4) (2018) 29–45.
[43] S. Salii, J. Ajdari, X. Zenuni, Migrating to a microservice architecture: benefits and challenges, in: 2023 46th MIPRO ICT and Electronics Convention, MIPRO, 2023, pp. 1670–1677, http://dx.doi.org/10.23919/MIPRO57284.2023.10159894.
[44] N. Dragoni, S. Giallorenzo, A.L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, L. Safina, Microservices: Yesterday, today, and tomorrow, in: M. Mazzara, B. Meyer (Eds.), Present and Ulterior Software Engineering, Springer International Publishing, Cham, 2017, pp. 195–216, http://dx.doi.org/10.1007/978-3-319-67425-4_12.
[45] M. Kalske, N. Mäkitalo, T. Mikkonen, Challenges when moving from monolith to microservice architecture, in: Current Trends in Web Engineering: ICWE 2017 International Workshops, Liquid Multi-Device Software and EnWoT, Practi-O-Web, NLPIT, SoWeMine, Rome, Italy, June 5-8, 2017, Revised Selected Papers 17, Springer, 2018, pp. 32–47.
[46] T. Salah, M.J. Zemerly, C.Y. Yeun, M. Al-Qutayri, Y. Al-Hammadi, The evolution of distributed systems towards microservices architecture, in: 2016 11th International Conference for Internet Technology and Secured Transactions, ICITST, IEEE, 2016, pp. 318–325.
[47] D.I. Savchenko, G.I. Radchenko, O. Taipale, Microservices validation: Mjolnirr platform case study, in: 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO, IEEE, 2015, pp. 235–240.
[48] J. Soldani, D.A. Tamburri, W.-J. Van Den Heuvel, The pains and gains of microservices: A systematic grey literature review, J. Syst. Softw. 146 (2018) 215–232.
[49] L. Chen, Microservices: architecting for continuous delivery and DevOps, in: 2018 IEEE International Conference on Software Architecture, ICSA, IEEE, 2018, pp. 39–397.
[50] A. Bucchiarone, N. Dragoni, S. Dustdar, S.T. Larsen, M. Mazzara, From monolithic to microservices: An experience report from the banking domain, Ieee Softw. 35 (3) (2018) 50–55.
[51] V. Heorhiadi, S. Rajagopalan, H. Jamjoom, M.K. Reiter, V. Sekar, Gremlin: Systematic resilience testing of microservices, in: 2016 IEEE 36th International Conference on Distributed Computing Systems, ICDCS, IEEE, 2016, pp. 57–66.
[52] A. Singleton, The economics of microservices, IEEE Cloud Comput. 3 (5) (2016) 16–20.
[53] D. Lu, D. Huang, A. Walenstein, D. Medhi, A secure microservice framework for iot, in: 2017 IEEE Symposium on Service-Oriented System Engineering, SOSE, IEEE, 2017, pp. 9–18.
[54] Z. Xiao, I. Wijegunaratne, X. Qiang, Reflections on SOA and microservices, in: 2016 4th International Conference on Enterprise Systems, ES, IEEE, 2016, pp. 60–67.
[55] N. Dragoni, I. Lanese, S.T. Larsen, M. Mazzara, R. Mustafin, L. Safina, Microservices: How to make your application scale, in: International Andrei Ershov Memorial Conference on Perspectives of System Informatics, Springer, 2017, pp. 95–104.
[56] W. Hasselbring, G. Steinacker, Microservice architectures for scalability, agility and reliability in e-commerce, in: 2017 IEEE International Conference on Software Architecture Workshops, ICSAW, IEEE, 2017, pp. 243–246.
[57] P. Di Francesco, P. Lago, I. Malavolta, Migrating towards microservice architectures: an industrial survey, in: 2018 IEEE International Conference on Software Architecture, ICSA, IEEE, 2018, pp. 29–2909.
[58] D. Taibi, V. Lenarduzzi, C. Pahl, Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation, IEEE Cloud Comput. 4 (5) (2017) 22–32.
[59] A. Balalaie, A. Heydarnoori, P. Jamshidi, Migrating to cloud-native architectures using microservices: an experience report, in: European Conference on Service-Oriented and Cloud Computing, Springer, 2015, pp. 201–215.
[60] D. Riehle, H. Züllighoven, Understanding and using patterns in software development, Tapos 2 (1) (1996) 3–13.
[61] Y.S. Lincoln, E.G. Guba, Naturalistic inquiry, sage, 1985.
[62] S. Spall, Peer debriefing in qualitative research: Emerging operational models, Qual. Inq. 4 (2) (1998) 280–292.