The Journal of Systems & Software 195 (2023) 111521

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

Revisiting the practices and pains of microservice architecture in reality: An industrial inquiry☆

Xin Zhou [a], Shanshan Li [a,*], Lingli Cao [a], He Zhang [a], Zijia Jia [b], Chenxing Zhong [a], Zhihao Shan [c], Muhammad Ali Babar [d]

[a] State Key Laboratory of Novel Software Technology, Software Institute, Nanjing University, Jiangsu, China
[b] ByteDance, Inc., Hangzhou, China
[c] Tencent Technology Co.Ltd, Shenzhen, China
[d] School of Computer Science, University of Adelaide, Australia

ARTICLE INFO

Article history:
Received 22 January 2022
Received in revised form 4 August 2022
Accepted 27 September 2022
Available online 6 October 2022

Keywords:
Microservices
Empirical study
Interview
Software architecture

ABSTRACT

Background: Seeking an appropriate architecture for the design of software is always a challenge. Although microservices are claimed to be a lightweight architecture style that can improve current practices with several characteristics, many practices are based on different circumstances and reflect variant effects. Empirical inquiry gives us a systematic insight into industrial practices and sufferings on microservices.
Objective: This study is to investigate the gaps between ideal visions and real industrial practices in microservices and what expenses microservices bring to industrial practitioners.
Method: We carried out a series of industrial interviews with practitioners from 20 software companies. The collected data were then codified using qualitative methods.
Results: Eight pairs of common practices and pains of microservices in industry were obtained after synthesizing the rich and detailed data collected. Five aspects that require careful decisions were extracted to help practitioners balance the possible benefits and pains of MSA. Five research directions that need further exploration were identified based on the pains associated with MSA.
Conclusion: While the benefits of microservices are confirmed from the point of view of practitioners, decisions should be carefully made and the possible problems identified must be addressed with additional expense from experience. Furthermore, some of the topics and pains outlined, e.g., systematic evaluation and assessment, organizational transformation, decomposition, distributed monitoring, and bug localization, may inspire researchers to conduct further research.

1. Introduction

Microservices Architecture (MSA) has become the latest trend in software development. It advocates the concept of componentization, based on which a single application is implemented as a set of small and independent services (Newman, 2015). Each MSA service is run in its own process and communicates with others through lightweight mechanisms (Fowler and Lewis, 2014). Many advantages related to MSA have been justified in both academia and industry, for example, scalability and independent deployment. Inspired by the perceived benefits of MSA, a large number of world-leading Internet companies such as Netflix, Amazon, and eBay have migrated to MSA.

However, research and practice in recent years justify the statements of Fowler and Lewis (2014) that MSA is not another silver bullet and has several challenges in different phases (Jamshidi et al., 2018; Dragoni et al., 2017). Due to the distributed setting of services, MSA could make the development and operation of microservices more complicated in practice, for example, causing potential problems in data consistency, transaction management, and service communication within the network (Neri et al., 2019; Alshuqayran et al., 2016; Furda et al., 2017; Carrasco et al., 2018). It is important to shed light on the gaps between the ideal characteristics and the realistic practices of MSA and to clarify the trade-offs between the benefits and costs associated with MSA (Hassan and Bahsoon, 2016; Hasselbring and Steinacker, 2017). The widely concerned characteristics and common industrial views of MSA in different domains can help practitioners, especially newcomers of MSA, in conducting the migration and

☆ Editor: Uwe Zdun.

* Corresponding author.
E-mail addresses: xinzhou@smail.nju.edu.cn (X. Zhou), lss@nju.edu.cn (S. Li), e19201094@stu.ahu.edu.cn (L. Cao), hezhang@nju.edu.cn (H. Zhang), stu_zijiajia@163.com (Z. Jia), chenxingzhong1125@gmail.com (C. Zhong), cycle44@gmail.com (Z. Shan), ali.babar@adelaide.edu.au (M.A. Babar).

https://doi.org/10.1016/j.jss.2022.111521
0164-1212/

other practices more efficiently and successfully. The accompanying pains may affect decision-making about migrating legacy systems to MSA or developing a new system using MSA style. In addition, the pains encountered by practitioners can inspire researchers willing to identify potential research opportunities in this area.

Currently, studies have been conducted to investigate the state-of-the-art of MSA adoption (Alshuqayran et al., 2016; Pahl and Jamshidi, 2016; Francesco et al., 2017), the migration processes to MSA (Taibi et al., 2017; Di Francesco et al., 2018), and even solutions to address specific challenges during the migration, for example, the MSA-oriented decomposition (Chen et al., 2017; Sayara et al., 2017; Mazlami et al., 2017; Ren et al., 2018; Jin et al., 2018; Eski and Buzluca, 2018; De Alwis et al., 2018; Tyszberowicz et al., 2018; Li et al., 2019; Jin et al., 2021; Taibi and Systä, 2019). On the contrary, this study takes a unique angle distinct from all other MSA studies, comparing the most influential statements of Fowler and Lewis with industrial practices after six years since their statements (Fowler and Lewis, 2014). According to our investigation, none of the other relevant studies provided an in-depth gap analysis between the characteristic statements to be expected for MSA (Fowler and Lewis, 2014) and the reality of the practical experiences of MSA in different domains. Furthermore, the compromise between the benefits and the sufferings of MSA in a practical context does not receive enough attention from the relevant studies.

Motivated by the lack of knowledge in recent studies on MSA, we intended to not only identify the gaps between the common visions and the practical state of MSA, but also understand the trade-offs between its recognized characteristics and its accompanying pains when designing, implementing, and operating microservices. To achieve this, we conducted a series of semi-structured interviews with practitioners from twenty software companies. Data collected from interviewees help reveal the state of practice, including concerns, benefits, and pains associated with MSA.

The main contributions of this study are threefold: (1) It identifies the gaps between the best-known characteristics and the real practices in MSA and provides a trade-off analysis of the benefits and pains of MSA taking a unique angle different from others; (2) It proposes a unified and integrated overview map of the general practices and pains around the best-known characteristics of microservices identified from the industry; (3) It condenses five decisions that should be carefully made by practitioners and discusses the potential future research directions in this area from both the perspectives of researchers.

This paper is an extended version of the conference paper (Zhang et al., 2019) published in the proceedings of the International Conference on Software Architecture (ICSA) in 2019. Compared to the conference paper, we extend and update this paper from the following aspects: (1) Increased sample size. Specifically, we additionally invite seven interviewees from specific domains: three from the software/IT domain, one from the IT service provider domain, one from the telecommunication domain, and two from the E-commerce domain; (2) Re-answering RQs around the domain. The gaps and pains of MSA are analyzed around different domains, which is not considered in the conference paper; (3) An integrated overview map and a more in-depth discussion. More interesting findings and results, for example, four types of testing tactics and two categories of governance tactics, are obtained through additional interviews and new data synthesis methods in this study. These findings are further unified into an overview map, which integrates the practices and pains around the best-known characteristics of MSA and aims to provide practitioners, especially newcomers of MSA, with an introductory guide for systems' migration to microservices. (4) This study gives a more in-depth discussion of the implications for practitioners. Specifically, it extracts five aspects that require careful decisions for the trade-off between possible benefits and pains of MSA before adoption.

The remainder of this paper is structured as follows. Section 2 explains the research method of this study. Section 3 shows the demographic results of all interviewees. In Section 4, we report the findings that answer our research questions and discuss the overview map that integrates the common practices and pains of MSA. Based on the findings in Section 4, Section 5 further discusses five decisions that practitioners should make with care and three potential directions worthy of further research in academia. Section 6 elaborates threats to the validity of our study and Section 7 presents an overview of related work. Section 8 draws the conclusion and suggests future work.

2. Research method

This study was initiated in the second half of 2020 and then carried out for four months using a common qualitative research approach: interview (Kvale, 2008). Three student researchers and their two supervisors participated in designing the interview instrument, collecting the data through interviewing practitioners, and analyzing the data for reporting.

2.1. Research questions

To characterize the practices and challenges related to the migration and adoption towards MSA, we defined the following two research questions (RQs).

RQ1: What are the gaps between visions and the reality of microservices? This RQ aims to: (1) investigate real concerns and practices in terms of the specific visions (characteristics) claimed of MSA between different domains; (2) compare the results around different domains; and (3) generate an overview map about practices to guide practitioners.

RQ2: What are the pains of implementing microservices in practice? This RQ is to identify the perceived pains that need to be addressed with additional costs in industrial practices and implications for potential research directions.

We took the best known visions in the community as the reference of the microservice characteristics researched in this study, which were proposed by Fowler and Lewis in 2014 (Fowler and Lewis, 2014). According to the citation information searched in Google Scholar, 129 studies have cited the nine characteristics about Fowler and Lewis microservices. Table 1 shows the list and brief description of these characteristics. Similar to our conference paper (Zhang et al., 2019), no practices and sufferings around the third characteristic–"products not projects" were obtained from the interviewees, which is not technical but process-related (Zimmermann, 2017). Therefore, we do not discuss this characteristic when reporting the findings later.

2.2. Research procedure

This study aims to gain an in-depth understanding of the state-of-the-practices of MSA in the software industry, with the specific focus on exploring the gaps between shared visions and real practices in microservices. Therefore, we collected data from practitioners with experience in MSA using semi-structured interviews, which is a common research method for collecting qualitative data (Seaman, 1999). The interviews were conducted through a high degree of discussion between interviewees and interviewers with a mixture of closed and open questions, which allows the researchers to obtain foreseeable and unexpected data across the interviews (questionnaires as shown in Table 2).

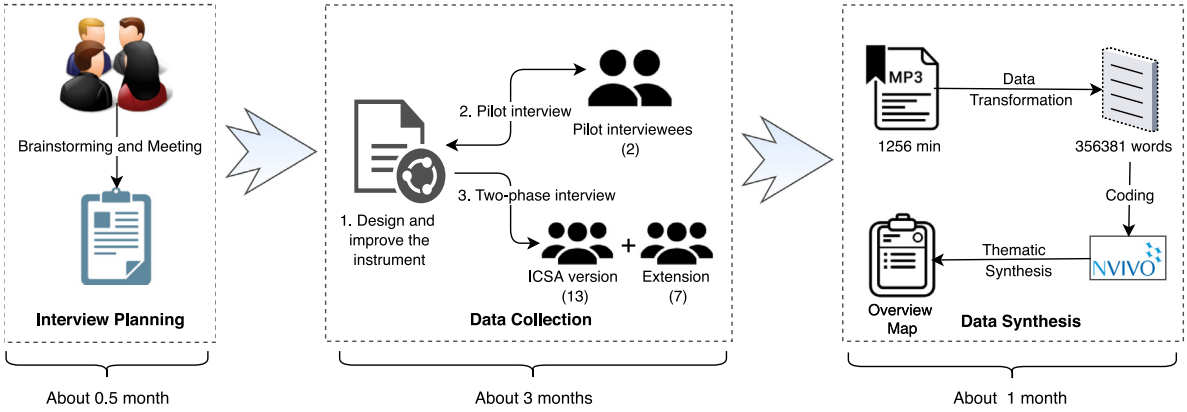

Fig. 1. The procedure of the interview study.

Table 1
Characteristics of microservices architecture from Fowler and Lewis (2014).

No.	Characteristics	Description
C1	Componentization via Services	Running in own process and communicating with lightweight mechanisms through separation
C2	Organized around Business Capabilities	Model around business concepts or team boundaries
C3	Products not Projects	Not technical but process-related (Zimmermann, 2017)
C4	Smart endpoints and dumb pipes	Communication protocols, e.g., HTTP RESTful API
C5	Decentralized Governance	Polyglot programming
C6	Decentralized Data Management	Polyglot persistence
C7	Infrastructure Automation	Continuous delivery
C8	Design for failure	Resilience and monitoring
C9	Evolutionary Design	Service design methods

Table 2
Guidelines for the core questions of the questionnaire.

Code	Questions	Notes
Q1	How many years have you been involved in software development? How many years have you been involved in architectural design?	
Q2	What drives you and your team or enterprise to choose MSA?	Business-driven, Technology-driven, Other
Q3	Which of the following aspects do you think your team's or enterprise's vision (Characteristics) for microservices includes?	C1. Componentization via Services C2. Organized around Business C3. Products not Projects C4. Smart endpoints and dumb pipes C5. Decentralized C6. Decentralized Data Management C7. Infrastructure Automation C8. Design for failure C9. Evolutionary Design
$Q4_1$	Can you share with us your practical experience about XX vision (characteristics)? (please give more examples)	
$Q4_2$	What are the pain points of your team implementing microservices in this practice?	

The procedure for this interview-based investigation is illustrated in Fig. 1. At the initial stage, three student researchers conducted brainstorming to jointly identify MSA interests and had a planning meeting with their supervisors to determine the interview plan. Then they designed the interview instrument with respect to the interests obtained from the meetings as well as inspired by the nine characteristics claim by Fowler and Lewis (2014) and other previous MSA related survey studies (Chen, 2018; Di Francesco et al., 2018; Taibi et al., 2017). Meanwhile, the researchers started inviting suitable practitioners to participate in the interviews. We searched for experienced practitioners among the speakers in three influential industrial forums related to software architecture in China. We then invited the selected practitioners who worked in the domain of software architecture for more than two years and also had MSA experience as our interviewee candidates. Finally, a two-phase interview (13 + 7) was conducted with practitioners from twenty companies. In the first phase, 13 practitioners accepted our invitation, from whom we collected useful data and obtained interesting findings, as described in the conference paper (Zhang et al., 2019). In the second phase, to identify more domain-specific findings, we found another seven practitioners from different domains who agreed to participate in our interview-based survey. To evaluate and improve the interview instruments, two pilot interviews (answers from pilot interviewers were not included in the results of the paper) with carefully selected domain experts from our partners were rigorously conducted before conducting interviews with the invited interviewees. The instruments experienced eight revisions based on useful feedback from the two pilot interviews.

2.3. Data collection

The instruments of our semi-structured interviews were designed under the Kvale guide (Kvale, 2008), and reviewed by the two supervisors with extensive experience in interview studies. During the pilot interviews with two domain experts, the instrument was improved and some questions were also clarified. The final version of our interview instrument is divided into three parts.

(1) Warm-up: Basic information about practitioners and their companies;
(2) Practices of MSA: Concerns, practices and processes around the characteristics of microservices claimed by Fowler and Lewis (2014);
(3) Gains and pains: Benefits and sufferings of microservices, moreover, those that need further research.

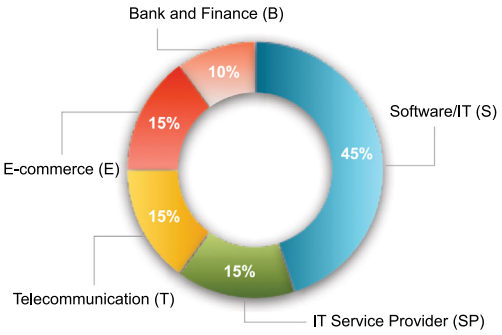

Fig. 2. Organization domains.

Some questions in the semi-structured interviews are designed to be straightforward and closed, e.g., the questions for the basic information. Others are open-ended to steer the interview under different circumstances, which means that the order or the details of specific questions are not fixed and depend on the interviewee's experience and answers.

These interviews were mostly conducted face-to-face at the interviewees' sites located in seven cities in China. Six interviews were conducted online due to geographic restriction. We spent about three months conducting the two-phase interview on all 20 interviews. Each interview involved an interviewee and two or three researchers as interviewers. During the interview process, one researcher was responsible for asking questions with the guide of the instruments, while the other(s) took notes and posed complementary questions to ensure the interest was covered. All interviews took 21 h in total with an average of 63 min each. Meanwhile, in case of losing some important information, all interviews were recorded using a digital voice recorder after getting the interviewees' permission.

2.4. Data synthesis

The data collected from the interviewees was processed and analyzed to generate the key findings of this study. Firstly, a student processed the transcription of the interview recordings with the aid of a speech recognition application, and then the other two students performed a double check. The transcripts of the 21 h of interviews that contain 356 381 words in Chinese were coded with the notes taken during the interviews following the coding manual (Saldaña, 2015). To avoid introducing the researcher's bias, three student researchers coded each interview transcript independently. Any disagreements were discussed and resolved through regular meetings with their supervisors.

As recommended in Welsh (2002), the data synthesis in this study combined both manual and computer-assisted methods with the facility of a qualitative data synthesis package, NVivo, which can help researchers explore patterns efficiently from large amounts of textual data and improve the rigor of the synthesis process. However, because NVivo is less useful in terms of the generation of thematic ideas, we only use NVivo for textual data management, word frequency search (to help quickly identify some important information) and first round of data coding (to build a preliminary mind map). Then we applied thematic synthesis (Patton, 1990) method to encode the data step by step, and manually analyzed concerns and practices in different phases of MSA.

Table 3
The detailed information of the 20 organizations.

No.	Domains	Locations	Employees
S1	Software/IT	Hefei	5000–10 000
S2	Software/IT	Shanghai	5000–10 000
S3	Software/IT	Beijing	200–500
S4	Software/IT	Hangzhou	500–2000
S5	Software/IT	Nanjing	<100
S6	Software/IT	Nanjing	<100
S7[a]	Software/IT	Shanghai	500–2000
S8[a]	Software/IT	Tianjin	2000–5000
S9[a]	Software/IT	Shanghai	2000–5000
SP1	IT Service Provider	Shanghai	200–500
SP2	IT Service Provider	Beijing	5000–10 000
SP3[a]	IT Service Provider	Shanghai	500–2000
T1	Telecommunication	Nanjing	>10 000
T2	Telecommunication	Shanghai	>10 000
T3[a]	Telecommunication	Shanghai	>10 000
E1	E-commerce	Nanjing	5000–10 000
E2[a]	E-commerce	Beijing	>10 000
E3[a]	E-commerce	Guangzhou	2000–5000
B1	Bank and Finance	Hefei	>10 000
B2	Bank and Finance	Guangzhou	5000–10 000

[a]Interviewees added in the journal version.

3. Demographic results

The interviewees are professionals who are involved in the MSA domain as project managers (6/20) or senior software architects (14/20). They have 3 to 18 years of experience in software architecture, and more than half of them have no less than five years of experience. Project managers are mainly responsible for the management of legacy systems' migration to MSA or new product lines, while the senior software architects primarily take charge of the architecture of projects. To gain an in-depth understanding of the state-of-the-practice of MSA, we took both completed and ongoing microservices projects into the synthesis.

The twenty organizations of our interviewees are located in seven major cities in China, i.e. Shanghai (35%), Nanjing (20%) and other cities (45%). In this study, the organizations with which the interviewees worked are categorized into five domains, as shown in Fig. 2. Nine interviewees work in the general Software/IT domain, three are in the IT Service Provider domain, three are in the Telecommunication domain, two are in the E-commerce domain, and the rest two are in the Bank and Finance domain. Table 3 presents the detailed information of the twenty organizations, that is, domains, locations, and the number of employees.

4. Findings

This study aims to reflect the state-of-the-practice of microservices in industry and identify the gaps between the visions of the community and the practical reality. During interviews, the interviewees were asked about their practices during the adoption of the microservices architecture (RQ1), problems that need to be addressed with additional expenses (RQ2) with microservices in terms of the characteristics claimed by Fowler and Lewis (2014). Unfortunately, we did not obtain any informative evidence related to the characteristic C3 (Products not Projects) as displayed in Table 1, which is not a technical characteristic and does not cause any expense to the interviewees. Consequently, this section reports the more evidential findings on the eight remaining characteristics. For a clear presentation, the findings are made up of two parts for each of the eight characteristics of microservices: (a) the practice that may benefit the practitioners; (b) the possible pain and expense caused by a specific characteristic.

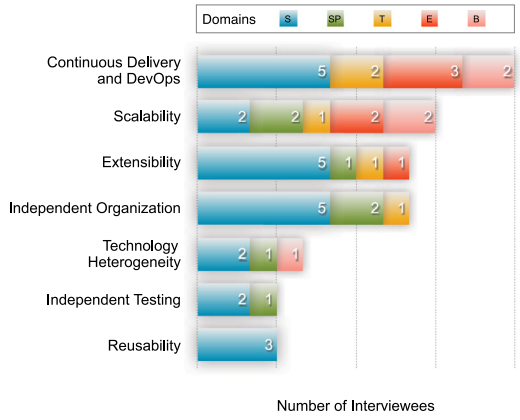

Fig. 3. Motivations and benefits related to componentization via the service.

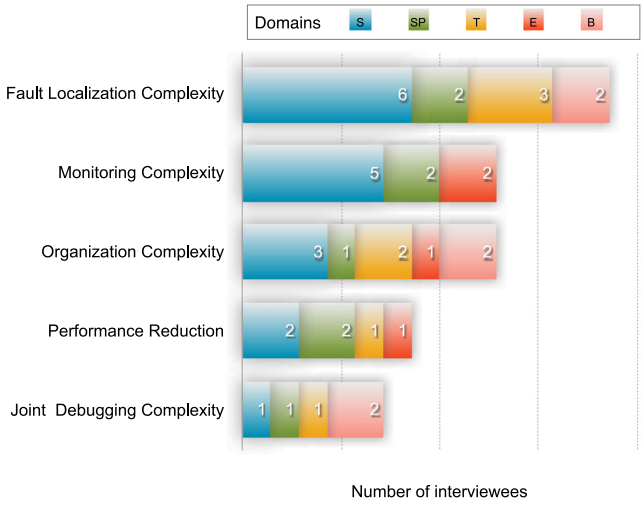

Fig. 4. Pains and expenses externalized by the chaotic independence of microservices.

4.1. Componentization via services

(1) Practice 1: ***Independence by separation***

Microservices are defined to be components out of the process that communicate with a mechanism such as a web service request or a remote procedure call (Fowler and Lewis, 2014). Overall, the underlying motivation and benefit of componentization are the independence or autonomy of microservices, which has been realized by all interviewees. One participant from B1 described the reason for adopting microservices as follows.

"Using microservices makes it possible to expand and scale some services independently, and save money and hardware resource costs while improving performance to a certain extent..."

Fig. 3 shows specific motivations of organizations related to componentization via service. *Continuous delivery and DevOps* are the most common motives claimed by 60% interviewees. Partitioning the monolith into a set of fine-grained components as microservices allows independent development and continuous deployment in one team, which would consequently facilitate the application of DevOps and help shorten the lead time of products in practice. Similarly, microservices are allowed to be *scaled up/down* easily (45%) and *extended/upgraded* independently (45%) due to their separation from each other. Only 15% interviewees (all from the software/IT domain) considered the *reusability* characteristic of microservices as the main motivation. Other motivations for the adoption of microservices are the reduction of *independent organization* (40%), the *technology heterogeneity* (20%), and *independent testing* (15%). It can be seen that *Extensibility* and *Independent Organization* are the main motivations considered by interviewees in the software/IT domain. *Scalability* and *Independent Organization* are the main motivations considered by interviewees in the IT Service Provider domain. *Continuous Delivery and DevOps* is the main motivation considered by interviewees in the Telecommunication domain and E-commerce domain. And *Continuous Delivery and DevOps* and *Scalability* are the main motivations considered by interviewees in the Bank and Finance domain.

(2) Pain 1: ***Chaotic***[1] ***independence***

However, interviewees do not always enjoy the benefits of componentization in MSA. Sometimes, the independence of microservices cannot be properly realized for various reasons, and then the gains shown in Fig. 3 may become pain and bring possible expenses in Fig. 4. First, inappropriate boundaries can decrease the availability of microservices. For example, the call chains of microservices will become longer and may burden *fault location* (65%), especially when faults are caused by different microservices simultaneously in long call chains. *Monitoring and Organization Complexities* are also two typical expenses claimed by almost half of the interviewees (45%). Moreover, the chaotic independence of microservices can reduce *performance* of systems due to increasing interaction and long call chain among distributed microservices (30%). As a concern of about 25% interviewees, the development or change of one feature can influence multiple microservices, which are distributed, developed, and deployed. Furthermore, microservices are frequently released and one microservice may have multiple versions, which can increase the complexity of *joint debugging* among different versions. Last but not least, developers may face requests for joint debugging from others at the same time. Some remote microservices are not always connectable in this process, which exacerbates the pain of testing. One participant from T2 nicely summarized:

"All these problems may seriously influence the speed of debugging and the efficiency of developments..."

The main pain claimed by interviewees in the software/IT domain and Telecommunication domain is *Fault Localization Complexity*. *Fault Localization Complexity*, *Monitoring Complexity* and *Performance Reduction* are the main pains considered by interviewees in the IT Service Provider domain. *Monitoring Complexity* is the main pain considered by interviewees in the E-commerce domain. Except for *Monitoring Complexity* and *Performance Reduction*, all other pains in Fig. 4 are considered by interviewees in the Bank and Finance domain.

4.2. Organized around business capabilities

(1) Practice 2: ***Organizational transformation***

According to Conway's law (Conway, 1968), the technology architecture and the organizational structure interrelate with each other. As suggested by Fowler and Lewis, teams of an *organization should be separated* with reference to the *business capability* of microservices and one product or component (based on microservices) are developed and operated by a dedicated team. 50% interviewees confirm this characteristic in their practice. They mentioned that microservices could drive the partition of teams, which reduces the dependency and communication among teams, and consequently lowers the cost of management.

[1] "Chaotic" represents that it is difficult to understand the internal behavior of the system, and it is difficult to improve the performance of service nodes in the call chain, thus affecting the overall situation.

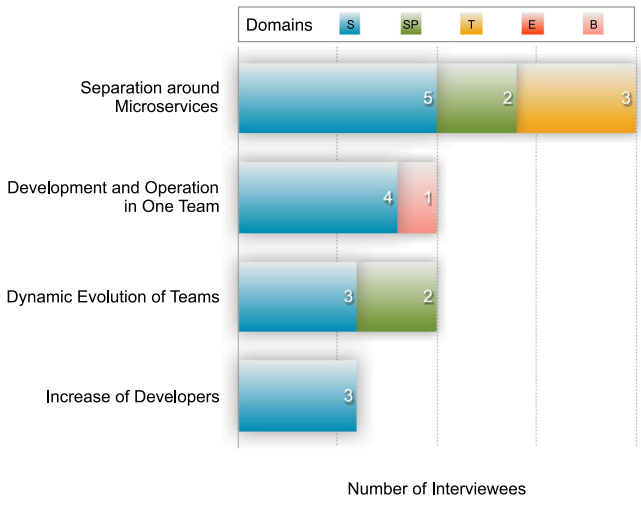

Fig. 5. Practices related to organizational transformation.

In contrast, the team structure can become a factor that influences the decision on the decomposition of microservice candidates, even the choice of technology architecture. Interviewees also take the team structure into account when decomposing the services of legacy systems or designing new microservices (see the quote from S1 below). However, respondents in the E-commerce domain did not mention any practices related to organizational transformation.

"The ideal is that the development of one microservice could be assigned to 1 to 2 developers. Having 3 to 5 developers to some extent indicates that the microservice can be further partitioned..."

Some common transformation practices adopted in the organizations of interviewees are displayed in Fig. 5: five interviewees including one from E-commerce company often *allocate both developers and operations engineer for each team* in MSA. However, this vision cannot be achieved in other organizations due to the shortage of personnel. In addition, *teams could vary dynamically* along with the evolution of microservices as mentioned by five interviewees, including two from the IT service providers domain. Once the development workload decreased, some team members may join other teams. Three software/IT interviewees mentioned that they experienced *the increase of developers* during transformation. The parallel development of microservices between different teams takes place in all interviewees' organizations, while one team may be responsible for developing multiple microservices.

(2) Pain 2: ***Ad-hoc organizational transformation***

Although drawing support from some strategies, organizations in this study still rely on human experience to transform themselves in an ad-hoc manner. As predicted by 25% interviewees, practitioners may suffer from inappropriate organization transformation along with migration to MSA. The mismatch between the organization structure and the architecture may cause low internal development efficiency or massive communication among teams. Because it is difficult to ensure the 100% independent decomposition of microservices, interactions may exist due to the coupling between microservices to a different extent. When debugging among the relevant microservices under this circumstance, it is doubtful whether the microservices could improve internal development efficiency. One participant from T1 explained his idea and requirement of this problem as follows:

"We are aware that the organization should catch up with the change of architecture accordingly, but we lack the guide on how to do it. Most importantly, we need the approval of the decision makers..."

4.3. Smart endpoints and dumb pipes

(1) Practice 3: ***Choosing communication protocol***

Compared to the Enterprise Service Bus (ESB) in SOA, Smart endpoints and dumb pipes are an alternative communication approach favored by the microservice community, the communication protocol of which commonly uses *HTTP-based RESTful API*, *RPC*, and *lightweight messaging*. Our interviews find that these protocols still form the mainstream in industry and have been applied by some well-known microservice frameworks. For example, `Spring Cloud`[2] based on the RESTful API is proved to be the most prevalent framework and therefore was adopted by 35% interviewees (as shown in Fig. 6) because it is easy to use and language independent.

Surprisingly, although Fowler and Lewis (2014) consider that "A naive conversion from in-memory method calls to RPC leads to chatty communications which don't perform well" and the biggest problem with the RPC protocol is that it cannot penetrate the firewall, there are still five interviewees using the `Apache Dubbo`[3] framework and one interviewee using `TARS`[4] whose communication protocols are all based on RPC. One profound disadvantage of the HTTP RESTful API protocol is its performance and the complexity of the operation. In contrast, communication through RPC only needs to transmit smaller data than the HTTP RESTful API and thus has higher performance. In addition to that, we also observe that the lightweight messaging frameworks `RabbitMQ`[5] and `Kafka`[6] are adopted by 10% and 5% interviewees, respectively. The main communication mechanisms claimed by interviewees in the software/IT domain and Bank and Finance domain are both HTTP RESTful API and RPC protocols. In addition to the above two protocols, lightweight messaging framework is also the selection considered by interviewees in the E-commerce domain. HTTP RESTful API protocol and lightweight messaging framework are the main selections considered by interviewees in the IT Service Provider domain. It is worth mentioning that no interviewee from the telecommunication domain is identified to use the aforementioned frameworks based on different communication protocols. One from T1 summarized:

"Many tools of telecommunication projects need independent research and development, and there are few tools to choose from..."

We asked the interviewees their reasons for why a framework or technology was selected. The answers are displayed in Fig. 7. Seven main factors concern practitioners are the maturity of the *technology's ecosystem* (55%), the activity of the *community* (50%), *previous experience* of the organization (40%), the *supporting documents* (30%), *performance or efficiency* (30%), *end-users' technology stack* requirement (20%), and just *following the trend* (20%). A technology's ecosystem, including the components or integrations it supports, may indicate whether it will develop well in the future. The activity of the supporting community and the richness of the documentation can determine whether the technology is easy to learn. Furthermore, about 23% interviewees considered open-source tools, with which they could easily carry out some secondary developments according to their requirements. Six interviewees preferred to select the technology stack with high performance and efficiency, for example, `Apache Dubbo` and `TARS` addressing performance problems using RPC communication protocols. All of the interviewees in the telecommunication domain made the decision according to the end-users' technology stack, since their products usually serve other communication operators

2 https://cloud.spring.io/.
3 http://dubbo.apache.org/.
4 https://tars.tencent.com/.
5 https://www.rabbitmq.com/.
6 http://kafka.apache.org/.

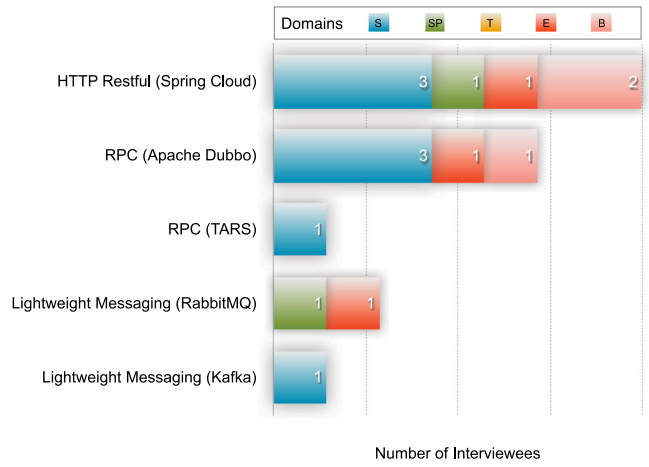

Fig. 6. Selection of communication protocols and frameworks for MSA in industry.

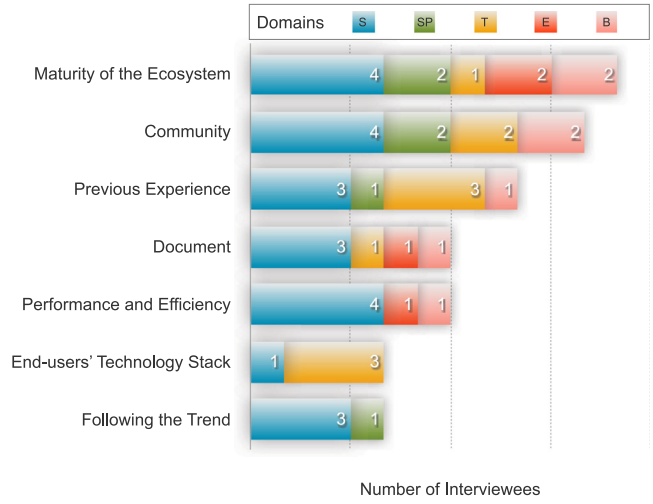

Fig. 7. Factors impacting the selection of the MSA framework in industry.

who have some restrictions about the technology stack's selection, for example, the language stack. Moreover, frameworks like `Spring Cloud` and `Apache Dubbo` are widely adopted in the organization of ten interviewees, because they provide multiple components necessary to develop MSA-based systems, for example, service discovery, API gateway, load balancer. Obviously, the impacting factors concerned in different domains are different. *Maturity of the Ecosystem* and *Community* are the main factors considered by interviewees in the IT Service Provider domain and Bank and Finance domain. In addition to the above two factors, *Performance and Efficiency* is also the main factor considered by interviewees in the software/IT domain. The impacting factors claimed by interviewees in the Telecommunication domain are *Previous Experience* and *End-users' Technology Stack*. *Maturity of the Ecosystem* is the main factor considered by interviewees in the E-commerce domain.

(2) Pain 3: ***Complexity of API management***

A serious pain related to the communications among microservices is the complexity of the implementation and management of API, as mentioned by 20% interviewees (4/20). The APIs provided in MSA are required to conform to the contract; otherwise, there would be problems in the interaction of services. As an example of this pain, a participant from T3 reported the following:

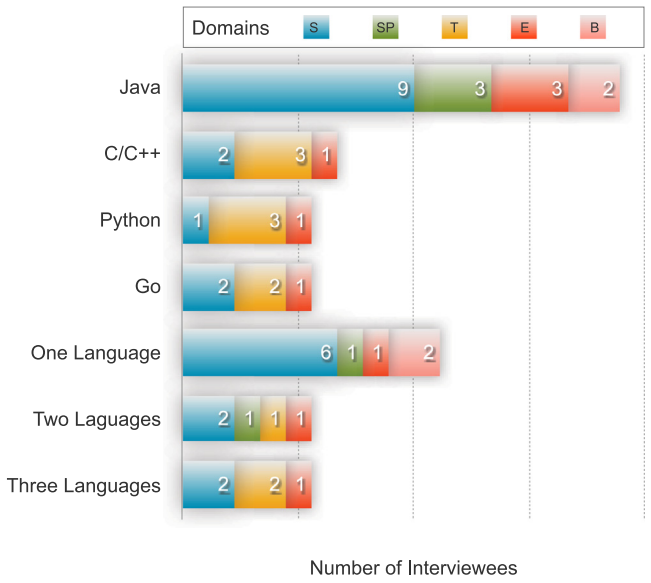

Fig. 8. Heterogeneous programming languages for decentralized microservices.

"We encountered the problems of implementing the repetitive interface, which was not addressed..."

According to the description of the other two interviewees, different teams may have inconsistent understanding on the APIs, and the problems may not be identified in time due to the independent releases of service. Two interviewees further complained that they had not found any effective way to ensure consistent understanding of APIs. Complex internal regulations were adopted in two participating companies, where they standardized the API development through some predefined rules, e.g., naming rules, and then some basic verification was manually conducted to ensure that the developers implement the API by following those rules defined.

4.4. Decentralized governance

(1) Practice 4: ***Support of technology diversity***

Different from the centralized governance in one single technology platform, microservices support a mix of multiple languages, development frameworks, and data-storage technologies, i.e. technology diversity. This was justified by the use of multiple programming languages and frameworks on the same system (as shown in Fig. 8). Java and One Language are the main programming languages considered by interviewees in the Bank and Finance domain. Moreover, the main programming language claimed by interviewees in the software/IT domain, IT Service Provider domain and E-commerce domain is also Java. C/C++ and Python are the main programming languages considered by interviewees in the Telecommunication domain. Among the interviewees, 50% used more than one programming language in their MSA-oriented systems, and Java is the most prevalent in the 65% interviewees. An interviewee from T3 well summarized the reasons for choosing Java:

"Java language technicians are easy to recruit, have high activity, and have rich technology stacks..."

Since different development languages are better suited for certain kinds of problem and developers may be proficient in dissimilar languages, MSA allows teams to choose more than one language according to their requirements, as mentioned by 50% interviewees. However, practitioners encourage controlling the diversity of technology stacks, since too diverse technology stacks may arouse many problems, e.g. versioning. Technology

selection decisions are generally made according to the considerations displayed in Fig. 7. For example, interviewees from the telecommunication domain prefer to select C, C++, and Go rather than Java due to the performance consideration.

(2) Pain 4: ***Excessive technology diversity***

As mentioned above, MSA can accommodate multiple technology stacks, which are not limited to the ones listed in Fig. 8. The diversity of technology stacks also raises the requirements on developers' ability and increases the complexity of learning and construction.

20% interviewees suggested that the high priority of technology architecture should be recognized. In other words, the design and the implementation of technology architecture can be done as early as possible, for example, at the beginning of considering migration towards MSA. Because learning new technologies and setting up technical frameworks may cost more time than expected. An interviewee from SP3 told us a relevant lesson they learned by saying:

"We thought it was easy, so we unhurriedly selected and constructed the technology stacks only after the microservices having been decomposed from the legacy system. As a result, no product was delivered in the following two months, because all people were busy coping with the problems of excessive technology stacks..."

4.5. Decentralized data management

(1) Practice 5: ***Compromise with database decomposition***

Although MSA advocates that each microservice manages its own database, we surprisingly identified that only three interviewees decomposed the database when migrating towards MSA. Accordingly, they experienced the pain of distributed transactions brought about by MSA. To alleviate this pain, two interviewees decomposed the databases vertically according to the coarse-grained business processes. A compromising strategy was also adopted by one interviewee's team: partial data tables were separated from the original database and owned by different microservices, while the rest were kept as a shared component. Some interviewees (60%) claimed that they did not consider decomposing the database for the following reasons:

- *Memory database in use*. Three interviewees from the telecommunication domain used the in-memory database rather than the traditional type on their legacy systems. Data are inherently tied to the relevant microservices; thus there is no need to decompose the database.
- *Databases partitioned in legacy systems*. Three interviewees expressed that they have implemented the database snapshot and separated data tables into different databases using technologies like Binlog. They thought this design is basically adequate for their requirements in the MSA-oriented systems, and it is unnecessary to further decompose the database into fine-grained ones.
- *Benefit analysis of decomposition*. As mentioned by six interviewees, although they were aware that decomposing the database contributes to decoupling the microservices, the problems (pains) after decomposition are prohibitive, such as the complexity of addressing data consistency. As a consequence, practitioners prefer to avoid possible troubles and persist in the shared database. Some traditional tactics are mentioned in their practices, for example, using caching to improve performance, and using locks or the data storage scheme `ETCD` to ensure data consistency.

Here is an interesting quote from B1 on compromise with database decomposition:

"For the front-end system, the database splitting is completed to a certain extent, while for the back-end system, the database splitting is not well considered. This is because the data are difficult to manage after decomposition, and there may be data consistency problems. Moreover, when the data volume is large, performance problems will be encountered, and the technical cost of decomposition the database, including maintenance and related development costs, is large..."

(2) Pain 5: ***Data inconsistence***

After migrating to MSA with multiple independent databases, practitioners are usually faced with the complexity of addressing distributed transactions while ensuring data consistency. Therefore, a few interviewees seriously considered and actually practiced the decomposition of the database (10%). One from E2 gave the following description about the pain of addressing data consistency.

"It troubles us for a long time. We need to consider different scenarios because you know that different scenarios have to do with different designs..."

MSA perplexes the inconsistency problems due to its distributed nature. Although the monolith is not free from this problem, it could be easily addressed by updating a bunch of things in a single transaction. Nevertheless, it becomes complex to handle this issue in different scenarios of MSA due to decentralized data management. As mentioned by an interviewee, most Internet companies prefer the 'eventual consistency of the distributed data to the strong consistency, especially when their requirements on performance or availability are higher than the data consistency (CAP). Although some methods are available for addressing the eventual consistency, e.g., messaging-based or event-based, their design and implementation are very complicated. Once the eventual consistency is taken, with the risk of some inconsistencies resulting from errors of a third party, to a great extent human intervention is required to get the final result manually and then make the data consistent rather than doing the rollback of data directly. The real process of addressing the data consistency is much more complicated than the description, which causes suffering for the practitioners after the migration to MSA.

4.6. Infrastructure automation

(1) Practice 6: ***CI & CD***

Fowler and Lewis argue that many teams building microservices possess rich experiences of Continuous Integration (CI) and Continuous Delivery (CD). When the infrastructure automation technology is applied, the deployment of microservices would be as simple as that of monolithic systems. To identify the practices of infrastructure automation in the organizations of the interviewees, we asked them about the automated tools being utilized in different phases of continuous delivery, such as source code management (SCM), Build & Configure, Test, and Deploy, etc.

Fig. 9 displays various automation technologies adopted by interviewees in the phase of developing microservices (Build & Config). `Git`[7] and `SVN`[8] are two commonly used tools in the SCM phase. `Jenkins`[9] gains the most popularity as a CI tool and also covers the Deploy phase, which is used by about 60% interviewees (12/20). `Docker`[10] and `Kubernetes`[11] are two typical `PaaS`[12] technologies with similar popularity in the Deploy phase. Practitioners prefer to develop their own tools for the

7 https://git-scm.com/.
8 http://subversion.apache.org/.
9 https://jenkins.io/.
10 https://www.docker.com/.
11 https://kubernetes.io/.
12 https://www.ibm.com/cloud/learn/paas.

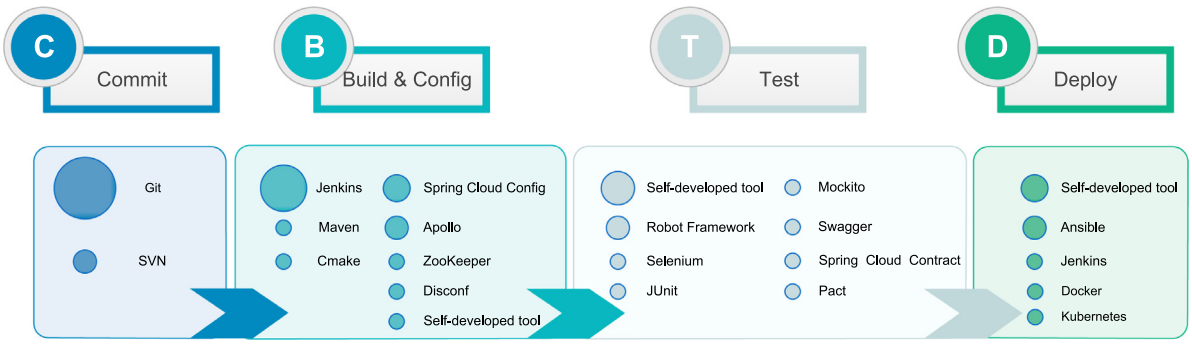

Fig. 9. Tool stacks used for infrastructure automation.

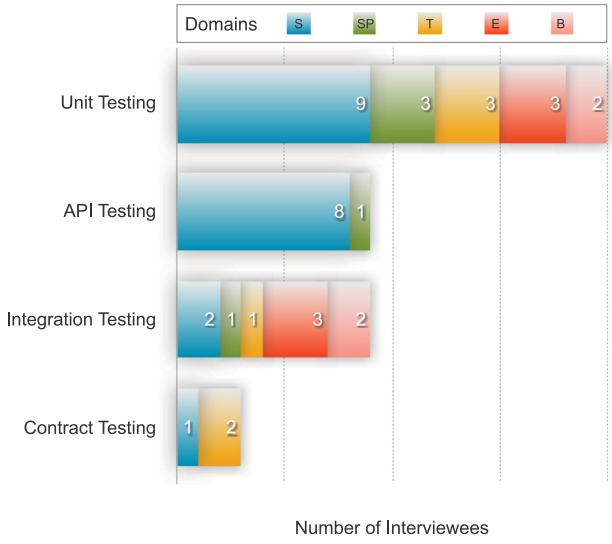

Fig. 10. Common testing tactics for MSA-based systems.

testing of microservices. `Robot Framework`[13] and `Selenium`[14] are also two popular tools used during the Test phase. Only one interviewee applied the new service mesh technology to facilitate the upgrade and the versioning of microservices (see quote from S2 below).

"The problem of service upgrade is very troublesome. Every time we upgrade a version, we need to perform integration tests. In addition, there is a problem in different languages. The service mesh technology can help us collect these functions..."

In addition, four testing tactics and the corresponding tools are mainly adopted by interviewees in practice (Fig. 10), namely unit testing, API testing, contract testing, and integration testing.

Unit testing All interviewees in microservices testing firstly use traditional unit testing. It means to exercise the small code units (usually classes) and determine whether they could run as expected. Regarding unit testing, some excellent testing tools and frameworks have already been provided to interviewees, such as `JUnit`.[15] In addition, mock tools are also necessary to simulate real scenarios, e.g., `Mockito`.[16]

API testing API testing concerns about the business unit/API of microservices, which was conducted by 45% interviewees to check the function completeness of the interfaces' implementation, internal logic, or address exceptions, etc. The interviewees believed that API testing has a high return-on-investment (ROI) ratio because the interfaces are relatively stable and it is easy to write test cases for API testing. When performing API testing, 35% interviewees used `Swagger Editor`[17] for the definition and design of the interface, `Swagger UI` [18] for the visualization of the information of the interface.

Integration testing The integration testing was adopted by 45% interviewees to verify the correctness of the function from the point of view of the users. Integration testing is an end-to-end process that incorporates user scenarios and data. Although this method verifies the complete process and has the highest business value, the accompanying complexity worries many interviewees. One practitioner recommends simplifying the integration testing to unit testing and API testing so that testing can be moved to the development stage and conducted as part of CI. In this way, potential problems can be identified as early as possible.

Contract testing Contract testing is aimed at verifying the correctness of connections/calls between microservices and checking whether the service provider's functionality can meet consumer requirements. Compared to integration testing, contract testing is lighter and faster because it decouples the development and testing process between API providers and consumers. API providers and the consumers only need to solidify the expectations of the API interface which they agree on through a "contract", for example, assuming what the provider will respond when receiving specific requests. After that, the testers of each group can independently test the contract according to their own development paces. In terms of contract testing, two popular supporting tools are `Spring Cloud Contract`[19] and `Pact`.[20] `Spring Cloud Contract` is a complete solution that helps users successfully implement consumer-driven contract methods and can integrate well with the Spring Cloud framework. However, it is complicated to some extent. In contrast, `Act` is lightweight and open source and supports multiple languages. In comparison with the three aforementioned testing tactics, contract testing is relatively new to practitioners in the MSA domain, therefore it only gains the attention of 20% interviewees.

Unit Testing is the main testing tactic considered by interviewees in the software/IT domain, IT Service Provider domain and Telecommunication domain. In addition to the above tactic, *Integration Testing* is also the main testing tactic considered by interviewees in the E-commerce domain and Bank and Finance domain.

13 https://robotframework.org/.
14 https://www.seleniumhq.org/.
15 https://junit.org/.
16 https://github.com/mockito/mockito.
17 http://editor.swagger.io/.
18 https://swagger.io/tools/swagger-ui/.
19 https://spring.io/projects/spring-cloud-contract.
20 https://github.com/pact-foundation/pact.io/.

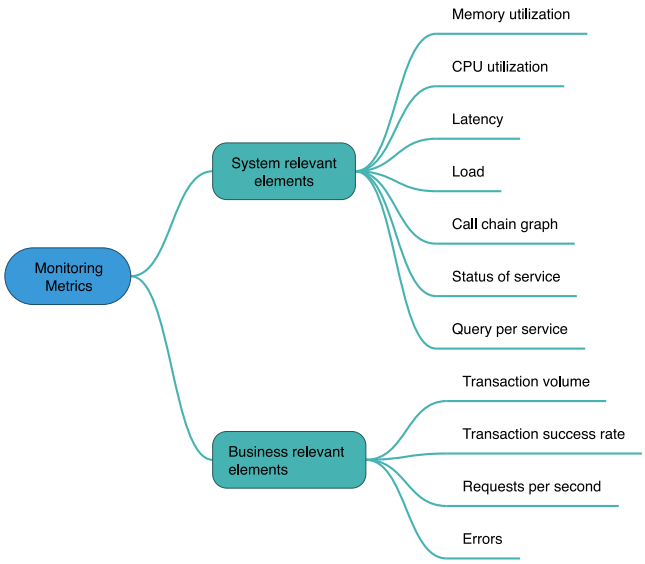

Fig. 11. Metrics related to the monitoring of MSA-based systems.

(2) Pain 6: ***Inadequate automation***

In terms of the construction of high-level automated infrastructure, 30% interviewees complained that many existing infrastructure-related technical frameworks are not satisfying their requirements. On the one hand, technologies and tools from abroad cost them a lot of effort and cost in setting up one by one. The relevant communities are not active enough, and incomplete documents can also make this pain more serious. On the other hand, as an important part of infrastructure automation, microservices governance brings difficulties to 15% interviewees in their practices, for example, microservices' auto-scaling in response to the peak flow and the automatic threshold setting rather than relying on the experience of architects. A participant from S5 reported the pain of inadequate automation as follows:

"Currently, most of the problems can be solved by tools, but in general, the degree of automation is not good enough. When there is a lot of data information to be detected, operation and maintenance personnel still need to make decisions..."

MSA brings many challenges to testing automation due to the following reasons, as mentioned by our interviewees:

High cost of communication In practice, microservices are often developed and maintained independently by different teams. When services are frequently changed or the previous versions of certain services are upgraded, it is easy to cause an incompatibility of microservices. Under this circumstance, testing automation may be more complex due to the increase of the communication cost between teams.

High cost of verification To verify the correctness of multiple microservices, testers need to build infrastructure (e.g., database and cache) for each service at high cost. Moreover, it is time-consuming to perform processes such as deployment and configuration to ensure that the service runs correctly.

High cost of coping with unstable results In distributed MSA-based systems, services communicate with each other through the network. However, factors such as network delay, timeout, and bandwidth can cause out-of-control test results.

High cost of integration testing Integration testing between APIs or microservices is not easy, even costly for interviewees. Compared to monolithic application, many units can be deployed independently under the microservice architecture. However, integration testing cannot be done alone, either as a provider of APIs or as a consumer. As a consumer, testing needs to start the provider's microservice, while it may already be the domain of other teams and vice versa. In general, the pace of development between a service provider and a service consumer may be inconsistent, resulting in unnecessary waiting times between each other. Therefore, the feedback period of microservices' integration testing is consequently longer than before.

4.7. Design for failure

(1) Practice 7: ***Microservices governance***

The characteristic of "Design for failure" characteristic emphasizes and relies on sophisticated governance setups for microservices: (a) monitoring and logging to prevent and locate faults; (b) fault tolerance mechanisms that recover microservices from faults. We asked the interviewees about their practices in these areas and accumulated some experience and practices in industry.

Monitoring and logging To help the localization of microservice faults, our interviewees adopted four common monitoring strategies: *log analysis*, *automatic alarming*, *call chain tracking*, and *dashboard*. Six interviewees (30%) said they used ELK stack (i.e. `Elasticsearch`,[21] `Logstash`,[22] and `Kibana`[23]) for the monitoring, search, and visualization of logs; nine (45%) indicated that the automatic alarming strategy was used in microservices' monitoring, while the system's setting threshold depends on previous experience of architects; seven (35%) paid special attention to troubleshooting and location of call chains of microservices, through using distributed tracing systems like `Zipkin`[24] and open-source APM tools like `Pinpoint`[25]; and seven (35%) replied to us that visual dashboards were relied on to monitor microservices, such as `Zabbix`,[26] `Grafana`,[27] `Kibana`, and self-developed tools.

Various metrics are checked and visualized through the dashboard in real-time monitoring of MSA-oriented applications. Fig. 11 displayed the major metrics that interviewees focused on. Among the 20 interviewees, ten mentioned that they paid attention to *system-relevant elements* of MSA-oriented systems, such as CPU, memory, and call chain graph of services; eleven mentioned that they monitored some *business-relevant metrics*, including transaction volume, transaction success rate, etc. `Prometheus`[28] is proved to be the most prominent monitoring tool through its application by 30% interviewees, as it to some content provides a unified solution for various metrics' monitoring, e.g., infrastructure, service, application, and even the common middleware.

Fault tolerance Apart from the faults' prevention and identification, fault tolerance, even recovery, is also important for implementing MSA-oriented systems with high availability. As a participant from T3 reported:

"Some performances of the upstream services of MSA systems are not easy to predict, so disaster tolerance drills are needed to judge how the upstream services react when the downstream services fail..."

21 https://github.com/elastic/elasticsearch/.
22 https://github.com/elastic/logstash/.
23 https://github.com/elastic/kibana/.
24 https://zipkin.apache.org/.
25 https://pinpoint.com/.
26 https://www.zabbix.com/.
27 https://grafana.com/.
28 https://prometheus.io/.

In MSA-oriented systems, microservices collaborate with each other when handling requests. That one basic service is down may cause cascading failures of other microservices throughout the system. Additionally, resources may be exhausted when a service consumer waits for the providers to respond. To address these problems, interviewees commonly adopted three fault tolerance patterns in industrial practices, namely: Circuit Breaker, Rate Limitation, and Degradation.

(a) *Circuit Breaker*. The circuit breaker pattern was used by 30% interviewees to mitigate the loss to the lowest level. In this pattern, an avoidance mechanism is added to the caller. When failures consecutively cross the threshold, a circuit breaker will be triggered and stop (fuse) the downstream request during the timeout period. After that period, the circuit breaker allows part of the test calls and resumes normal operation until these calls succeed. The most famous tool that supports *Circuit breaker pattern* is `Hystrix`.[29]

(b) *Rate Limiting*. The rate limit pattern was adopted by 35% interviewees. To use this pattern, the highest QPS (Query Per Second) threshold for each type of request needs to be set in advance. If the QPS of an MSA-oriented system is higher than the threshold predefined, the request will be directly returned without calling subsequent resources. Interviewees often consider two types of thresholds and set them in their practices, the service threshold and infrastructure threshold. With the aid of visualization tools such as `Kibana`, changes in monitoring data can be visually displayed. Specifically, the dramatic change may imply that there is a fault in the system. However, five interviewees complained that the configuration of the threshold is a complicated process, which relies on analyzing different scenarios and is difficult to automatically derive without human intervention.

(c) *Degradation*. The degradation pattern is similar to the *Circuit Breaker* pattern and was used by 30% interviewees. When using the *Degradation* pattern, decisions are made according to the load of the entire system. If the overall load of the entire MSA-oriented system exceeds or is expected to exceed a preset threshold, some non-core microservices may be degraded or handled in some specific way, for example, the fallback of `Hystrix`, to ensure the availability of core microservices.

(2) Pain 7: ***Unsatisfying monitoring and logging***

The distributed nature of microservices brings several challenges to monitoring MSA systems. All interviewees in our study expressed the pains caused by distributed microservices, which can be described in the following four aspects.

Insufficiency of automated operation When talking about automatic microservices monitoring, six interviewees (30%) complained to us that the current automation was not satisfactory. Some operation work still depends on manually checking the logging, which is time-consuming and inefficient.

Difficulty in troubleshooting A microservices-based system may include hundreds of instances of microservices, so it is often difficult for practitioners to locate problems among a large amount of distributed microservices. Five interviewees (25%) reported that they still relied on traditional troubleshooting methods, such as logging and distributed tracking.

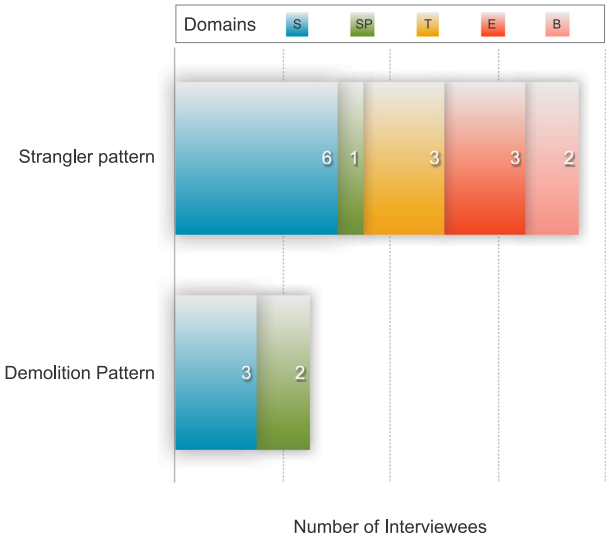

Fig. 12. Patterns of the evolution to MSA in practice.

Difficulty in threshold setting Setting appropriate alarm thresholds in MSA-oriented systems is also painful for many practitioners, which relies on previous experiences and lacks methodological guidance. The improper threshold settings are likely to be accompanied by subsequent misjudgments and omissions of anomalies or even faults. Moreover, the threshold setting requires continuous intelligent adjustments, which is being focused on by `AIOps`.[30]

No well-rounded monitoring system Microservices put forward new requirements for the monitoring and operation of the systems. Six interviewees (30%) have expressed that they wanted a monitoring platform that could provide microservice monitoring and visualization in multiple dimensions, including infrastructure, application, microservices, and middleware, to form an integrated monitoring scheme. They complained that it is difficult to switch between different monitoring platforms, which are only able to monitor partial metrics. `Prometheus`[31] is said to be powerful for the monitoring different kinds of metrics; however, it also has many drawbacks: e.g., no logging support, which is necessary to provide full visibility to the user's application; no persistent long-term storage, anomaly detection, automatic horizontal scaling, and user management, which are required in most enterprise environments; no dashboard solution and only giving a simple UI for PromQL queries, which relies on the operation of the graft and adds some extra setup complexity.

One participant from S2 nicely summarized:

"At present, monitoring MSA systems still relies heavily on the accumulated experience of architects, and the problems existing in the monitoring, such as misjudgment, missed judgment, automatic addition of monitoring, dynamic adjustment of threshold, and so on, have not been solved..."

4.8. Evolutionary design

(1) Practice 8: ***Stepwise evolution***

In terms of the evolution of microservices, two corresponding patterns are extracted from the experiences and practices of our interviewees (Fig. 12): strangler pattern and the demolition pattern. The demolition pattern refers to the one-step migration

29 https://github.com/Netflix/Hystrix/.

30 https://www.ibm.com/cloud/learn/aiops.

31 https://prometheus.io/.

to MSA from legacy systems or the development of new microservice systems regardless of old systems. This pattern is only adopted by 25% interviewees from the domain of software/IT and IT service providers, while the rest of the interviewees prefer the strangler pattern in practice. In other words, excepting that the main pattern of the evolution claimed by interviewee in the IT Service Provider domain is Demolition Pattern, all other four domains selected main pattern of the evolution is strangler pattern. Compared to the one-step demolition pattern, the strangler pattern is in common with the evolution characteristics of microservices described by Fowler and Lewis: the monolithic system should not suddenly migrate to MSA without transition, and the separation of microservices from the monolith should be done step by step. This pattern is appropriate for legacy systems that are old and difficult to change, especially for systems in the domain of Telecommunication, E-Commerce, and Bank. Strangler pattern adopters usually migrate their legacy systems to MSA by gradually replacing specific features with new microservices.

When applying the strangler pattern and migrating to MSA from legacy systems, there are two major concerns for practitioners: understanding legacy systems and decomposition oriented to microservices. Wolfart et al. (2021) proposed a roadmap for modernizing legacy monolithic systems with microservices. Carvalho et al. (2019a) explored how to extract configurable and reusable microservices from legacy systems. Based on the factors concerned in migration towards MSA of different application domains, Cao and Zhang (2022) proposed a decomposition method based on node-attributed network to provide generalized microservice candidates recommendations for different domains.

Understanding of legacy systems When microservices-based systems are migrated from legacy systems rather than developed from a green field design, it is necessary to understand the business logic and architecture of legacy systems to gain knowledge about microservices. From the interviews, we observed that experienced developers, architects, and design documents play an important role in completing this work. However, five interviewees mentioned that they experienced many difficulties in achieving this because legacy systems in their companies had been working for a long time with the absence of necessary design documents. For example, an interviewee from B1 described one of their legacy systems as follows:

"The system has been around for more than 30 years old. We have nothing but the puzzling source code. You cannot imagine how complex it is! It is hard for us to understand this monster, not to mention the decomposition of it".

In a circumstance like this, practitioners have no choice but to rely on costly manual code reading and analysis of the dependency among components. Moreover, the technology stacks of these systems are often out of date, so it is difficult to choose qualified reverse engineering tools that are available to improve the productivity of the time-consuming job.

Microservices-oriented decomposition This is to identify the boundary of microservices and determine the appropriate microservice candidates, as considered by 75% interviewees. The general principle of MSA-oriented decomposition is to achieve a suitable granularity of microservices considering different factors, e.g., scalability. The decision on which microservice should be separated from the monolith is made by experienced architects in most cases. Common practice is that architects or decision makers prioritize candidate microservices, according to one or more factors they may be most concerned about in Fig. 13, and then dissect the monster step by step.

Business domain is the most common factor considered in industry (55%), through which services are decomposed vertically according to business capabilities. Different business capabilities often correspond to different high-level domain objects, which are easy for practitioners to identify, e.g., the order management and the product management in the online store system. Businesses with a similar domain tend to be aligned with the same microservice, which reduces the coupling among them. Domain-Driven Design (DDD) from Evans (2004) is a popular method used by five interviewees to identify the boundary of microservices based on the factor of the business domain. DDD offers many methods to help practitioners analyze the business process of legacy systems, construct domain models and the bounded context, and finally determine the boundary of microservices. Popular DDD methods are *Brain Storming*, *Event Storming*, and *Four Color*. An interviewee applied the first two methods in decomposition and felt that the modeling process of *Event Storming* is more suitable for them. The time consumed and the results of the *Brain Storming* method usually depend on the degree of understanding of the legacy system. In contrast, *Event Storming* is more controllable, in which a structured process is defined to guide practitioners to obtain the aggregates. With the *Four Color* method, according to the business processes introduced by relevant people or collected through some operation tools, roles and roles' processes are analyzed and labeled by different colors to deliver subdomains related to microservice candidates.

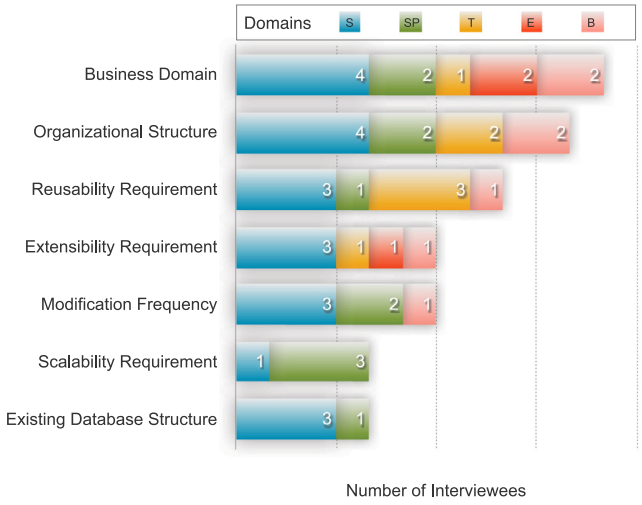

Fig. 13. Factors considered in the MSA-oriented decomposition.

Furthermore, 50% interviewees said that they took the *organizational structure* into account during MSA-oriented decomposition and tried to ensure that the original teams or developers still develop the identified microservices. This is because the structure of these organizations was originally set up based on different business modules. Reusability is also applied for extracting some common and core components by 40% interviewees. These core components, including the nonfunctional ones, are identified to be fundamental microservice candidates from the aspect of reusability. Furthermore, there are also other five important factors used for microservices-oriented decomposition: *extensibility requirement* (30%), *modification frequency* (30%), *scalability requirement* (20%), and *existing database structure* (20%).

It can be seen that monolith systems in different domains consider different decomposition factors when migrating to MSA. When implementing the MSA-oriented decomposition, we need to select the appropriate decomposition methods according to the factors that the current domain focused on, so as to generate high-quality microservice candidates.

(2) Pain 8: ***Subjective microservices-oriented decomposition without an effective guide***

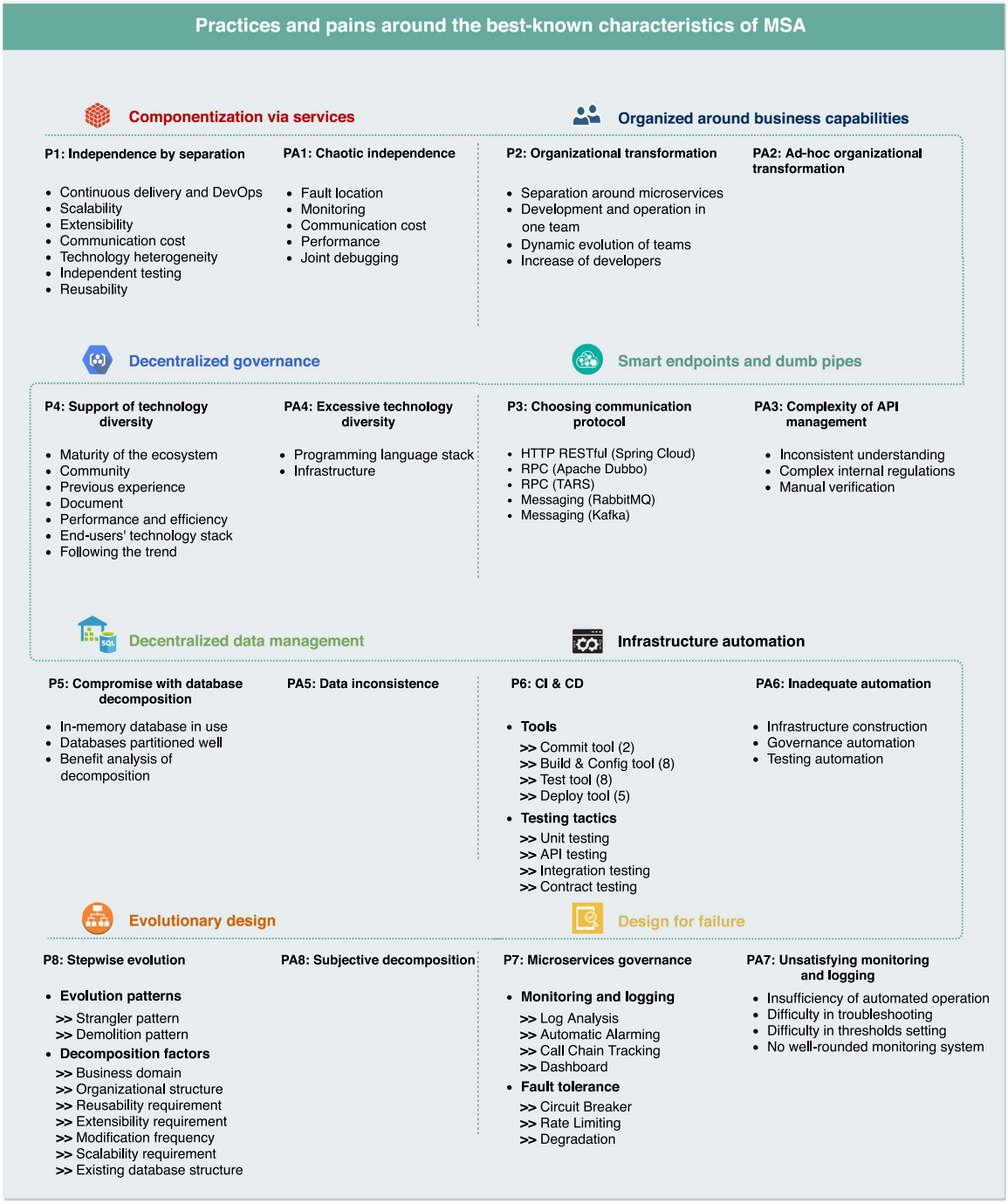

Fig. 14. The integrated overview map of practices (RQ1) and pains (RQ2) around the best-known characteristics of MSA.

During the design of MSA, subjective microservices-oriented decomposition due to the lack of an effective decomposition guide seems to be the most painful aspect for practitioners. In practice, it is the architect's experience that plays an important role in the process of decomposition. As mentioned by more than half of the interviewees (60%), they did not have a systematic guide on decomposition and the decision was usually made based on experience. It does not mean that they all agree to be guided by experience, as one respondent from S7 said below.

"Experience can tell us how to do it, but it can not tell us why we should do it..."

Although many theoretical strategies (e.g., vertical decomposition pattern Hasselbring, 2016 and DDD Evans, 2004) are attempted to be adopted in industry, they are not as effective and cannot meet the practical requirements of practitioners. For example, an interviewee from B2 complained about the problems of applying DDD in decomposition as follows.

"DDD is a thing relatively abstract to us, without a specific process guide. Since different people had different views on it, we spent a lot of time and effort to understand its utilization in practice and unify understanding."

4.9. Summary to answer RQ1 and RQ2

To our knowledge, this study is the first to take a unique angle based on the best-known characteristics claimed by Fowler and Lewis (2014), to conduct the largest interview study in practitioners with MSA experience. It finally identified eight pairs

of common practices (RQ1) and pains (RQ2) of microservices in industry after synthesizing the rich detailed data collected.

To answer the research questions more clearly, we integrated the eight pairs of common practices (RQ1) and pains (RQ2) around the characteristics claimed by Fowler and Lewis into an overview map, as shown in Fig. 14. Note that the characteristics are arranged using the claim sequence of Fowler and Lewis (2014), rather than chronological order. For each characteristic, the left part categorizes the common practices (P) commonly adopted in industry and the right part lists the pains (PA) complained about by the interviewees.

This overview map may contribute to providing practitioners with a systematic landscape on the practices and pains of MSA. When considering migration to MSA, practitioners can make their decisions combining the eight pairs of practices and pains related to the corresponding characteristic in Fig. 14. Specifically, in terms of "*componentization via services*", practitioners should first decide whether to migrate to MSA and separate their applications through microservices to achieve independence (P1). MSA has been justified to motivate our interviewees to do the separation from many aspects (cf. Fig. 3), for example, *Continuous delivery and DevOps*. Therefore, practitioners with these motivations can consider migration to MSA and gain the benefits of microservices due to independence by separation. However, the microservices-oriented separation and componentization should be carefully designed and properly realized to avoid chaotic independence (PA1), which may cause the aforementioned benefits to become pains and bring possible expenses in practice (cf. Fig. 4), e.g., *Fault location*.

5. Discussion

This section further discusses some implications for practitioners and researchers reflected by the findings in Section 4.

5.1. Implications to practitioners

The aforementioned complexities and pains externalized by MSA (cf. Section 4) demonstrate that this architectural style is not a free lunch and should not be embraced mindlessly by practitioners. Our interviewees have shown that adopting the microservice architecture may help reduce internal complexities of the legacy system and also brings many potential benefits (cf. P1 in Fig. 14), for example, continuous delivery and DevOps, scalability, and extensibility. However, in most cases, it merely changes the form of the complexities of monoliths rather than completely addressing them. In other words, the inner complexities of monoliths can simply transform into other external and visible complications (PA1–PA8) and become new pains for practitioners due to separation and componentization, especially in an inappropriate way. Consequently, decisions should be carefully made when embracing this architectural style.

Based on the evidence identified from the interviews of this study, i.e. the eight pairs of practices (P1–P8) and pains (PA1–PA8) of MSA that have been justified by our interviewees in the previous section, we further extracted five aspects that need careful decisions for the trade-off between possible benefits and pains of MSA before its adoption.

Decision 1: MSA or not

The first decision is whether or not practitioners should migrate to MSA. Although MSA has developed rapidly and gained momentum in recent years, any technology is not perfect, including MSA. Therefore, MSA should not be adopted blindly due to "everyone is doing it" and it must be comprehensively understood before deciding to migrate to MSA.

This study has provided the necessary information and experiences to help practitioners decide whether microservices are for them. The findings justify seven motivations and benefits related to componentization via microservice (cf. P1 in Section 4.1), that is, *Continuous delivery and DevOps*, *Scalability*, *Extensibility*, *Reducing the communication cost*, *Technology heterogeneity*, *Independent testing*, and *Reusability* (Fig. 3). However, the results also demonstrate that the componentization of microservices usually tends to be chaotic in practice and usually brings five common pains to practitioners (cf. PA1 in Section 4.1). The chaotic componentization and independence of microservices may require additional efforts and cost for practitioners to deal with complexities from many other aspects, i.e. *Fault location*, *Monitoring*, *Communication*, *Performance*, and *Joint debugging* (Fig. 4). Therefore, it is suggested that practitioners consider the trade-offs between the benefits and pains of MSA based on their own business drivers and then determine if it is really suitable for them to adopt MSA rather than just following the trend flippantly. When they decide to migrate to microservices, some reverse engineering technologies, such as Dehghani et al. (2022), would be helpful to understand and decompose old monoliths and improve migration efficiency.

Decision 2: Smaller size or not

Once practitioners decide to embrace MSA driven by their business goals, the following decision is about the size (granularity/boundary) of microservices, which may cause the trade-off between specific nonfunctional requirements of systems, e.g., scalability and performance, or extensibility and testability. As discussed in Decision 1, many advantages, such as high scalability and extensibility, may be achieved by decomposing a monolithic system into independent microservices with smaller granularity. However, it does not mean that the smaller the microservices are, the better. Decreasing the size of microservices may conversely increase the number of communication among distributed services and thus impact the performance of the application or cause the extra complexity of joint debugging and the excessive communication among teams.

Consequently, practitioners should carefully deal with the size of microservices and try to reduce the possible pains to a tolerable level. According to our findings P8 (see Section 4.8), considering one or more factors can be helpful to identify microservices with an appropriate size or limit: *Business domain*, *Organizational structure*, *Reusability requirement*, *Extensibility requirement*, *Modification frequency*, *Scalability requirement*, and *Existing database structure*, as shown in Fig. 13. Domain-Driven Design (DDD) is also the most common method mentioned by more than half of our interviewees (60%) to identify a suitable boundary of microservices considering the *Business domain* factor. In this method, it is necessary to align the bounded context and decompose the business domain according to the Single Responsibility Principle (SRP) to avoid frequent interactions caused by an over-fine-grained decomposition.

Decision 3: REST or not

The communication protocol of services is a basic decision that can affect the efficiency and stability of an MSA-based system. The results show that REST over HTTP(s) is the most prevalent option for the microservice communication protocol (cf. Section 4.3). The RESTful protocol represented by Spring Cloud has been shown to be easy to use and language independent. Basically, systems implemented in various development languages can accept RESTful requests and thus was adopted by 35% of our interviewees (cf. Fig. 6 for P3). However, due to the relatively low performance and complexity of API management (PA3) from a technical perspective, the HTTP RESTful API may sometimes be replaced by RPC in practice. The communication via RPC only needs to transmit data related to business content,

and the transmitted data is smaller and has higher performance than HTTP RESTful API. According to our observation, the RPC-based communication protocol was surprisingly considered by 30% interviewees in their practices.

Our findings in Section 4.3 indicate that practitioners should decide whether to use REST over HTTP or not based on the characteristic of their applications. We suggest adopting RPC to deal with high volumes of messages rather than HTTP/REST within an organization, especially when practitioners find latency or network saturation to be any sort of bottleneck. Otherwise, considering the advantages like less complexity of development and the ability to penetrate the firewall, REST over HTTP may be more suitable for practitioners. Moreover, it is better to select representative frameworks that support different communication protocols and provide multiple components necessary to develop MSA-based systems. For example, `Spring Cloud` for REST over HTTP and `Apache Dubbo` for RPC all provide basic components like service discovery, API gateway, and load balancing.

Decision 4: Diverse technologies or not

Diverse technologies or not is the fourth decision when adopting MSA. As described in Section 4.4, the decentralized governance of microservices supports their development using various languages, libraries, and databases. It means that MSA gives developers the freedom to select appropriate technologies based on their experiences or try new technologies to address specific problems (P4). However, our interviews find that dangers and issues may also come together with the benefits, e.g., versioning issues when upgrading microservices with different technologies (PA4). Therefore, the practice is that all of our interviewees in this study prefer to control the number of heterogeneous programming languages within three.

The above practices and pains may imply to practitioners that they should consider if too much technology diversity must be needed for their MSA-based applications. If not, the diversity of the technology should be controlled to find the right balance between improving the efficiency of addressing problems and reducing the possible pains that diverse technologies may cause. Furthermore, we also identified seven main factors that can be considered when selecting appropriate technologies (Fig. 7): *the maturity of the technology's ecosystem*, *the activity of the community*, *previous experience of the organization*, *the supporting documents*, *performance or efficiency*, *end-users' technology stack requirement*, and *just following the trend*.

Decision 5: Database per Service or not

Last but not least, we must decide whether to select the Database per Service pattern to manage the data database architecture of microservices. Database per Service is the most widely accepted pattern to manage the data in a microservices-based application, while our interviews reflect that this pattern should be carefully determined by practitioners. Database per Service pattern ensures that tables, schema, or database servers are split and privately owned by specific microservices. Through this pattern, transactions associated with a microservice involve only its database and can thus reduce the impact between microservices. This makes microservices more autonomous and deployable independently. However, as mentioned in Section 4.5, we find that more than 80% interviewees finally gave up the database decomposition over using a database for each service in practice (P5), because this pattern can cause complexities like data inconsistency (PA5), which might not be neglected by practitioners.

Our findings indicate that practitioners can adopt the Database per Service pattern if their requirement on data consistency is not very high, or they can deal with the difficulties of addressing inconsistency problems when implementing business transactions that span multiple microservices. Otherwise, compromises may be considered by using other alternative patterns or solutions, e.g., a partially shared database (P5).

5.2. Implications to researchers

Emerging findings, especially pains associated with MSA, may inspire new research directions. This section discusses five main topics worthy of further exploration and study by academic researchers.

(1) MSA-oriented decision models

Based on the feedback of interviewers and other cognition of microservice research, we believe that it is very necessary to form decision models to support better design, development, and even deployment of microservices in different enterprise contexts. The model can be constructed according to the metamodel used in the work of Ayas et al. (2021). At present, only a few studies focus on decision guidance for microservices, such as the process model for identifying service granularity in MSA-oriented migration (Schröer et al., 2021), and the decision guidance model for specific design patterns, such as service discovery (Haselböck et al., 2017) and microservice monitoring (Haselböck and Weinreich, 2017), were proposed, lack the fine-grained decision model that focuses on the flow of decision points and common options in terms of different phases when migrating to MSA. The generated decision guidance models may focus on the flow of decision points and common options in terms of different phases when migrating to MSA.

(2) Systematic evaluation and assessment

This study has observed that the adoption of MSA may incur and accumulate some challenges, pains, or even debts in terms of the development and operation of a microservices-based system. Although we tried to collect useful practices and condense necessary decisions to help practitioners relieve possible pains, most of the findings are qualitative and preliminary. Many problems are still obscure and worthy of in-depth study. For example, studies associated with a systematic prediction and evaluation of the potential impacts of MSA on systems may be helpful for practitioners to make their decisions when adopting MSA. Particularly, quantitative methods, which can visually model and measure the real impact of microservices on specific quality attributes, e.g., performance, testability, or scalability, are also needed to help practitioners verify their decisions.

Capuano and Muccini (2022) found that quality attributes play an important role in driving migration to MSA, but relevant research is still limited. Through the analysis of 72 selected primary studies related to the quality attributes of MSA, Li et al. (2021) sorted out the six key quality attributes associated with MSA in these papers, and proposed 19 tactics to quantitatively evaluate the quality attributes impacted by MSA. It contributes to the efficient decision making of software architects. However, how to trade off different quality attributes, improve, and evaluate other quality attributes such as maintainability in the process of adopting MSA needs more attention in the future.

(3) Empirical research on organizational transformation

Conway's law (Conway, 1968) indicates that the organizational structure influences the (product) architecture. Five interviewees also justified the inverse law, who mentioned that microservices had impacts on the decomposition of their teams. Three interviewees thought that the mismatch between the organization structure and the (product) architecture might slow down internal development and/or lead to extra communication overhead among teams. Through participant observations and interviews with practitioners in three different types of companies that adopted DevOps and microservices architecture, zhou et al. found that the problems faced by these companies, such as fragmented DevOps, came from organizational structure (Zhou et al., 2022). Based on empirical research, Cui et al. identified seven impacts of using microservice architecture on organizational structure based on empirical research and further proposed an assessment

framework to improve the adaptability of organization structure to microservice architecture (Cui et al., 2021).

In the practice of using microservices architecture, how to adjust the organizational structure to match the (product) architecture to achieve the expected benefits is a challenging problem. Based on the group role assignment (GRA) algorithm and E-CARGO model, Zhang et al. proposed three strategies to evaluate the organization structure adopting microservices, including performance first, positivity first and communication costs first, and then proposed two optimized role allocation strategies (Zhang et al., 2022). To some extent, this can help companies build corresponding team organizations in different application contexts. However, further research is still needed on how to dynamically adjust the organization structure in different scenarios during the development process, improve team communication and collaboration ability, and evaluate team organization performance.

(4) MSA-oriented decomposition & migration

Identifying the boundary of microservices and determining the appropriate granularity of the microservice is a critical concern for 65% interviewees. Microservices at inappropriate granularity levels or mistakenly separated may make the pain (P1—chaotic independence) even more serious and consequently result in many consecutive problems, e.g., the extra complexity of joint debugging and the excessive communication among teams.

However, as reported by our interviewees, they lack an effective guide on service decomposition (P8) and MSA-oriented migration and commonly rely on some experienced architects to make the decisions. Consequently, research is encouraged to focus on such MSA-oriented decomposition or migration methods/models with some specific concerns of practitioners taken into consideration, for example, the prioritization of different factors in Fig. 7, which is supposed to deliver different decomposition or migration strategies, including the boundary recommendation of microservices, benefit analysis and risk assessment of each decomposition strategy for a specific case. Based on the factors involved in migration towards MSA of different application domains, Cao and Zhang (2022) proposed a decomposition method based on node-attributed network to provide generalized microservice candidate recommendations for different domains. Jin et al. (2021) proposed a functionality-oriented service candidate identification (FoSCI) framework based on the analysis of the monolith's execution traces, and created a comprehensive measurement system to evaluate microservice candidates. This facilitates the decomposition of the monolith into microservices with good functionality, modularity, and evolvability.

The trade-off of key standards based on MSA, such as feature modularization and reducing network communication overhead, is a tricky problem in MSA-oriented migration. Assunção et al. (2021) proposed a redesign strategy based on legacy features code, which was fully discussed in an industrial case study. It adopted search-based optimization to deal with the conflict between different key standards and provided an architectural design of redesigned features. The semiautomatic method of data flow-driven MSA-oriented decomposition proposed by Li et al. (2019) takes into account database decomposition. However, there is still little attention to properly decomposing the database during the migration to MSA. How to choose appropriate solutions from different methods of MSA-oriented decomposition and migration patterns also needs further research.

(5) MSA bad smells & anti-patterns

Based on the synthesis of the interview content and the cognition of software quality research, we believe that the MSA bad smells and antipatterns need to be paid attention in the migration practice towards MSA to avoid bad practices. Bad smells of MSA are a symptom of poor design, which may reduce the maintainability and affect the understandability of code (Taibi and Lenarduzzi, 2018). Although MSA bad smells has always had a negative impact on software quality, its definition and category are still not clear. By analyzing the literature related to MSA bad smells, Ding and Zhang (2022) defined 22 types of MSA bad smells and proposed solutions to the problems caused by these smells in MSA-oriented migration. Arcelli Fontana et al. Arcelli Fontana et al. (2020) used the Arcan tool that can identify eight MSA bad smells to assess architectural erosion in an industrial environment. The μTOSCA toolchain proposed by Soldani et al. Soldani et al. (2021) provides support for automated analysis of applications based on MSA. However, how to detect MSA bad smells and how to refactor the architecture against the smells need further research.

MSA antipatterns are a bad realization in practice, which can be divided into organizational antipatterns and technical antipatterns (Taibi et al., 2020). Guilherme Luciano Donin (2022) identified specific antipatterns related to MSA from the antipatterns that have been cataloged in the literature, and proposed 10 strategies to mitigate these antipatterns before migrating monolith to MSA. However, how to apply the strategies to alleviate the antipatterns of MSA and evaluate their impact on MSA in practice still needs in-depth research.

6. Threats to validity

An extremely cautious attitude was taken to mitigate threats when we designed and conducted this study. Specifically, the main threats to validity and the corresponding mitigation strategies were considered when characterizing the target population, designing and drafting the questionnaire, conducting the interviews, and the phases of data collection and synthesis.

Construct Validity: The most recurrent threats in the questionnaire-based survey are considered the phrasing adopted to define the statements and questions in the questionnaire. To mitigate these threats, we first define and refine the questionnaire continuously with the final eight major overhauls and several minor alterations. Second, a pundit who is experienced in the research of unstructured interviews was invited to verify the validity and integrity of the questionnaire. Third, we conducted a pilot study with randomly selected interviewees and modified the questionnaire extensively. Furthermore, the characteristics used in this study may be a potential threat to the validity of the construction.

External Validity: Representativeness of the data may be a potential issue in our study. To migrate to this threat, all interviewees in our survey were obtained from prudent selection. Specifically, we invited interviewees who work for enterprises of five domains and seven different cities based on the careful selection of hundreds of interviewee candidates obtained from three famous industrial conferences. Varying the levels of experience of our interviewees ensure the generalizability of our research. Meanwhile, although the respondents are Chinese, most of the companies we selected respondents belong to are multinational companies, and the projects the respondents participated in are not only applicable to China. Moreover, while we discussed the business domains, contexts for MSA are complex and involve many other aspects. Hence, threats remains, which do however, not invalidate our results.

Reliability: To mitigate this threat, we exploited theme coding and inter-rater reliability assessment to limit observer bias and interpretation bias. To be specific, (1) a qualitative data analysis software, NVivo, was employed to help theme coding; (2) we also cross-analyzed the answers of different questions from every interviewer, and then compared every extracted data to find disagreements; (3) every disagreement found in data synthesis was discussed until resolved.

7. Related work

This study contributes to the knowledge and community of microservices through two specific ways: (1) it takes a unique angle different from others to study MSA, which compares the most influential statements of Fowler and Lewis with industrial practices after six years since their statements (Fowler and Lewis, 2014); (2) it is an interview study of 20 software companies, with rich detailed data collected from experienced practitioners.

With increasing interest in microservices, some systematic mapping studies related to MSA have been reported. Pahl and Jamshidi (2016) conducted a systematic mapping study on the motivation, type, techniques, and challenges of microservices. Focusing on taxonomically classifying and comparing their 21 identified studies, they proposed a characterization framework. Alshuqayran et al. (2016) reported another systematic mapping study on microservices. In order to present an overview of existing challenges, architectural diagrams/views, and quality assurance, 33 relevant studies were identified and analyzed in their review. Similar to the reviews mentioned above, Francesco et al. (2017) performed a systematic mapping study on the trends, focus and potential directions of MSA most recently. All these secondary reviews were based on academic studies and did not directly address migration to MSA. By contrast, our work involved industrial experience from practitioners and specifically focused on practices and challenges when and after adopting MSA.

There are several studies that directly investigate migration towards MSA. Through interviews with some professionals, Auer et al. (2021) identified a set consisting of information and metrics that companies can use to decide whether to migrate to microservices or not. Through a survey, Colanzi et al. (2021) identified the most anticipated benefits of driving migration to microservices. Chen (2018) observed the benefits and new challenges associated with MSA. He shared practical strategies that can be employed to address these new challenges, discussed situations for which MSA may not be a good choice, and outlined areas that require further research. However, all experiences and observations from Chen originate from a single company and only represent one data point. Luz et al. (2018) discussed the motivation, benefits, and challenges related to the migration from monolithic enterprise architecture to microservice-based architecture. Similar studies have Di Francesco et al. (2018), Taibi et al. (2017) and Baškarada et al. (2018), etc. The above studies did not cover the specific migration process. To overcome this limitation, Wolfart et al. (2021) proposed a roadmap to modernize legacy monolithic systems with microservices, which is distilled from the existing body of knowledge. Kirby et al. (2021) investigated the applicability and usefulness of different types of relationships when extracting microservices. Carvalho et al. (2019a) explored how to extract configurable and reusable microservices from legacy systems. Laigner et al. (2020) reported the experience of replacing traditional big data systems with event-driven systems based on microservices. Megargel et al. (2020) provided a comparison between monolithic and microservice styles of application architecture in the context of the perspective of practice and cloud computing vision, and proposed a method for the transition from monolithic to cloud-based microservices. Waseem et al. (2020) selected 47 primary studies on Microservice Architecture in DevOps to bring more specialized solutions to MSA problems in DevOps.

However, Carvalho et al. (2019b) founded that academic research on migration to MSA could not fully meet the needs of practitioners. Practitioners lacked specialized techniques, tools, and metrics to support industry ongoing assessment of service granularity and dependencies (Bogner et al., 2021). Schröer et al. (2021) proposed a process model to guide the identification and design of microservices. Haselböck et al. (2018) interviewed ten microservice domain experts to find out which design areas are relevant for microservices, how important they are, and why they are important. Tizzei et al. (2020) analyzed 19 microservices and evaluated how the microservice architecture supports software maintenance through empirical quantitative research on scientific applications built from scratch. Wang et al. (2021) collected and categorized best practices, challenges, and some solutions to the challenges of practitioners. Lenarduzzi et al. (2020) monitored the technical debt of an SME while it migrated from a legacy monolithic system to an ecosystem of microservices. But they only investigated an old project. Pigazzini et al. (2019) proposed a tool named Arcan to identify microservice candidates based on architectural smell detection. Avgeriou et al. (2021) compared different technical debt measurement tools to help detect different levels of architectural smell. Bogner et al. (2019) focused on applied technologies, microservice characteristics, and the perceived influence on software quality during migration. Our work differs from these studies in two ways: (1) we focused on the gaps between the visions and the industrial practices from the characteristic dimensions widely accepted both in academic and industrial area; (2) We also paid attention to the pains, the migration road map, and the potential research directions of all those dimensions, which were not discussed in much detail in the aforementioned studies.

Soldani et al. (2018) also identified gains and pains by analyzing industrial grey literature. Similar aspects are covered in the study by Knoche and Hasselbring (2019), which investigated the drivers, obstacles, and strategies to introduce microservices in the German industry. However, the analysis in these two studies can be seen as a similar but quantitative study among practitioners. By contrast, we performed in-depth interviews with industrial software professionals and investigated the industrial practices, new pains, and potential solutions of MSA. Furthermore, we also defined an action framework for migration to MSA based on the common practices identified in the industry through both quantitative and qualitative analysis.

8. Conclusion

Microservices is the latest trend in architectural style, which has attracted growing interest in industry and academia. Many initiatives have contributed to understanding various aspects of microservices, e.g., benefits and challenges. In contrast, this study takes a unique angle different from all other studies on MSA, comparing the most influential characteristic statements of MSA by Fowler and Lewis with industrial practices. We aim to gain an in-depth understanding of the gaps between these community-wide accepted visions and the real practices of microservices, including the practical experiences with specific ideal characteristics of microservices, the reasons behind the gap, and the pains that should be mitigated.

An empirical study was carried out applying semi-structured interviews to collect data from 20 experienced interviewees in a variety of domains of the software industry. Based on the data collected, we characterized the gaps in terms of the best-known characteristics of microservices. Furthermore, the trade-offs between benefits and suffering of microservices in these characteristics-related dimensions were analyzed. The findings of industrial experiences related to these grand visions of microservices will benefit practitioners for their exercises in adopting and implementing microservices. In addition, the pains associated with some aspects of microservices, e.g., organizational transformation, decomposition, monitoring, and logging, may also inspire researchers with further research directions. Then we proposed an integrated overview map of the practices and pains around

the best-known characteristics of MSA to help the understanding and the decisions of practitioners, especially newcomers in this domain. Last but not least, this study condensed five decisions that should be carefully made by practitioners and three potential directions worthy of further research in academia.

In the next stage, we plan to closely work with more practitioners to define an exaction framework that guides evaluation and migration to MSA. Our future research may also include a grey literature review and a questionnaire-based survey in a larger population to validate and generalize the critical findings.

CRediT authorship contribution statement

Xin Zhou: Conceptualization, Methodology, Writing – original draft. **Shanshan Li:** Conceptualization, Methodology, Writing – original draft. **Lingli Cao:** Writing – review & editing. **He Zhang:** Conceptualization, Methodology. **Zijia Jia:** Visualization, Investigation. **Chenxing Zhong:** Visualization, Investigation. **Zhihao Shan:** Visualization, Investigation. **Muhammad Ali Babar:** Editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is jointly supported by the National Key Research and Development Program of China (No. 2019YFE0105500) and the Research Council of Norway (No. 309494), the Key Research and Development Program of Jiangsu Province (No. BE2021002-2), the National Natural Science Foundation of China (Grants No. 62072227, 62202219), as well as the Intergovernmental Bilateral Innovation Project of Jiangsu Province (No.BZ2020017).

References

Alshuqayran, N., Ali, N., Evans, R., 2016. A systematic mapping study in microservice architecture. In: Proceedings of the 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA). IEEE, pp. 44–51.

Arcelli Fontana, F., Locatelli, F., Ilaria, P., Mereghetti, P., 2020. An Architectural Smell Evaluation in an Industrial Context. In: ICSEA 2020, ISBN: 978-1-61208-827-3, pp. 68–74, https://www.researchgate.net/publication/346965175_ICSEA_2020_The_Fifteenth_International_Conference_on_Software_Engineering_Advances#page=79.

Assunção, W.K.G., Colanzi, T.E., Carvalho, L., Pereira, J.A., Garcia, A., de Lima, M.J., Lucena, C., 2021. A multi-criteria strategy for redesigning legacy features as microservices: An industrial case study. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) pp. 377–387. http://dx.doi.org/10.1109/SANER50967.2021.00042.

Auer, F., Lenarduzzi, V., Felderer, M., Taibi, D., 2021. From monolithic systems to Microservices: An assessment framework. Inf. Softw. Technol. 137, 106600. http://dx.doi.org/10.1016/j.infsof.2021.106600.

Avgeriou, P.C., Taibi, D., Ampatzoglou, A., Arcelli Fontana, F., Besker, T., Chatzigeorgiou, A., Lenarduzzi, V., Martini, A., Moschou, A., Pigazzini, I., Saarimaki, N., Sas, D.D., de Toledo, S.S., Tsintzira, A.A., 2021. An overview and comparison of technical debt measurement tools. IEEE Softw. 38 (3), 61–71. http://dx.doi.org/10.1109/MS.2020.3024958.

Ayas, H.M., Leitner, P., Hebig, R., 2021. Facing the giant: A grounded theory study of decision-making in microservices migrations. In: Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). Association for Computing Machinery, New York, NY, USA.

Baškarada, S., Nguyen, V., Koronios, A., 2018. Architecting microservices: Practical opportunities and challenges. J. Comput. Inf. Syst. 1–9.

Bogner, J., Fritzsch, J., Wagner, S., Zimmermann, A., 2019. Microservices in industry: Insights into technologies, characteristics, and software quality. In: Proceedings of the 2019 IEEE International Conference on Software Architecture Companion (ICSA-C). IEEE, pp. 187–195.

Bogner, J., Fritzsch, J., Wagner, S., Zimmermann, A., 2021. Industry practices and challenges for the evolvability assurance of microservices. Empir. Softw. Eng. 26 (5), 104. http://dx.doi.org/10.1007/s10664-021-09999-9.

Cao, L., Zhang, C., 2022. Implementation of domain-oriented microservices decomposition based on node-attributed network. In: 2022 11th International Conference on Software and Computer Applications. In: ICSCA 2022, Association for Computing Machinery, New York, NY, USA, pp. 136–142. http://dx.doi.org/10.1145/3524304.3524325.

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

Carrasco, A., Bladel, B.v., Demeyer, S., 2018. Migrating towards microservices: migration and architecture smells. In: Proceedings of the 2nd International Workshop on Refactoring. ACM, pp. 1–6.

Carvalho, L., Garcia, A., Assunção, W.K.G., Bonifácio, R., Tizzei, L.P., Colanzi, T.E., 2019a. Extraction of Configurable and Reusable Microservices from Legacy Systems: An Exploratory Study. SPLC '19, Association for Computing Machinery, New York, NY, USA, pp. 26–31. http://dx.doi.org/10.1145/3336294.3336319.

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

Chen, L., 2018. Microservices: Architecting for continuous delivery and DevOps. In: Proceedings of the 2018 IEEE International Conference on Software Architecture (ICSA). IEEE, pp. 39–46.

Chen, R., Li, S., Li, Z., 2017. From monolith to microservices: A dataflow-driven approach. In: Proceedings of the 2017 24th Asia-Pacific Software Engineering Conference (APSEC). IEEE, pp. 466–475.

Colanzi, T., Amaral, A., Assunção, W., Zavadski, A., Tanno, D., Garcia, A., Lucena, C., 2021. Are we speaking the industry language? The practice and literature of modernizing legacy systems with microservices. In: 15th Brazilian Symposium on Software Components, Architectures, and Reuse. SBCARS '21, Association for Computing Machinery, New York, NY, USA, pp. 61–70. http://dx.doi.org/10.1145/3483899.3483904.

Conway, M.E., 1968. How do committees invent. Datamation 14 (4), 28–31.

Cui, H., Zhang, C., Ding, X., Cao, L., Yang, Y., 2021. Evaluation framework for development organizations'adaptability to microservices architecture. J. Softw. 32 (5), 1256. http://dx.doi.org/10.13328/j.cnki.jos.006232.

De Alwis, A.A.C., Barros, A., Polyvyanyy, A., Fidge, C., 2018. Function-splitting heuristics for discovery of microservices in enterprise systems. In: Proceedings of the International Conference on Service-Oriented Computing. Springer, Cham, pp. 37–53.

Dehghani, M., Kolahdouz-Rahimi, S., Tisi, M., Tamzalit, D., 2022. Facilitating the migration to the microservice architecture via model-driven reverse engineering and reinforcement learning. Softw. Syst. Model. 21 (3), 1115–1133. http://dx.doi.org/10.1007/s10270-022-00977-3.

Di Francesco, P., Lago, P., Malavolta, I., 2018. Migrating towards microservice architectures: an industrial survey. In: Proceedings of the 2018 IEEE International Conference on Software Architecture (ICSA). IEEE, pp. 29–38.

Ding, X., Zhang, C., 2022. How Can We Cope with the Impact of Microservice Architecture Smells?. In: ICSCA 2022, Association for Computing Machinery, New York, NY, USA, pp. 8–14. http://dx.doi.org/10.1145/3524304.3524306.

Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R., Safina, L., 2017. Microservices: yesterday, today, and tomorrow. Present Ulterior Softw. Eng. 195–216.

Eski, S., Buzluca, F., 2018. An automatic extraction approach: transition to microservices architecture from monolithic application. In: Proceedings of the 19th International Conference on Agile Software Development: Companion. ACM, pp. 1–6.

Evans, E., 2004. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional.

Fowler, M., Lewis, J., 2014. Microservices - A definition of this new architectural term. URL https://martinfowler.com/articles/microservices.html.

Francesco, P.D., Malavolta, I., Lago, P., 2017. Research on architecting microservices: Trends, focus, and potential for industrial adoption. In: Proceedings of the 2017 IEEE International Conference on Software Architecture (ICSA). IEEE, pp. 21–30.

Furda, A., Fidge, C., Zimmermann, O., Kelly, W., Barros, A., 2017. Migrating enterprise legacy source code to microservices: On multi-tenancy, statefulness and data consistency. IEEE Softw. 35 (3), 63–72.

Guilherme Luciano Donin, V., 2022. Strategies to Mitigate Anti-Patterns in Microservices Before Migrating from a Monolithic System to Microservices. Centro de Ciências Exatas e Tecnológicas, URL https://tede.unioeste.br/handle/tede/6034.

Haselböck, S., Weinreich, R., 2017. Decision guidance models for microservice monitoring. In: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW). pp. 54–61. http://dx.doi.org/10.1109/ICSAW.2017.31.

Haselböck, S., Weinreich, R., Buchgeher, G., 2017. Decision guidance models for microservices: service discovery and fault tolerance. In: Proceedings of the Fifth European Conference on the Engineering of Computer-Based Systems. ECBS '17, Association for Computing Machinery, New York, NY, USA, http://dx.doi.org/10.1145/3123779.3123804.

Haselböck, S., Weinreich, R., Buchgeher, G., 2018. An expert interview study on areas of microservice design. In: Proceedings of the 2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA). IEEE, pp. 137–144.

Hassan, S., Bahsoon, R., 2016. Microservices and their design trade-offs: A self-adaptive roadmap. In: Proceedings of the 2016 IEEE International Conference on Services Computing (SCC). IEEE, pp. 813–818.

Hasselbring, W., 2016. Microservices for Scalability: Keynote Talk Abstract. ICPE '16, Association for Computing Machinery, New York, NY, USA, http://dx.doi.org/10.1145/2851553.2858659.

Hasselbring, W., Steinacker, G., 2017. Microservice architectures for scalability, agility and reliability in e-commerce. In: Proceedings of the 2017 IEEE International Conference on Software Architecture Workshops (ICSAW). IEEE, pp. 243–246.

Jamshidi, P., Pahl, C., Mendonça, N.C., Lewis, J., Tilkov, S., 2018. Microservices: The journey so far and challenges ahead. IEEE Softw. 35 (3), 24–35.

Jin, W., Liu, T., Cai, Y., Kazman, R., Mo, R., Zheng, Q., 2021. Service candidate identification from monolithic systems based on execution traces. IEEE Trans. Softw. Eng. 47 (5), 987–1007. http://dx.doi.org/10.1109/TSE.2019.2910531.

Jin, W., Liu, T., Zheng, Q., Cui, D., Cai, Y., 2018. Functionality-oriented microservice extraction based on execution trace clustering. In: Proceedings of the 2018 IEEE International Conference on Web Services (ICWS). IEEE pp. 211–218.

Kirby, L.J., Boerstra, E., Anderson, Z.J., Rubin, J., 2021. Weighing the evidence: On relationship types in microservice extraction. In: 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC). pp. 358–368. http://dx.doi.org/10.1109/ICPC52881.2021.00041.

Knoche, H., Hasselbring, W., 2019. Drivers and barriers for microservice adoption–A survey among professionals in Germany. Enterpr. Model. Inf. Syst. Archit. (EMISAJ)–Int. J. Concept. Model. 14 (1), 1–35.

Kvale, S., 2008. Doing Interviews. Sage.

Laigner, R., Kalinowski, M., Diniz, P., Barros, L., Cassino, C., Lemos, M., Arruda, D., Lifschitz, S., Zhou, Y., 2020. From a monolithic big data system to a microservices event-driven architecture. In: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). pp. 213–220. http://dx.doi.org/10.1109/SEAA51224.2020.00045.

Lenarduzzi, V., Lomio, F., Saarimäki, N., Taibi, D., 2020. Does migrating a monolithic system to microservices decrease the technical debt? J. Syst. Softw. 169, 110710. http://dx.doi.org/10.1016/j.jss.2020.110710.

Li, S., Zhang, H., Jia, Z., Li, Z., Zhang, C., Li, J., Gao, Q., Ge, J., Shan, Z., 2019. A dataflow-driven approach to identifying microservices from monolithic applications. J. Syst. Softw. 157, 1–16.

Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., Shan, Z., Shen, J., Babar, M.A., 2021. Understanding and addressing quality attributes of microservices architecture: A systematic literature review. Inf. Softw. Technol. (ISSN: 0950-5849) 131, 106449. http://dx.doi.org/10.1016/j.infsof.2020.106449, URL https://www.sciencedirect.com/science/article/pii/S0950584920301993.

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

Mazlami, G., Cito, J., Leitner, P., 2017. Extraction of microservices from monolithic software architectures. In: Proceedings of the 2017 IEEE International Conference on Web Services (ICWS). IEEE, pp. 524–531.

Megargel, A., Shankararaman, V., Walker, D.K., 2020. Migrating from monoliths to cloud-based microservices: A banking industry example. In: Ramachandran, M., Mahmood, Z. (Eds.), Software Engineering in the Era of Cloud Computing. Springer International Publishing, Cham, pp. 85–108.

Neri, D., Soldani, J., Zimmermann, O., Brogi, A., 2019. Design principles, architectural smells and refactorings for microservices: a multivocal review. SICS Softw.-Intensiv. Cyber-Phys. Syst. 1–13.

Newman, S., 2015. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, Inc..

Pahl, C., Jamshidi, P., 2016. Microservices: A systematic mapping study. In: Proceedings of the 6th International Conference on Cloud Computing and Services Science. ACM, pp. 137–146.

Patton, M.Q., 1990. Qualitative Evaluation and Research Methods. SAGE Publications, Inc.

Pigazzini, I., Arcelli Fontana, F., Maggioni, A., 2019. Tool support for the migration to microservice architecture: An industrial case study. In: Bures, T., Duchien, L., Inverardi, P. (Eds.), Software Architecture. Springer International Publishing, Cham, pp. 247–263.

Ren, Z., Wang, W., Wu, G., Gao, C., Chen, W., Wei, J., Huang, T., 2018. Migrating web applications from monolithic structure to microservices architecture. In: Proceedings of the 10th Asia-Pacific Symposium on Internetware. ACM pp. 1–10.

Saldaña, J., 2015. The Coding Manual for Qualitative Researchers. Sage.

Sayara, A., Towhid, M.S., Hossain, M.S., 2017. A probabilistic approach for obtaining an optimized number of services using weighted matrix and multidimensional scaling. In: Proceedings of the 2017 20th International Conference of Computer and Information Technology (ICCIT). IEEE, pp. 1–6.

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

Seaman, C.B., 1999. Qualitative methods in empirical studies of software engineering. IEEE Trans. Softw. Eng. 25 (4), 557–572.

Soldani, J., Muntoni, G., Neri, D., Brogi, A., 2021. The μtosca toolchain: mining, analyzing, and refactoring microservice-based architectures. Software: Practice and Experience 51 (7), 1591–1621. http://dx.doi.org/10.1002/spe.2974.

Soldani, J., Tamburri, D.A., Heuvel, W.-J.V.D., 2018. The pains and gains of microservices: A systematic grey literature review. J. Syst. Softw. 146, 215–232.

Taibi, D., Lenarduzzi, V., 2018. On the definition of microservice bad smells. IEEE Softw. 35 (3), 56–62. http://dx.doi.org/10.1109/MS.2018.2141031.

Taibi, D., Lenarduzzi, V., Pahl, C., 2017. Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. IEEE Cloud Comput. 4 (5), 22–32.

Taibi, D., Lenarduzzi, V., Pahl, C., 2020. Microservices anti-patterns: A taxonomy. In: Bucchiarone, A., Dragoni, N., Dustdar, S., Lago, P., Mazzara, M., Rivera, V., Sadovykh, A. (Eds.), Microservices: Science and Engineering. Springer International Publishing, Cham, pp. 111–128. http://dx.doi.org/10.1007/978-3-030-31646-4_5.

Taibi, D., Systä, K., 2019. From monolithic systems to microservices: a decomposition framework based on process mining. In: Proceedings of the 9th International Conference on Cloud Computing and Services Science. SciTePress, pp. 153–164.

Tizzei, L.P., Azevedo, L., Soares, E., Thiago, R., Costa, R., 2020. On the maintenance of a scientific application based on microservices: an experience report. In: 2020 IEEE International Conference on Web Services (ICWS). pp. 102–109. http://dx.doi.org/10.1109/ICWS49710.2020.00021.

Tyszberowicz, S., Heinrich, R., Liu, B., Liu, Z., 2018. Identifying microservices using functional decomposition. In: Proceedings of the International Symposium on Dependable Software Engineering: Theories, Tools, and Applications. Springer, Cham, pp. 50–65.

Wang, Y., Kadiyala, H., Rubin, J., 2021. Promises and challenges of microservices: an exploratory study. Empir. Softw. Eng. 26 (4), 63. http://dx.doi.org/10.1007/s10664-020-09910-y.

Waseem, M., Liang, P., Shahin, M., 2020. A systematic mapping study on microservices architecture in DevOps. J. Syst. Softw. 170, 110798. http://dx.doi.org/10.1016/j.jss.2020.110798.

Welsh, E., 2002. Dealing with data: Using NVivo in the qualitative data analysis process. Forum Qual. Sozialforschung / Forum: Qual. Soc. Res. 3 (2).

Wolfart, D., Assunção, W.K.G., da Silva, I.F., Domingos, D.C.P., Schmeing, E., Villaca, G.L.D., Paza, D.d.N., 2021. Modernizing legacy systems with microservices: A roadmap. In: Evaluation and Assessment in Software Engineering. In: EASE 2021, Association for Computing Machinery, New York, NY, USA, pp. 149–159. http://dx.doi.org/10.1145/3463274.3463334.

Zhang, C., Cui, H., Cao, L., Wang, F., Yang, Y., 2022. Group role assignment strategies in microservices team based on E-CARGO model. Knowl.-Based Syst. 248, 108910. http://dx.doi.org/10.1016/j.knosys.2022.108910, URL https://www.sciencedirect.com/science/article/pii/S095070512200435X.

Zhang, H., Li, S., Jia, Z., Zhong, C., Zhang, C., 2019. Microservice architecture in reality: An industrial inquiry. In: Proceedings of the 2019 IEEE International Conference on Software Architecture (ICSA). IEEE, pp. 51–60.

Zhou, X., Huang, H., Zhang, H., Huang, X., Shao, D., Zhong, C., 2022. A cross-company ethnographic study on software teams for devops and microservices: organization, benefits, and issues. In: 2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). pp. 1–10. http://dx.doi.org/10.1145/3510457.3513054.

Zimmermann, O., 2017. Microservices tenets. Comput. Sci. Res. Dev. 32 (3–4), 301–310.

Xin Zhou is now a Ph.D. candidate at the Nanjing University. He was awarded M.PM from Nanjing University. He has published over 10 peer-reviewed papers in high quality international conferences and journals (e.g. ICSE/FSE/IST/JSS), and won a Best Paper Award from APSEC2016.

Shanshan Li is an assistant researcher at the Nanjing University. She was awarded Ph.D.from Nanjing University. She has published many papers in IST, JSS, EASE, APSEC, ICSA and other high quality international conferences and journals.

Lingli Cao received the Master degree in software engineering from Anhui University, Hefei, China, in 2022. Her research interests include microservices architecture and evidence-based software engineering.

He Zhang is a Full Professor of Software Engineering and the Director of DevOps+ Research Laboratory at the Nanjing University, China, also a Principal Scientist with CSIRO, Australia. He was awarded B.Eng (NUAA), M.PM (USyd) and Ph.D. (UNSW). Prof. Zhang is an internationally recognized researcher and expert in software engineering. He joined academia after seven years working in software industry, developing software systems in the areas of aerospace and complex data management. He undertakes research in software engineering, in particular software process (modeling, simulation, analytics and improvement), software architecture, software security, blockchain-oriented software engineering, empirical and evidence-based software engineering, service-oriented computing, and data-driven software engineering. He has published over 150 peer-reviewed papers in high quality international conferences and journals, and won 11 Best/Distinguished Paper Awards from several prestigious international conferences and journals. He is a Member of the IEEE Computer Society and the ACM (SIGSOFT), a Senior Member of China Computer Federation (CCF), and serves on the Steering Committees and Program Committees of a number of high quality international conferences in software engineering community.

Zijia Jia is a PM at the ByteDance, Inc., Hangzhou, China. He was awarded M.PM from Nanjing University. He has been engaged in research work related to microservices in the industry for a long time.

Chenxing Zhong is a Ph.D. candidate in the State Key Laboratory for Novel Software Technology at the Nanjing University. Her research mainly focuses on assessing and improving Microservices Architecture.

Zhihao Shan is a PM at the Tencent Technology Co.Ltd, Shenzhen, China.

Mr. Ali Babar is a professor in the School of Computer Science at the University of Adelaide. During the more than 18 years of His research career, he have individually and as a team lead carried out World leading research on many topics that are considered vital for industrial software intensive systems and services. For the last 5 years, his team and he have started mainly focusing on developing approaches and tools for engineering Secure Software Systems and Services for/with emerging technologies such as Cloud Computing, Edge Computing, Internet of Things (IoT), and Big Data. Technologically his research falls at the intersection of Software Engineering, Artificial Intelligence (ML/NLP), and Cyber Security; methodologically he use empirical research methods, both qualitative and quantitative. His research is being carried out and evaluated in close collaboration with industry/government partners such as DST, ActewAGL, TSS, ATO, Jemena, Cisco, DST Group, Health SA and Defence SA.