On the Way to Microservices: Exploring Problems and Solutions from Online Q&A Community

Menghan Wu†, Yang Zhang†, Jiakun Liu*, Shangwen Wang†, Zhang Zhang†, Xin Xia§, Xinjun Mao†
†*National University of Defense Technology, Changsha, China*
**Central South University, Changsha, China*
§*Huawei, Hangzhou, China*
{wumengh, yangzhang15, wangshangwen13, zhangzhang14, xjmao}@nudt.edu.cn,
jiakunliu17@outlook.com, xin.xia@acm.org

***Abstract*—Microservice architecture is a dominant architectural style in SaaS industry, which helps to develop a single application as a collection of independent, well-defined, and intercommunicating services. The number of microservice-related questions in Q&A websites, such as Stack Overflow, has expanded substantially over the last years. Due to its increasing popularity, it is essential to understand the existing problems that microservice developers face in practices as well as the potential solutions to these problems. Such an investigation of problems and solutions is vital for long-term, impactful, and qualified research and practices in microservice community. Unfortunately, we currently know relatively little about such knowledge. To fill this gap, we conduct a large-scale in-depth empirical study on 17,522 Stack Overflow microservice-related posts. Our analysis leads to the first taxonomy of microservice-related topics based on the software development process. By analyzing the characteristics of the accepted answers, we find that there are fewer experts in the microservice than other domains, and such a phenomenon is most significant with respect to the microservice design phase. Furthermore, we perform manual analysis on 6,013 answers accepted by developers and distill 47 general solution strategies for different microservice-related problems, 22 of which are proposed for the first time. For instance, several problems inherent in the delivery phase can be lessened by referring to external sources like GitHub code examples. Our findings can therefore facilitate research and development on emerging microservice systems.**

***Index Terms*—Microservices, Empirical Study, Stack Overflow**

I. Introduction

Microservice architecture (MSA), as a novel architectural style, has become particularly popular in cloud-based Software-as-a-Service (SaaS) offerings, together with the spread of DevOps practices and containers technologies, e.g., Docker and Kubernetes [1]. As Lewis and Fowler's blog stated, microservices can provide greater software development agility and improve the scalability of deployed applications [2]. Due to its advantages, many successful companies have adopted this new architectural style, including Amazon, IBM, LinkedIn, and eBay [3]. It is not surprising then that the O'Reilly report [4] found that 61% of organizations had used microservices for a year or more. Moreover, the survey conducted by Bourne [5], with 200 senior IT leaders in industrial organizations, found that microservices helped the organization perform well concerning the development efficiency, the ability to use new platforms, the collaboration across teams, and sharing of services across applications.

However, the characteristics that lead to the success of microservices also introduce microservices' specific issues [6], e.g., the decentralization of microservice systems asks more effort to optimize the communication between services [7]. Investigating the challenges and issues in microservice systems and proposing potential solution strategies are significant since our observations show that microservice is becoming a hot topic on Stack Overflow (SO). Specifically, the number of microservice-related questions and users in SO has expanded substantially over the last eight years, especially after 2014[1]. Such analysis can benefit both the academic and industry communities. Researchers can perform studies towards solutions for the frequently reported problems in the software development process of microservice systems, and practitioners can be allocated to address the most frequent and challenging problems identified through the empirical study. A few studies have recently targeted this direction [9], [10], [7]. For example, Jamshidi et al. [7] discussed the benefits, evolution, and future challenges of microservices. Yarygina et al. [10] dissected the problems related to microservice security. Nevertheless, these efforts target only some specific problems of microservices and lack in-depth and comprehensive studies on microservice problems. Furthermore, to our best knowledge, few attempts so far provide solutions for problems encountered during microservice practices.

To fill this gap, in this paper, we propose a systematic and in-depth study to explore the practitioners' perspectives on microservice practices by mining posts on SO. We use SO as a data source because: (i) SO is one of the most popular question and answer (Q&A) communities and contains a lot of posts data related to software development [11]; (ii) SO has been the official discussion platform recommended by many popular microservice frameworks, e.g., Spring Cloud[2]. Our study works with a large-scale investigation of posted questions, identifying dominant topics and problems reported by practitioners and exploring preliminary solutions for these

Menghan Wu and Yang Zhang are both first authors and contributed equally to this work. Xinjun Mao is the corresponding author.

[1]The distribution of posts has been provided in the replication package [8].
[2]https://spring.io/community.

problems. In this respect, we conduct an empirical study on 17,522 questions and 22,215 answers collected from SO. We use Natural Language Processing (NLP) techniques for topic modeling, statistical analysis for topic characterization, online survey for topic comprehension, and in-depth discussion for our findings. We aim to answer the following research questions:

RQ1: (Microservice Topics) *What are the frequent topics that developers discuss in the software development process of microservice systems?* - RQ1 aims to systematically identify and taxonomically classify the problems that microservice developers face. We find that microservice-related posts consist of 10 categories and 16 topics in 4 phases of the software development process, and microservice developers encounter more problems in the governance phase.

RQ2: (Answer Characteristics) *What are the answer efficiency, answer rate, and expertise status of the identified topics?* - RQ2 aims to understand the answer status of different topics and investigate whether the microservice domain lacks active experts. We find that there is a lack of microservice experts, and the current answer status is not optimistic.

RQ3: (Solution Strategies) *What are the general solution strategies for different problem topics?* - RQ3 aims to explore the solution strategies and map them with identified topics, providing insights about microservice problem solving. At this RQ, we distill 47 general solution strategies for 14 microservice-related problems.

To summarize, our paper makes the following contributions:

- We develop the first taxonomy of microservice-related topics based on the software development process to the best of our knowledge.
- We perform a mixed-method study to shed light on characteristics of microservice-related topics and practitioners' experiences with microservices by quantitative analysis and survey.
- We distill a refined list of solution strategies for identified problems of microservices, which can be adopted to facilitate manual and automated solving of microservice-related problems.
- We provide practical implications of our findings for three kinds of potential users: researchers, developers, and service/tool builders.
- We publicly release a replication package [8] to enable researchers and practitioners to access all collected data and replicate and validate our study.

II. Preliminaries

A. Microservice Characteristics

MSA, an architectural style derived from practice, has two typical key characteristics: *Decentralization* and *Autonomy* [12]. Decentralization means that a large amount of work in MSA will no longer be managed and controlled by the center. Autonomy (also considered team autonomy) means that each development team makes its own decisions about its software. In more detail, MSA has several characteristics in practice, e.g., Componentization via Services, Products not Projects, Smart endpoints and Dumb pipes, Decentralized Governance, Decentralized Data Management, Infrastructure Automation, Design for Failure, and Evolutionary Design [2]. It is worth noting that not all of the characteristics can be implemented in microservice systems.

Componentization via Services is the most apparent characteristic of microservices. The primary way of componentizing MSA software is by breaking it down into services. These services are external components communicating with a mechanism such as a web service request or remote procedure call. When it comes to **Products not Projects**, MSA prefers that a team own a product over its entire lifetime instead of handing it over to a maintenance organization as software. Microservices introduce a new style of service integration called **Smart endpoints and Dumb pipes** instead of using an ESB (Enterprise Service Bus), with which all the business logic containing inter-service communication (smart-endpoint) and all such services are connected to a primitive messaging system (a dump pipe). It aims to achieve lightweight communication to adapt to high cohesion and low coupling. The decentralization of MSA is mainly reflected in **Decentralized Governance** and **Decentralized Data Management**. Distributed governance tends to manage its internal state separately for each service rather than through central control. For decentralized data management, microservices prefer polyglot persistence (e.g., letting each service manage its database, different instances of the same database technology stack, or entirely different database technology stacks). **Design for Failure** is the guarantee for the standard and stable operation of microservices. It is essential to be able to detect failures quickly and restore service automatically. **Infrastructure Automation** is the requirement of DevOps, which is usually applied in MSA consisting of automated testing, automated building, and automated deployment. **Evolutionary Design** considers the flexibility, replacement, and upgradeability in applying both services and technology stacks (e.g., programming languages, frameworks). For example, developers can change the technology stacks used by services with another style.

B. Research Questions

Over the last decade, leading software consultancy firms and product design companies have found that MSA is an appealing architecture that allows teams and software organizations to be more productive in general [13]. Many previous studies [9], [14], [7], [10] have concluded that there are overwhelming differences between microservice systems and traditional software systems in terms of design, implementation, test, and deployment. For example, due to the separation services and the requirement of inner-communication of services, microservice systems need more efficient communication to ensure reliability. It is reflected in the rapid increase of posts about microservices in the online Q&A community, e.g., Stack Overflow (SO). Although SO has been successfully used in studies regarding different domains, such as mobile applications [15], docker container [16], and web-based communication systems [17], we know relatively little about the existing problems that microservice developers discussed in the SO community. The only exception is performed by Bandeira et al. [18], who presented a preliminary analysis of microservice-related posts on SO and found 18 and 15 subjects for technical and conceptual discussions, respectively. However, due to the limited sample size and preliminary research goals, their works have not given in-depth and comprehensive insights about the nature of microservice developers' problems and their potential solutions. Therefore, we are motivated to conduct a systematic and comprehensive study to distill the practitioners' problems and solutions on microservice practices.

First, we seek to replicate and improve Bandeira et al.'s work [18] by investigating more SO posts samples. Instead of dividing the posts into conceptual and technical types, we want to systematically identify and get the fine-grained taxonomy of microservice-related topics based on the software development process. Hence, we devise the first research question to know more about microservice-related topics:

RQ1: *What are the frequent topics that developers discuss in the software development process of microservice systems?*

After identifying the taxonomy of microservice-related topics, we want to investigate the differences of characteristics towards each topic, e.g., answer efficiency, answer rate, and expert status. Those characteristics allow us to draw more profound insights into the challenges of microservice-related topics. Also, this analysis may help us understand the solving status and potential reasons for microservice problems. Therefore, we ask:

RQ2: *What are the answer efficiency, answer rate, and expertise status of the identified topics?*

Even though microservices have attracted much attention from the academic and industry communities, there is a lack of investigation regarding the available solution strategies for microservice-related problems on SO. We may extract and gain some general solution strategies by analyzing the SO posts, mainly those accepted answers. It can help developers solve problems effectively and further facilitate the realization of automatic tools for better microservice systems development. Thus we ask:

RQ3: *What are the general solution strategies for different problem topics?*

III. METHODOLOGY

A. Data Collection

First, we download all publicly accessed data in SOTorrent [19] (as of February 2021). To identify all the microservice-related topics that developers discussed on SO, we select all SO questions and their answers.

Tag-based filtering. Similar to approaches in [16], [20], we develop a set of microservice-related tags T to determine and extract posts from SO. In this work, T contains tags '*micro-service*', '*microservice*', '*micro-services*', and '*microservices*'[3]. We extract all posts with at least one of such tags to develop the initial dataset.

Content-based filtering. We find that some microservice-related posts may not have tags in T, e.g., this post[4] describes a question about the deployment of microservices on Amazon Web Services (AWS), but only with tags as '*amazon-web-services*', '*kubernetes*', '*amazon-eks*', and '*nginx-ingress*'. Besides, developers may assign questions with improper tags [11]. Therefore, we use keywords (i.e., tags from T) and perform string matching on each post title and body. Finally, we collect 17,522 questions with 22,215 answers by combining the result datasets generated from tag-based filtering and content-based filtering.

B. Problem Classification

We perform data pre-processing on the collected dataset above and apply the topic modeling technique to get the fine-grained topic taxonomy of microservice-related problems. Each topic in the fine-grained taxonomy represents the problem in microservice practices. Note that we only consider the title, body, and answers of each question for topic modeling.

Data pre-processing. First, we remove code snippets, HTML tags, URLs, and stopwords [16]. Second, to minimize noise, we use the initial set of *NLTK* stopwords to build a list of stopwords specifically for this study by adding generic, non-microservice-specific, but high-frequency words to the initial set, e.g., *'jar'* and *'bean'* are common words in Java while they do not give us a clear indication of the current topics in microservices [16]. Next, we adopt the *bigram* and *trigram* models using *Gensim* [21] to improve the quality of text processing [22]. Additionally, we also perform the lemmatization on the collection of documents by *SpaCy* [23]. Finally, we remove the words that appear in more than 80% and less than 2% of the documents to get the corpus [24].

Topic modeling. Following previous studies [16], [25], we use the Latent Dirichlet Allocation (LDA), which is the powerful topic modeling technique with the implementation of LDA provided by *Gensim*'s `LdaMalletModel`. LDA finds topics based on the co-occurrence frequency of the word sets in the corpus of documents [26]. To choose the best number of topics (denoted K, which influences the performance and takes control of the topic granularity) and iteration value (denoted Ir, which affects classification accuracy) in the LDA model, we perform a broad ranger of experiments by varying K from 2 to 50 in a step of 1 and Ir with 100, 500, 1,000, and 2,000 respectively. Next, the first two authors test a randomly selected sample of 50 posts from each topic for different K values to ensure we choose the best K value. This experiment finds that K=16 with Ir=2000 provides a sufficiently granular set of topics for our dataset. Moreover, we set the hyper-parameter of LDA α=50/K and β=0.01, consistent with the values used by previous works [27], [16]. In general, this model only generates a list of 10 phrases as a topic, but it does

[3]The complete list of our candidate tags can be found in our reproduction package at [8].

[4]https://stackoverflow.com/questions/53227358.

TABLE I: Summary of findings and implications.

Findings of microservice topics	**Implications**
F.1 We derive a taxonomy of microservice-related discussion consisting of 4 phases, 10 categories, and 16 topics. Interestingly, microservice developers encounter more problems in the governance phase rather than the construction phase.	**I.1** Practitioners face diverse challenges when developing microservice systems. Researchers should conduct more studies to shed light on the potential reasons. Also, practitioners should seriously pay attention to the governance phase which is a challenge for developers.
F.2 In the design phase of microservice systems, developers look forward to obtaining the guidance of best practices or examples for their microservice design requirements. In particular, the most problems that developers encounter are related to microservice boundaries (28.71%).	**I.2** Both researchers and service providers should summarize more practical design specifications and detailed instructions to help developers construct their microservices more easily and quickly. Moreover, researchers should focus on defining appropriate decomposition boundaries and conditions, and practitioners should weaken the dependencies between services to help microservice systems maintain the low coupling characteristic.
F.3 The communication, failure tolerance, and data management cover the challenges appearing in the construction phase. The majority (11.71%) of these challenges are thrown with *Communication*. Although microservices encourage technology diversity, developers do not have a definite standard for the current selection of these technologies.	**I.3** Researchers and service providers should deliver more supported techniques for topics (i.e., web interaction, inner-communication, exception handling, and data management). Developers should use more sophisticated tools to manage and analyze logs to recover from failures, e.g., the data formatting error.
F.4 The delivery of microservice (e.g., testing, building, deployment) is a challenge for developers, especially in the management and configuration of diversified CI/CD tools or services. During the testing, the most frequent problems that developers face are related to project environment management, versioning control, mocking, and CI/CD.	**I.4** It is still challenging for developers to adopt DevOps effectively in microservice practices. Therefore, how to manage and help developers configure the best delivery pipeline needs to be addressed. Also, service / tool providers should simplify the configuration complexity to lower the initiation obstacles for more inexperienced developers.
F.5 *Microservice Governance* has emerged as the most problematic and essential phase. Managing and monitoring common resources shared by microservices (e.g., API-gateway, component application, authorization and authentication, and logging) has been a critical problem that developers seek to solve.	**I.5** These results confirm that microservice systems have many management problems and gain a large attack surface. Researchers and practitioners should propose design techniques focusing on highly resilient and low coupled microservice systems, and develop effective solutions to trace and monitor vulnerabilities and related risks. For instance, researchers should investigate the historical logs and exceptions to distill the factors that affect microservices' reliability and security.
Findings of answer characteristics	**Implications**
F.6 Compared to other *SE discussion domains*, there is a lack of experts in *Microservices*, and only a small percentage of the problems have been resolved (less than 25%).	**I.6** Microservices are hard for newcomers. The SO community should propose incentives to encourage microservice experts to contribute. Besides, microservices have accumulated a lot of unresolved problems. Our finding motivates more researchers and experienced developers to explore the microservice-related problems discussed by practitioners and solutions for these problems in microservices.
F.7 There are fewer answers and experts in the design phase than in other phases. In those microservice-related topics, questions about *Containers* topic have the highest expertise value and faster response time, while questions about *Spring Cloud Components* topic need more experts to participate in discussions.	**I.7** Researchers should further evaluate the characteristics differences of microservice-related topics in other dimensions (e.g., trends and participants) and explore their potential reasons between these complex topics. Also, developers and service/tool providers should pay more attention to those phases and topics with longer response times, fewer answers, or fewer experts.
Findings of solution strategies	**Implications**
F.8 We distill 47 general solution strategies for 14 problems in the microservice-related topic taxonomy. However, there is no silver bullet for fixing arbitrary phases of microservice-related problems.	**I.8** Researchers may facilitate automated resolution via embedding more prior-experience rules (including the strategies in our study) or mining more cases from big data to expand solution strategies.

not output actual meaningful and representative topic names. To label topics, following previous work [16], we inspect the top 10 posts and read through the most relevant 15 posts for each topic.

Manual classifying. First two authors analyze and conclude the topics from the topic modeling results to determine the taxonomy of microservice topics. First, each topic is classified into categories and phases by authors' experience and referring literature [3], [28], [29]. Next, we use Cohen's Kappa (κ) [30] to calculate the inter-rater agreement in manual classification. The value of the inter-rater agreement is more than 97% by iterative discussions, promoting the perfect agreement and rationality of classification. Finally, we get the fine-grained taxonomy of microservice-related topics.

C. Quantitative Analysis

To have a deeper understanding of the answer characteristics of the topics, we use *resp time* (the duration from when the question is raised to when the answer is proposed, in minutes) to measure the answer efficiency following [20], we use *% acc.answer* (ratio of posts having accepted answers per topic) and *% answer* (ratio of posts having any answer per topic) to measure the answer rate following [16]. Besides, we also provide *% avg.postAge* (days of the post's age) in order to mitigate its impact on the *% acc.answer* and *% answer*. Following Tian et al.'s work [31], we use the metric *ExpertiseValue* to measure the expert status, which considers user attributes (e.g., user reputation), quality of the answered questions, profile view count, tags, and post characteristics.

In addition, we compare the microservices with the other three baseline samples (i.e., Docker, Web, and SO community) to gain a clear profile of microservices on SO. Docker and Web are two popular discussion domains of SO [11], [16]. Docker plays an essential role in microservice practices, and Web is the main application scenario of microservices. By comparing with those baselines, we can gain insights about how microservice practices differ from other domains within the SO community. The samples of these baselines have been generated by tag-based selecting for Docker and Web posts. We also select the overall SO dataset for the samples of SO community. This processing allows us to obtain posts that are strongly correlated with the baselines.

D. Solution Strategies Extraction

We extract the general microservice solution strategies by manual labeling. Each post in the dataset may cover some information describing (i) the problem in microservice practices, (ii) the potential reason for this problem, and (iii) the solution provided by the SO community. In this paper, we extract the most general solutions from 6,013 accepted answers in our dataset as the accepted answers are verified solutions in the questioners' problems. To facilitate the manual labeling process, we apply LDA again for all accepted answers, as that the automated method such as LDA could extract the keywords and obtain solution groups instead of random and accidental manual sampling. Next, first two authors read all the answer topics (solution groups) and assign each topic with a short but descriptive phrase as initial codes. Then, they group similar codes into categories and phases. The grouping process is iterative, in which they continuously go back and forth between the results and posts to refine the answer taxonomy. When there is no agreement between the two authors, another annotator (the third author) is introduced to discuss and resolve the conflicts. They follow this procedure until they reach an agreement on all answer topics. Finally, we get a refined list of solution strategies for each problem in answer taxonomy by manual summarizing similar solutions in answer topics.

E. Survey

To obtain additional insights into the microservice practices, we conduct a survey with a sample of industry developers. The survey[5] is organized into three parts: (i) basic information about respondents, (ii) taxonomy details of the microservice topics, and (iii) key observations' overview in this paper. Due to the developers are overlapped in GitHub and SO [32], we identified those developers who also use SO, sampled 200 of them, and sent them email invitation with a link to the online form. Within 10 days, we receive 26 responses, for a response rate of 13% which is consistent with other software engineering online surveys [33], [34]. Respondents indicate that their experience in the industry was 7.10 years on average (median: 8; range 1—13), while their microservice experience was 3.07 years on average (median: 3; range 1—9).

IV. RESULTS

A. RQ1: Taxonomy of Microservice-related Topics

Fig. 1 provides a fine-grained taxonomy of microservice-related topics. This taxonomy defines the topics as the results from the LDA model, the categories and phases as high-level topics' recapitulation. We present 16 topics of 10 categories in 4 phases and 53 frequent examples in the corresponding topics. Our survey result shows that more than 92% of respondents are satisfied with our phase division (i.e., *Microservice Design*, *Microservice Construction*, *Microservice Delivery*, *Microservice Governance*), and nearly 85% of respondents are satisfied with detailed topic classification. We add up the percentages of topics in each phase to explore the numerical characteristics of different phases.

Moreover, we find that developers ask the most questions in the *Microservice Governance* phase (32.96%), followed by *Microservice Construction* (29.25%), *Microservice Delivery* (28.52%), and *Microservice Design* (12.24%). The topic of the most interest to developers is *Service Management* topic, which plays a vital role in MSA. These indicate that microservice developers concern more about *Microservice Governance* instead of *Microservice Construction*. The broad taxonomy overview is summarized by **Finding F.1 and Implication I.1 in Table I**.

In the following, we explain problems represented by categories along with their constituent topics and examples. Note that due to the limit of pages, we only focus on those topics that characterize microservices and will not introduce the topics that have emerged in other domains.

1) Microservice Design (12.24%): Developers usually discuss design strategies of microservices at the beginning of development [35]. The design process is critical to realizing the microservice systems [36]. We find that the most questions asked in this phase are about microservice boundaries (nearly 28.71%). A bounded context defines the boundary and granularity of the services, which is a crucial requirement of migrating towards microservice architecture [37], [38]. For instance, one developer asked the question[6]: when *"defining Microservice boundaries"*, and "*...how would you design your micro-service if you need aggregated outputs?*" was the specific requirement with which he/she was confused.

Most of our survey respondents (90%) stated that they sought to guide best practices or examples when designing microservice systems. One respondent reminds us that microservices lose their original purpose of low aggregation due to the complex invocation relationships between services. Therefore, to maintain the low coupling characteristic of microservice systems, defining appropriate decomposition boundaries and conditions for microservices and weakening the dependencies between modules are issues that academia and industry should focus on.

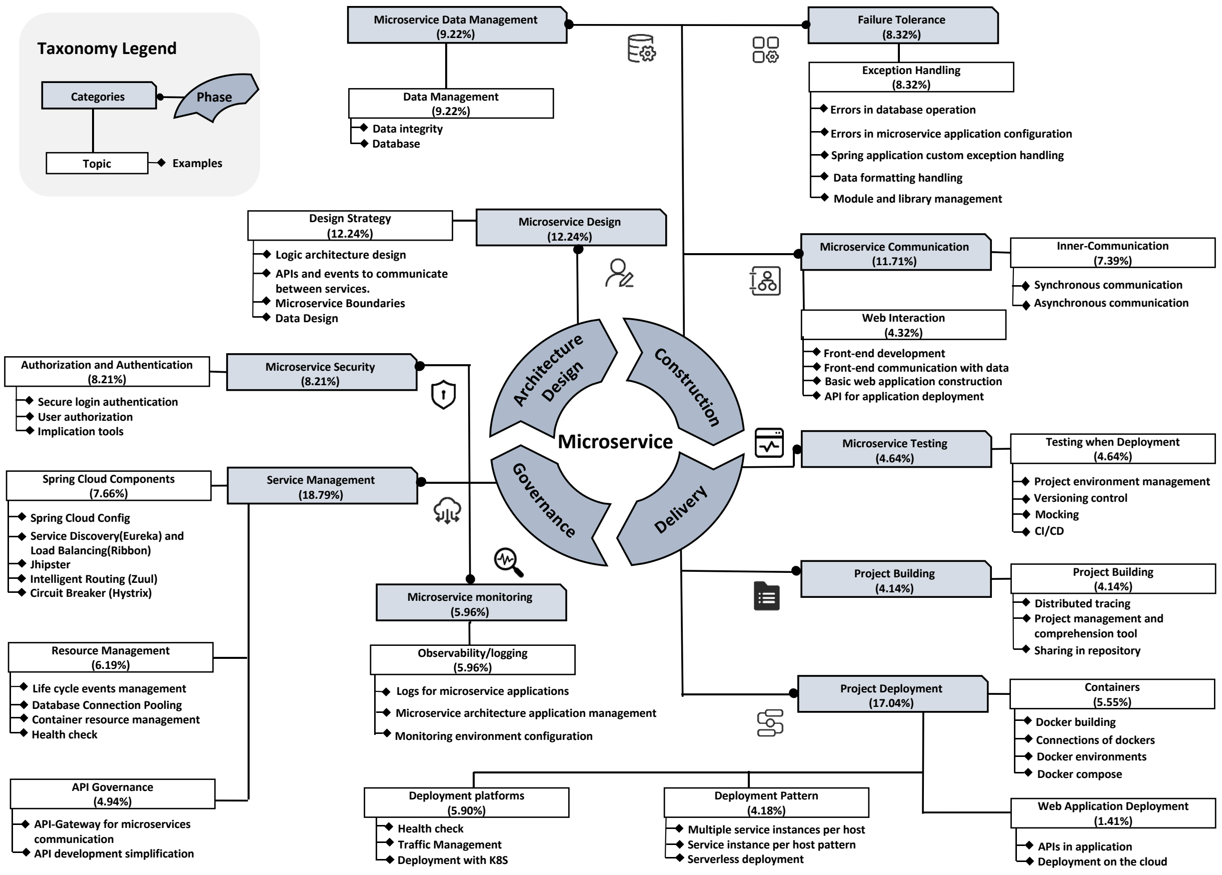
The analysis of topics in the design phase is summarized by **Finding F.2 and Implication I.2 in Table I**.

2) Microservice Construction (29.25%): We observe three categories, i.e., *Microservice Communication*, *Failure Tolerance*, and *Microservice Data Management*.

Microservice Communication (11.71%): Due to microservice decentralization, microservice communication is quietly important for componentization via services and smart endpoints and dumb pipes. Problems with communication are difficult for microservice developers [6]. Under our observation,

[5]Our survey questions can be found in our reproduction package at [8].

[6]https://stackoverflow.com/questions/62001909.



Taxonomy Legend
Categories
Phase
Topic
Examples
Microservice Data Management (9.22%)
Data Management (9.22%)
Data integrity
Database
Failure Tolerance (8.32%)
Exception Handling (8.32%)
Errors in database operation
Errors in microservice application configuration
Spring application custom exception handling
Data formatting handling
Module and library management
Design Strategy (12.24%)
Logic architecture design
APIs and events to communicate between services.
Microservice Boundaries
Data Design
Microservice Design (12.24%)
Microservice Communication (11.71%)
Inner-Communication (7.39%)
Synchronous communication
Asynchronous communication
Web Interaction (4.32%)
Front-end development
Front-end communication with data
Basic web application construction
API for application deployment
Authorization and Authentication (8.21%)
Secure login authentication
User authorization
Implication tools
Microservice Security (8.21%)
Architecture Design
Construction
Microservice
Governance
Delivery
Microservice Testing (4.64%)
Testing when Deployment (4.64%)
Project environment management
Versioning control
Mocking
CI/CD
Spring Cloud Components (7.66%)
Spring Cloud Config
Service Discovery(Eureka) and Load Balancing(Ribbon)
Jhipster
Intelligent Routing (Zuul)
Circuit Breaker (Hystrix)
Service Management (18.79%)
Project Building (4.14%)
Project Building (4.14%)
Distributed tracing
Project management and comprehension tool
Sharing in repository
Microservice monitoring (5.96%)
Observability/logging (5.96%)
Logs for microservice applications
Microservice architecture application management
Monitoring environment configuration
Resource Management (6.19%)
Life cycle events management
Database Connection Pooling
Container resource management
Health check
Project Deployment (17.04%)
Containers (5.55%)
Docker building
Connections of dockers
Docker environments
Docker compose
API Governance (4.94%)
API-Gateway for microservices communication
API development simplification
Deployment platforms (5.90%)
Health check
Traffic Management
Deployment with K8S
Deployment Pattern (4.18%)
Multiple service instances per host
Service instance per host pattern
Serverless deployment
Web Application Deployment (1.41%)
APIs in application
Deployment on the cloud

Fig. 1: The taxonomy of microservice-related topics.

there are two topics (i.e., *Inner-communication* and *Web Interaction*) in the *Microservice Communication* category. Particularly, inner-communication has synchronous and asynchronous approaches available. In asynchronous communication, developers are more concerned about stream processing technologies and application integration approaches (e.g., message-driven approach and event-driven approach). For instance, this question[7] asked *"Getting response for user-initiated action from microservices with asynchronous communication."* to get an actual implementation of asynchronous communication. In our survey, 73.1% of respondents confirmed that communication had been the major challenge in the construction phase.

Failure Tolerance (8.32%): Failure tolerance is necessary for the decentralization and autonomy of microservices. Systems may fail at any time, which makes detecting the failures quickly and restoring service automatically a challenge for microservices [39]. That would put a lot of emphasis on real-time monitoring of the application to achieve observability and resilience of microservice systems. We find that questions on this topic include several types of error. Among these errors, data formatting error is the most (38.09%). This error often shows its symptom via logs. For example, this question[8] *"Spring RestTemplate sends empty string as null?"* described a data formatting error reflected in logs.

Data Management (9.22%): Microservices favor decentralization in all aspects, including how data is persisted. Maintaining data consistency across microservices is one of the challenges under this background. Moreover, hiding data implementation details may be a criterion of data management, aiming to interact with other architecture components. For instance, this question[9] about *"Event Sourcing in occasionally connected systems: what if there are two servers?"* described event sourcing as an approach for data integration.

Based on our observations and summaries of construction questions, we find that many microservice construction techniques and approaches have emerged (Microservices encourage technology diversity [40]). However, developers do not have a definite standard for the current selection of these approaches. Moreover, our survey confirms that the communication is the most critical concern for current developers.

The analysis of topics in the construction phase is summarized by **Finding F.3 and Implication I.3 in Table I**.

[7]https://stackoverflow.com/questions/50070895.

[8]https://stackoverflow.com/questions/63338042.

[9]https://stackoverflow.com/questions/41297568.

3) Microservice Delivery (25.82%): We observe three categories, i.e., *Microservice Testing* (4.64%), *Project Building* (4.14%), and *Project Deployment* (17.40%).

We find *Project Deployment* category has more problems than other categories. Without an excellent continuous integration and continuous delivery (CI/CD) process, developers cannot achieve the agility that microservices promise. Mendonça et al. [36] showed that CD usage was challenging in developing self-adaptive microservice systems. In other words, developers concern about the practice of the best delivery pipeline. Primarily, we observe that developers ask questions about three deployment patterns: multiple services per host, service instance per host, or serverless deployment. For instance, serverless deployment uses a deployment infrastructure that hides any concept of servers. This question[10] asked about "*Boilerplate for independently deployable microservices based on Serverless framework*" for serverless deployment implementation.

In our survey, 69.23% of respondents agreed that the management and configuration of diversified CI/CD tools or services would produce more problems. In particular, one respondent told us, "*...in order to deploy, we need to know a lot about the new technologies such us Docker*", but "*how we could have everything with DevOps like as automation of the deploy and the tests*" is currently the main pain point. Our survey shows that the *Testing* topic has been a primary concern of developers, followed by *Containers*, *Deployment patterns*, and *Web application deployment*. Researchers and tool supporters should mainly focus on the complexity and flexibility of microservice delivery.

The analysis of topics in the delivery phase is summarized by **Finding F.4 and Implication I.4 in Table I**.

4) Microservice Governance (32.96%): In our taxonomy, this phase is the most prominent phase, including the categories (i.e., *Service Management*, *Microservice Security*, and *Microservice Monitoring*).

Service Management (18.79%): The service management techniques are centrally applied to microservice systems. These techniques include service registry and discovery, how to control the services resource, etc. We observe that most of these capabilities are implemented as part of API management, resource management, or applying with spring cloud components.

Specifically, the first topic *API Governance* (4.94%) is applying common rules relating to API standards and security policies for APIs. Managing service APIs and API ecosystems has been an emerging challenge and has drawn attention from developers [41]. For example, the API management layer or API gateway is used to expose microservices to the consumers. This question[11] about *"Is aws lambda can expose only one spring boot api"* described the developers' problem when using the API-Gateway.

[10]https://stackoverflow.com/questions/65119342.

[11]https://stackoverflow.com/questions/52208226.

The second topic is related to *Resource Management* (6.19%). Resource management can help systems reach reusability and improve systems' overall efficiency [42]. As the following example[12] *"Kubernetes pods restart issue anomaly"* showed, developers were confused about getting resources like CPU and Memory under control.

The last topic is *Spring Cloud Components* (7.66%). They are tools that allow developers to build common usage patterns for a distributed system. Among these, Spring Cloud Netflix (a mature solution for Spring Cloud Components) provides Netflix OSS integrations for spring boot applications, aiming at solving the distributed-system problems at a scalable level for microservice governance. For example, this developer asked the question[13] *"Have Zuul edge service automatically use microservice's basic auth credentials"* for determining the actual function of each component.

Microservice Security (8.21%): Microservice security is often considered to promote the reliability and stability of microservice systems. It is a multi-faceted problem that requires a layered security solution that is not available out of the box at the moment [10]. Under our observation, developers pay attention to problems about securing service-to-service communication such as authorization and authentication. For example, the question[14] about *"How to implement OAuth2 'Token Exchange' with Spring Cloud Security"* presented the difficulty of OAuth2 protocol usages about securing service-to-service communication.

Microservice Monitoring (5.96%): *Observability* in the governance phase is reflected in microservice monitoring. It aims to provide data on the behavior of systems. The robustness of the logging and monitoring framework is one of the main criteria for mature microservice practices [42]. For example, logging can be a mechanism to retrospect the failed projects and learn from system mistakes for microservice teams. This question[15] about *"Best practices for LOGGING in microservice architecture"* described the logging approach for microservice monitoring. One of our respondents pointed that "*logging is important in terms of something bad happens ... it helps with parameters to replicate the issue in local and debug.*"

The analysis of topics in the governance phase is summarized by **Finding F.5 and Implication I.5 in Table I**.

B. RQ2: Answer Characteristics Comparison.

Table II summarizes characteristics metrics on *Microservices* and other *SE domains*. It also presents the same metrics in different phases and topics corresponding to the taxonomy in section IV-A.

1) SE Discussion Domain Comparison: Based on Table II, it is more challenging for developers to solve problems related to *Microservices* than other SE discussion domains. The value of *% acc.answer* is less than 30%. While in SO, it is over 50%.

[12]https://stackoverflow.com/questions/51474223.

[13]https://stackoverflow.com/questions/48215910.

[14]https://stackoverflow.com/questions/34905628.

[15]https://stackoverflow.com/questions/56533725.

TABLE II: The metrics of different topics.

Name	*resp time*	*%acc.answer*	*%answer*	*%avg.postAge*	*ExpertiseValue*
Phase: Microservice Design	272	16.21	34.74	1,384.57	3,572.69
Design Strategy	272	**16.21**	**34.74**	**1,384.57**	3,572.69
Phase: Microservice Construction	336	25.52	53.97	1,102.18	5,264.47
Data Management	418	22.87	51.84	1,121.34	4,523.20
Exception handling	191	**32.76**	**62.30**	1,070.49	6,659.58
Communication	452	26.85	55.86	1,066.09	6,747.65
Web Interaction	285	19.58	45.89	1,184.51	3,127.48
Phase: Microservice Delivery	400	25.72	54.77	1,131.20	4,903.32
Testing when Deployment	324	23.92	52.58	1,116.47	6,448.34
Project Building	508	25.43	56.45	1,145.27	3,212.14
Deployment Platforms	564	28.82	59.00	**1,015.63**	3,473.52
Containers	**160**	27.55	57.91	1,170.82	**7,815.64**
Deployment Pattern	443	23.95	49.14	1,236.59	5,502.62
Web Application Deployment	409	24.65	53.54	1,143.47	2,967.68
Phase: Microservice Governance	549	23.81	52.78	1,125.44	3,943.40
Authorization and Authentication	705	22.12	52.46	1,094.82	3,268.31
Resource Management	340	22.89	51.97	1,137.58	6,513.34
API Governance	580	19.24	45.11	1,064.47	3,583.87
Spring Cloud Components	**773**	25.34	56.10	1,201.24	**1,970.04**
Observability/logging	351	29.45	58.27	1,101.12	4,381.46
Microservice	442	24.48	52.70	1,144.43	4,606.81
Docker	350	40.36	77.64	-	11,098.79
Web [43], [16]	19	52.00	80.51	-	13,864.53
SO Community [44], [16]	35	51.97	60.89	-	6,129.21

Compared with others, *Microservices* has a smaller *Expertise-Value*. Therefore, it is evident that *Microservices* has a much lower number of experts than other SE discussion domains, which is also verified in our survey, i.e., 65.4% of repondents agreed that the current microservice community lacks experts. There have been many successful microservice practitioners in current practices, but general guidance and practical solution strategies are lacking. Researchers should use the successful practices as prior knowledge, summarize solution strategies and develop tools to promote the development of microservice systems.

2) Phase Comparison: We find that the *Microservice Governance* phase has a long response time in terms of accepted answers (549 minutes). The *Microservice design* phase has a lower percentage of answers (accepted answers) that they receive than other phases'. They also have the lowest expertise value among all phases, which can be interpreted as a difficulty for developers. In particular, the *Spring Cloud Components* topic has the longest time (773 minutes) to receive an accepted answer. In addition, it has the lowest expertise value, which can be inferred as a challenging topic. Our survey respondents (73.1%) also reported that this topic needs more experts to participate in the discussion. On the other hand, the *Containers* topic has the shortest response time (160 minutes) and the highest expertise value. Although the *Exception Handling* topic has the highest number of answers, its accepted answers are still less than 33%. This evidence indicates that microservice-related topics of all phases are complex.

For RQ2, the analysis is summarized by **Finding F.6 F.7 and Implication I.6 I.7 in Table I**.

C. RQ3: Solution Strategies Exploration.

Table III illustrates the general solution strategies for different problems. The columns "Phase" and "Category" (part of) are consistent with our taxonomy in Fig. 1, also providing the percentages of solution categories in parentheses; the column "Problem" corresponds to the problems (examples) of topics found in RQ1; and the last column "Solution Strategy" briefly describes the general solution strategies. In our study, we observe two kinds of solution strategies: **high-level instructions** (e.g., asynchronous communicating by message-based mechanisms), most of which are also mentioned in other literature, it focuses on answering the overall challenges of microservice practices (e.g., technologies and ideas); and **general solution patterns** (e.g., changing port path), it focuses on answering specific questions in different phases.

Overall, we distill 47 general solution strategies, including 25 high-level instructions and 22 general solution patterns for 14 problems, accounting for 93.75% of topics in the microservice-related answer taxonomy. We observe the similarity of solution strategies between different problems, confirming that solving microservice-related problems is a challenge. Also, we find that most categories do not have a general solution pattern. One reason is that developers can only answer parts of the questions, most of which are high-level instructions and applied in other SE fields. Moreover, high-level instructions need more up-to-date knowledge on third-party resources and should be considered case by case.

The construction phase has the most solutions (i.e., 34.06%, not including *Error Handling*) while the design phase has the least solutions (i.e., 11.25%). It is consistent with the notion that developers spend more time on the construction phase [53]. Note that, due to the space limitations, we only explain general solution patterns in detail[16].

1) Architecture Design: We identify 9 frequent solution strategies (all of them are high-level instructions) for *Architecture Design* problems. We find that the requirements of MSA vary in different scenes, and most questions need to be solved case by case. Also, those solution strategies have been proposed in the reference papers in Table III.

2) Microservice Construction: We identify 13 frequent solution strategies (including 7 high-level instructions and 6 general solution patterns) for *Microservice Construction* problems. According to our statistics, 25.64% of solution strategies are related to *Inner-service Communication*. For example, we observe that some questions about asynchronous request-response are solved by adding extra knowledge, such as logs in *Data Formatting*. Developers on SO often paste logs for additional data to help the askers avoid keeping a local state during the communication. This answer[17] presented a solution that uses logs as extra response information to avoid infinite loops when communicating.

3) Microservice Deployment: We identify 17 frequent solution strategies (including 9 high-level instructions and 8 general solution patterns) for *Microservice Deployment* problems.

As the observation in section IV-A, Docker is widely used in microservice systems, and the problems about this topic are heatedly discussed. After the authors check and conclude, there are several solutions (i.e., updating the version of corresponding components, changing the port path, configuring docker composing, and changing the details of

[16]The answer examples and detailed discussion of each solution strategy can be accessed in our replication package [8].

[17]https://stackoverflow.com/questions/50625038/console-print-statement-not-working-in-junit-test-in-spring-boot-microservice/50625222#50625222.

TABLE III: Solution strategies for microservices-related problems.

Phase	Category	Problem	Solution Strategy
Architecture Design (11.25%)	Microservice Design (11.25%)	Design Strategy	Decentralizing application: ① by scale [45], ② by domain [29]
			Designing database architecture by [46]: ① Approach with a shared database, ② Approach with a domain-based database
			Organizing microservice teams [47]: ① CI/CD ② Automated Testing
			Communicating between internal services [3]: ① Synchronous message-passing pattern, ② Asynchronous Message-passing pattern, ③ Asynchronous Event-driven pattern
Microservice Construction (40.23%)	Microservice Data Management (8.42%)	Distributed Data Management	Managing data with the combination of event sourcing and CQRS [28]
			Managing data with a single database per microservice (not per instance) [47] [45]
			Storing data by transaction reliability pattern and processing [28]
	Inner-service Communication (25.64%)	Asynchronous Communication	Asynchronous communicating by message-based mechanisms [48]
			Applying message brokers: Kafka, RabbitMQ, and Redis. [49]
			Communicating with forms [50]: ① Publish and subscribe (Topics) approach ② Point-to-point (Queues) approach
		Data Formatting	Defining decoder function
			Defining response, request, or header
			Using Extra knowledge such as logs
		Synchronous Communication	Rewriting call method
			Changing parameters
			Applying healthy check
	Failure Tolerance (6.17%)	Error Handling	Resource Management
			CS Connection
			API Governance
Microservice Deployment (28.89%)	Building (7.67%)	Microservice Project Building [13]	Building with Maven
			Building with Gradle
			Adopting popular application framework: Spring framework
	Deploying (21.22%)	Deployment to Kubernet	Exposing services with Istio [51]
			Orchestrating services by [47]: ① Kubernetes, ② Docker swarm
		Docker	Updating the version of corresponding components
			Changing the port path
			Configuring docker composing
			Changing the details of configuration files
		Spring Application	Referring to the spring official documentation and GitHub issues
			Referring to GitHub examples
		Application Hosting	Referring to the official documentation
			Referring to GitHub examples
			Hosting application by [52]: ① Amazon Web Services (AWS), ② Google Cloud,
Microservice Governance (19.63%)	Service Management (12.52%)	Spring Cloud Netflix	Referring to Spring official documentation and GitHub issues
			Changing configuration files
			Referring to GitHub examples
		API-Governance	Referring to GitHub examples
			Formatting data style
	Microservice Security (7.11%)	Authentication and Authorization	Adopting OAuth and JWT (JSON Web Token) [28]
			Authenticating with password-based approach [28]
			Performing security with a popular framework [3]: ① Spring cloud gateway, ② Spring security

configuration files) are available to address those problems, especially for docker configuration. For example, this answer[18] suggested that developers change configuration files to open up the connection between dockers. Furthermore, some solution strategies are non-intuitive, and developers are confused by specific applications in microservice deployment. Particularly, as Table III shows, we find that some solution patterns about problems in this phase are mainly about referring to GitHub examples and official documentation. For example, this answer[19] advised developers to refer to GitHub examples.

4) Microservice Governance: We identify 9 frequent solution strategies (including 4 high-level instructions and 5 general solution patterns) for *Microservice Governance* problems.

Microservice governance runs through the critical phases in the software development process of microservice systems. It is an essential guarantee for the correct and stable operation of microservice systems. However, according to our statistics, current solutions about the governance phase are not accurate

[18]https://stackoverflow.com/questions/56501428/connect-to-a-service-on-a-different-stack-from-docker-compose/56523538#56523538.

[19]https://stackoverflow.com/questions/66974806/github-actions-gke-workflow-deployment-clarification/67023631#67023631.

and detailed. Answers from SO show that spring cloud netflix is the most popular framework used for microservice governance. It has many accepted answers about service register and discovery, which indicates that the starting point (service register and discovery) is crucial for successfully building of microservice applications.

Additionally, we observe some questions about *Spring Cloud Components* and *Spring Applications* are solved by referring to GitHub examples or official documentation. In addition, correcting configuration files is another choice for these problems. As Table III shows, we find that the current solutions are primarily about referring to tutorials. It indicates that learning costs are high in microservices. At the early stage of microservices, developers should learn the concepts and corresponding tools to prevent misuse [54].

For RQ3, the analysis is summarized by **Finding F.8 and Implication I.8 in Table I**.

V. Threats to Validity

Internal validity: Threats to internal validity are related to experimenter bias and errors. Our study applies qualitative analysis in RQ1 and RQ3. To reduce the bias and errors, we make every data item that needs to be marked by at least two authors working together with a third annotator resolving the conflicts and inspecting all results. We use Cohen's kappa as an indicator of author agreement of more than 97%, to reduce the possible subjectiveness. During the experiment, we observe that the LDA model has randomness. To solve this problem, we select the best random number seed when tuning the parameters (e.g., iterations, best topic number), with which we can train our model on a fixed path. This method can reduce the impact of randomness brought by the LDA model and promote the best results.

External validity: Threats to external validity are related to the generalization of our study. In this study, we concentrate on microservice-related posts in SO as the data source. Data bias caused by a single data source cannot guarantee the generalizability of our observations. In the future, we plan to extend our study in more data sources (e.g., GitHub). In terms of qualitative analysis, we rely heavily on manual analyzing. Due to the sheer volume of our data, we selected 1,000 posts and 750 accepted answers as representative data for manual analysis after LDA, giving us a confidence level of 95% and a confidence interval of 5%.

VI. Related Work

Recently, there have been various investigations on microservice practices. Indrasiri et al. [28] provided a comprehensive understanding of MSA principles and usage in practices. Yarygina et al. [10] studied microservices security's taxonomy and described the design and implementation of a simple security framework for microservices. These studies proposed some MSA based on specific application scenarios. Considering the increasing popularity of microservices, our work seeks to identify the current challenges and makes the first step to investigate their solutions from the online Q&A community with a large scale of development discussions.

Several researchers have studied the history, benefits, and future challenges of microservices by presenting systematic mapping studies from the literature [55]. Soldani et al. [56] systematically analyze the industrial grey literature on microservices to identify benefits and issues related to microservice-based architectures. For example, Francesco et al. [57] contributed with a classification framework for research studies on architecting with microservices and discussed emerging findings and implications for future research. Hassan et al. [58] better understood the transition to microservices, reviewed modeling approaches, and processed used to reason about microservice granularity. They analyzed the current microservice state and provided the classification of microservices to illustrate the existed challenges. Unlike previous work, our study focuses on the current practices and challenges of the microservice practitioners' experience as long as current solutions summarized by authors.

Some other researchers also investigated the microservice systems in practice by using questionnaires or interviews. Xiang et al. [39] identified 11 practical issues that constrained the microservice capabilities of organizations and summarized the practices that had been explored and adopted by the industry, along with the remaining challenges. Wang et al. [42] collected and categorized best practices, challenges, and some existing solutions by interviews with practitioners that have successfully developed microservice-based applications for commercial use. Our work can be considered a complement, i.e., we explore the microservice-related problems and potential solutions by mining the questions posted on SO.

VII. Conclusion

In this work, we perform an empirical study of 17,522 microservice-related posts on SO to explore the practitioners' perspectives on microservice practices. Using LDA modeling and manual analysis, we develop the first taxonomy of microservice-related topics based on the software development process, including 4 phases, 10 categories, and 16 topics. Next, we investigate the characteristics of answers (i.e., answer efficiency, answer rate, and expertise status) and find that there is a lack of microservice experts, and the current answer status is not optimistic (less than 25%). Furthermore, we perform manual analysis on 6,013 answers accepted by developers and distill 47 general solution strategies for 14 microservice-related problems, out of which 22 are proposed for the first time. Based on our quantitative analysis and online survey, we also distill many implications for different stakeholders.

Acknowledgment

The authors greatly thank the 26 survey respondents for their valuable answers and the anonymous reviewers for their constructive comments. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1004202 and in part by the Laboratory of Software Engineering for Complex Systems.

References

[1] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[2] J. Lewis and M. Fowler, "Microservices: a definition of this new architectural term," https://martinfowler.com/articles/microservices.html, last accessed: September 2021.

[3] C. Richardson, "Who is using microservices?" https://microservices.io/articles/whoisusingmicroservices.html, last accessed: September 2021.

[4] M. Loukides and S. Swoyer, "Microservices adoption in 2020," https://www.oreilly.com/radar/microservices-adoption-in-2020/, last accessed: September 2021.

[5] V. Bourne, "2020 digital innovation benchmark," Tech. Rep., 2020, https://devops.com/survey-sees-massive-adoption-of-microservices/.

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

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

[8] *Replication Package of Submission 151 for SANER 2022*. Zenodo, Oct. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.5574860

[9] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," *Present and ulterior software engineering*, pp. 195–216, 2017.

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

[11] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[12] I. Nadareishvili, R. Mitra, M. McLarty, and M. Amundsen, *Microservice architecture: aligning principles, practices, and culture*. " O'Reilly Media, Inc.", 2016.

[13] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2016, pp. 44–51.

[14] C. Esposito, A. Castiglione, and K.-K. R. Choo, "Challenges in delivering software in the cloud as microservices," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 10–14, 2016.

[15] C. Rosen and E. Shihab, "What are mobile developers asking about? a large scale study using stack overflow," *Empirical Software Engineering*, vol. 21, no. 3, pp. 1192–1223, 2016.

[16] M. U. Haque, L. H. Iwaya, and M. A. Babar, "Challenges in docker development: A large-scale study using stack overflow," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020, pp. 1–11.

[17] K. Bajaj, K. Pattabiraman, and A. Mesbah, "Mining questions asked by web developers," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 112–121.

[18] A. Bandeira, C. A. Medeiros, M. Paixao, and P. H. Maia, "We need to talk about microservices: An analysis from the discussions on stackoverflow," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 255–259.

[19] S. Baltes, L. Dumani, C. Treude, and S. Diehl, "Sotorrent: reconstructing and analyzing the evolution of stack overflow posts," in *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, A. Zaidman, Y. Kamei, and E. Hill, Eds. ACM, 2018, pp. 319–330. [Online]. Available: https://doi.org/10.1145/3196398.3196430

[20] J. Wen, Z. Chen, Y. Liu, Y. Lou, Y. Ma, G. Huang, X. Jin, and X. Liu, "An empirical study on challenges of application development in serverless computing," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 416–428.

[21] R. Rehurek and P. Sojka, "Gensim–python framework for vector space modelling," *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, vol. 3, no. 2, 2011.

[22] T.-H. Chen, S. W. Thomas, and A. E. Hassan, "A survey on the use of topic models when mining software repositories," *Empirical Software Engineering*, vol. 21, no. 5, pp. 1843–1919, 2016.

[23] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," 2017, to appear.

[24] S. W. Thomas, A. E. Hassan, and D. Blostein, "Mining unstructured software repositories," in *Evolving Software Systems*. Springer, 2014, pp. 139–162.

[25] M. Openja, B. Adams, and F. Khomh, "Analysis of modern release engineering topics:–a large-scale study using stackoverflow–," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2020, pp. 104–114.

[26] Z. Wan, X. Xia, and A. E. Hassan, "What do programmers discuss about blockchain? a case study on the use of balanced lda and the reference architecture of a domain to capture online discussions about blockchain platforms across stack exchange communities," *IEEE Transactions on Software Engineering*, vol. 47, no. 7, pp. 1331–1349, 2021.

[27] M. Bagherzadeh and R. Khatchadourian, "Going big: a large-scale study on what big data developers ask," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 432–442.

[28] K. Indrasiri and P. Siriwardena, "Microservices for the enterprise," *Apress, Berkeley*, 2018.

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

[30] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[31] Y. Tian, W. Ng, J. Cao, and S. McIntosh, "Geek talents: Who are the top experts on github and stack overflow?" *CMC-COMPUTERS MATERIALS & CONTINUA*, vol. 61, no. 2, pp. 465–479, 2019.

[32] B. Vasilescu, V. Filkov, and A. Serebrenik, "Stackoverflow and github: Associations between software development and crowdsourced knowledge," in *2013 International Conference on Social Computing*. IEEE, 2013, pp. 188–195.

[33] G. Gousios, A. Zaidman, M.-A. Storey, and A. v. Deursen, "Work practices and challenges in pull-based development: The integrator's perspective," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, 2015, pp. 358–368.

[34] Y. Zhang, B. Vasilescu, H. Wang, and V. Filkov, "One size does not fit all: an empirical study of containerized continuous deployment workflows," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 295–306.

[35] T. Cerny, M. J. Donahoo, and M. Trnka, "Contextual understanding of microservice architecture: current and future directions," *ACM SIGAPP Applied Computing Review*, vol. 17, no. 4, pp. 29–45, 2018.

[36] N. C. Mendonça, P. Jamshidi, D. Garlan, and C. Pahl, "Developing self-adaptive microservice systems: Challenges and directions," *IEEE Software*, 2019.

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

[38] S. Li, H. Zhang, Z. Jia, Z. Li, C. Zhang, J. Li, Q. Gao, J. Ge, and Z. Shan, "A dataflow-driven approach to identifying microservices from monolithic applications," *Journal of Systems and Software*, vol. 157, p. 110380, 2019.

[39] Q. Xiang, X. Peng, C. He, H. Wang, T. Xie, D. Liu, G. Zhang, and Y. Cai, "No free lunch: Microservice practices reconsidered in industry," 2021.

[40] L. Chen, "Microservices: Architecting for continuous delivery and devops," in *2018 IEEE International Conference on Software Architecture (ICSA)*, 2018, pp. 39–397.

[41] U. Zdun, E. Wittern, and P. Leitner, "Emerging trends, challenges, and experiences in devops and microservice apis," *IEEE Software*, vol. 37, no. 1, pp. 87–91, 2019.

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

[43] M. Alshangiti, H. Sapkota, P. K. Murukannaiah, X. Liu, and Q. Yu, "Why is developing machine learning applications challenging? a study on stack overflow posts," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2019, pp. 1–11.

[44] Z. Chen, H. Yao, Y. Lou, Y. Cao, Y. Liu, H. Wang, and X. Liu, “An empirical study on deployment faults of deep learning based mobile applications,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 674–685.

[45] S. V. Zykov, *Managing software crisis: a smart way to enterprise agility*. Springer, 2018, vol. 92.

[46] M. Waseem, P. Liang, and M. Shahin, “A systematic mapping study on microservices architecture in devops,” *Journal of Systems and Software*, vol. 170, p. 110798, 2020.

[47] M. Kalske, N. Mäkitalo, and T. Mikkonen, “Challenges when moving from monolith to microservice architecture,” in *International Conference on Web Engineering*. Springer, 2017, pp. 32–47.

[48] D. Richter, M. Konrad, K. Utecht, and A. Polze, “Highly-available applications on unreliable infrastructure: Microservice architectures in practice,” in *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2017, pp. 130–137.

[49] W. K. A. N. Dias and P. Siriwardena, “Microservices security in action,” https://otonomo.io/redis-kafka-or-rabbitmq-which-microservices-message-broker-to-choose/, last accessed: October 2021.

[50] Https://dev.to/tranthanhdeveloper/point-to-point-and-publish-subscribe-messaging-model-41j0, last accessed: October 2021.

[51] W. K. A. N. Dias and P. Siriwardena, *Microservices Security in Action*. Simon and Schuster, 2020.

[52] Https://kinsta.com/blog/google-cloud-vs-aws/. Retrieved on September 15, 2021.

[53] A. N. Meyer, E. T. Barr, C. Bird, and T. Zimmermann, “Today was a good day: The daily life of software developers,” *IEEE Transactions on Software Engineering*, vol. 47, no. 5, pp. 863–880, 2021.

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

[55] S. Li, H. Zhang, Z. Jia, C. Zhong, C. Zhang, Z. Shan, J. Shen, and M. A. Babar, “Understanding and addressing quality attributes of microservices architecture: A systematic literature review,” *Information and Software Technology*, vol. 131, p. 106449, 2021.

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

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

[58] S. Hassan, R. Bahsoon, and R. Kazman, “Microservice transition and its granularity problem: A systematic mapping study,” *Software: Practice and Experience*, vol. 50, no. 9, pp. 1651–1681, 2020.