Information and Software Technology 137 (2021) 106600

Contents lists available at ScienceDirect

Information and Software Technology

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

From monolithic systems to Microservices: An assessment framework

Florian Auer [a,*], Valentina Lenarduzzi [b], Michael Felderer [a,c], Davide Taibi [d]

[a] University of Innsbruck, Austria
[b] LUT University, Finland
[c] Blekinge Institute of Technology, Sweden
[d] Tampere University, Finland

ARTICLE INFO

Keywords:
Microservices
Cloud migration
Software measurement

ABSTRACT

Context: Re-architecting monolithic systems with Microservices-based architecture is a common trend. Various companies are migrating to Microservices for different reasons. However, making such an important decision like re-architecting an entire system must be based on real facts and not only on gut feelings.
Objective: The goal of this work is to propose an evidence-based decision support framework for companies that need to migrate to Microservices, based on the analysis of a set of characteristics and metrics they should collect before re-architecting their monolithic system.
Method: We conducted a survey done in the form of interviews with professionals to derive the assessment framework based on Grounded Theory.
Results: We identified a set consisting of information and metrics that companies can use to decide whether to migrate to Microservices or not. The proposed assessment framework, based on the aforementioned metrics, could be useful for companies if they need to migrate to Microservices and do not want to run the risk of failing to consider some important information.

1. Introduction

Microservices are becoming more and more popular. Big players such as Amazon,[1] Netflix,[2] Spotify,[3] as well as small and medium-sized enterprises are developing Microservices-based systems [1].

Microservices are autonomous services deployed independently, with a single and clearly defined purpose [2]. Microservices propose vertically decomposing applications into a subset of business-driven independent services. Each service can be developed, deployed, and tested independently by different development teams and using different technology stacks. Microservices have a variety of different advantages. They can be developed in different programming languages, can scale independently from other services, and can be deployed on the hardware that best suits their needs. Moreover, because of their size, they are easier to maintain and more fault-tolerant since the failure of one service will not disrupt the whole system, which could happen in a monolithic system. However, the migration to Microservices is not an easy task [1,3]. Companies commonly start the migration without any experience with Microservices, only rarely hiring a consultant to support them during the migration [1,3].

Various companies are adopting Microservices since they believe that it will facilitate their software maintenance. In addition, companies hope to improve the delegation of responsibilities among teams. Furthermore, there are still some companies that refactor their applications with a Microservices-based architecture just to follow the current trend [1,3].

The economic impact of such a change is not negligible, and taking such an important decision to re-architect an existing system should always be based on solid information, so as to ensure that the migration will allow achieving the expected benefits.

In this work, we propose an evidence-based decision support framework to allow companies, and especially software architects, to make their decision on migrating monolithic systems to Microservices based on the evaluation of a set of objective measures regarding their systems. The framework supports companies in discussing and analyzing potential benefits and drawbacks of the migration and re-architecting process.

* Corresponding author.
E-mail addresses: florian.auer@uibk.ac.at (F. Auer), valentina.lenarduzzi@lut.fi (V. Lenarduzzi), michael.felderer@uibk.ac.at (M. Felderer), davide.taibi@tuni.fi (D. Taibi).

[1] https://gigaom.com/2011/10/12/419-the-biggest-thing-amazon-got-right-the-platform/
[2] http://nginx.com/blog/Microservices-at-netflix-architectural-best-practices/
[3] www.infoq.com/presentations/linkedin-Microservices-urn

https://doi.org/10.1016/j.infsof.2021.106600
Received 31 January 2020; Received in revised form 1 March 2021; Accepted 14 April 2021
Available online 30 April 2021

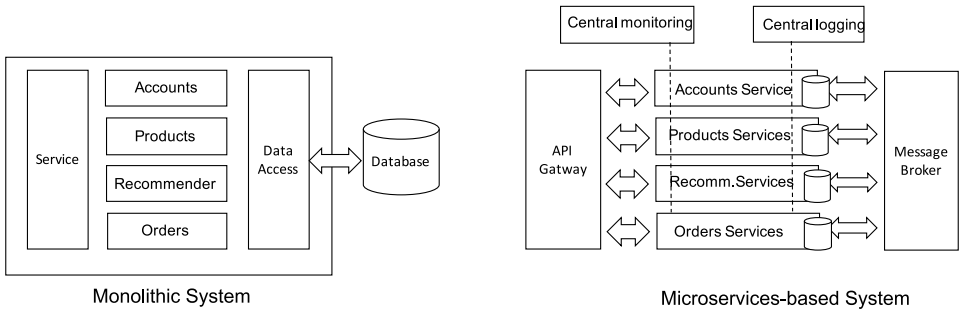

Fig. 1. Comparison between Microservices and monolithic architectures.

For this purpose we designed and conducted interviews with experienced practitioners as participants, to understand which characteristics and metrics they had considered before the migration and which they should have considered, comparing the usefulness of the collection of these characteristics. Finally, based on the application of Grounded Theory on the interviews, we developed our decision support framework.

Paper structure. Section 2 presents the background and Section 3 the related work. Section 4 presents the design and the results of the survey. In Section 5, we present the defined framework. In Section 6, we discuss the results we obtained and the defined framework. In Section 7, we identify threats to the validity of this work. Finally, we draw conclusions in Section 8 and highlight future work.

2. Background

The Microservice architecture pattern emerged from Service-Oriented Architecture (SOA). Although services in SOA have dedicated responsibilities, too, they are not independent. The services in such an architecture cannot be turned on or off independently. This is because the individual services are neither full-stack (e.g., the same database is shared among multiple services) nor fully autonomous (e.g., service A depends on service B). As a result, services in SOA cannot be deployed independently.

In contrast, Microservices are independent, deployable, and have a lot of advantages in terms of continuous delivery compared to SOA services. They can be developed in different programming languages, can scale independently from other services, and can be deployed on the hardware that best suits their needs because of their autonomous characteristics. Moreover, their typically small size, compared to large monolithic systems, facilitates maintainability and improves the fault tolerance of the services. One consequence of this architecture is that the failure of one service will not disrupt the whole system, which could happen in a monolithic system [2]. Nevertheless, the overall system architecture changes dramatically (see Fig. 1). One monolithic service is broken down into several Microservices. Thus, not only the service's internal architecture changes, but also the requirements on the environment. Each Microservice can be considered as a full-stack that requires a full environment (e.g., its own database, its own service interface). Hence, coordination among the services is needed.

3. Related work

In this section, we analyze the characteristics and measures adopted by previous studies, in order to classify the characteristics and metrics adopted in empirical studies that compared monolithic and Microservices-based systems.

3.1. Microservice migration

Many studies concerning specific characteristics of them have already been published. However, there are still some challenges in understanding how to develop such kinds of architectures [4–6]. A few secondary studies in the field of Microservices (i.e., [3,7–10] and [11]) have synthesized the research in this field and provide an overview of the state of the art and further research directions.

Di Francesco et al. [7] studied a large corpus of 71 studies in order to identify the current state of the art on Microservices architecture. They found that the number of publications about Microservices sharply increased in 2015. In addition, they observed that most publications are spread across many publication venues and concluded that the field is rooted in practice. In their follow-up work, Di Francesco et al. [8], provided an improved version, considering 103 papers.

Pahl et al. [11] covered 21 studies. They discovered, among other things, that most papers are about technological reviews, test environments, and use case architectures. Furthermore, they found no large-scale empirical evaluation of Microservices. These observations made them conclude that the field is still immature. Furthermore, they stated a lack of deployment of Microservice examples beyond large corporations like Netflix.

Soldani et al. [3] identified and provided a taxonomic classification comparing the existing gray literature on the pains and gains of Microservices, from design to development. They considered 51 industrial studies. Based on the results, they prepared a catalog of migration and re-architecting patterns in order to facilitate re-architecting non-cloud-native architectures during migration to a cloud-native Microservices-based architecture.

All studies agree that it is not clear when companies should migrate to Microservices and which characteristics the companies or the software should have in order to benefit from the advantages of Microservices.

Thus, our work is an attempt to close this gap by providing a set of characteristics and measures together with an assessment framework, as designed in our previous proposal [12].

3.2. Characteristics and measures investigated in empirical studies on microservices

Different product and process characteristics and measures have been investigated in the literature while comparing monolithic systems with Microservices architectures.

Different studies focused only on product characteristics [13–23], on process characteristics [13,20,22–24] or on both [13,15,20,22,23,25–27]. Moreover, other studies [13,16,24] investigated and compared costs. Furthermore, other studies, investigated several characteristics at the same time [13].

As for the product characteristics, the most frequently addressed one is performance (see Table 2). In detail, the papers [13–20,22] have a focus on performance. This is followed by scalability, which is discussed by the papers [14–19,21], and [22]. Other characteristics like

Table 1
Product-related measures.

Characteristic	Measures
Performance	**Response time:** The time between sending a request and receiving the corresponding response. This is a common metric for measuring the performance impact of approaches [13,15–17,19,20,22]. **CPU utilization:** The percentage of time the CPU is not idle. Used to measure performance. [20] reports the relationship between the number of VMs and the overall VMs utilization. In addition, [22] analyzes the impact of the decision between VMs and containers on CPU utilization. **Impact of programming language:** Communication between Microservices is network-based. Hence, network input and output operations require a considerable amount of the total processing time. The network performance is influenced amongst others by the selection of the programming language. That is due to the different implementations of the communication protocols. [18] analyzed the impact of design decisions on the performance and recommend specific programming languages for specific ranges of network message sizes. However, considering scalability in the system design seems to mitigate programming language impact. An example therefore is the routing mechanism for Microservices proposed in [19]. **Path length:** The number of CPU instructions to process a client request. [14] reports that the length of the code path of a Microservice application developed using Java with a hardware configuration of one core, using a bare process, docker host, and docker bridge, is nearly twice as high as in a monolithic system. **Usage of containers:** The usage of containers can influence performance, since they need additional computational time compared to monolithic applications deployed in a single container. [18] reports that the impact of containers on performance might not always be negligible. **Waiting time:** The time a service request spends in a waiting queue before it gets processed. [17,21] discuss the relationship between waiting time and number of services. Furthermore, [19] mentions an architecture design that halves the waiting time compared to other design scenarios.
Scalability	**Number of requests per minute or second:** (also referred to as throughput [14,16,22] or average latency [15,18]), is a performance metric. [22] found that in their experimental setting, the container-based scenario could perform more requests per second than the VM-based scenario. **Number of features per Microservice:** [21] points out that the number of features per Microservice affects scalability, influences communication overhead, and impacts performance.
Availability	**Downtime:** Microservice might suffer of downtime, if the system is not properly designed [15,28,29]. **Mean time to recover:** The mean time it takes to repair a failure and return back to operations. [20] uses this measure to quantify availability. **Mean time to failure:** The mean time until the first failure. [20] uses this measure together with mean time to recover as a proxy for availability.
Maintenance	**Complexity:** [13,16] notes that Microservices reduce the complexity of a monolithic application by breaking it down into a set of services. However, some development activities like testing may become more complex [16]. Furthermore, [18] state that the usage of different languages for different Microservices increases the overall complexity. **Testability:** [23] concludes that the loose coupling of Microservices at the application's front-end level improves testability.

Table 2
Process-related factors.

Characteristic	Measures
Process-related benefits	**Development independence between teams:** The migration from a monolithic architecture to a Microservices-oriented one changes the way in which the development team is organized. Typically, a development team is reorganized around the Microservices into small, cross-functional, and self-managed teams [13,15,20,23,24]. **Continuous delivery:** [13] notes that the deployment in a Microservices environment is more complex, given the high number of deployment targets. Hence, the authors of [13] suggest automating the deployment as much as possible. **Reusability:** Microservices are designed to be independent of their environment and other services [22]. This facilitates their reusability.

Table 3
Cost-related measures.

Characteristic	Measure
Personnel Cost	**Development costs:** [16] argues that Microservices reduce the development costs given that complex monolithic applications are broken down into a set of services that only provide a single functionality. Furthermore, most changes affect only one service instead of the whole system.
Infrastructure Cost	**Cost per hour:** Is a measure used to determine the infrastructure costs [13]. According to the experiment done in [24], the Microservices architecture had lower infrastructure costs compared to monolithic designs. **Cost per million requests:** In comparison to cost per hour, this measure is based on the number of requests/usage of the infrastructure. [24] uses the infrastructure costs of a million requests to compare different deployment scenarios.

availability [15,20] or maintenance [13,16,18,23] are considered only in a few papers.

Overall, related works identified the following characteristics as reported in Tables 1, 2, and 3:

- **Product**
  - Performance
  - Scalability
  - Availability
  - Maintenance
- **Process**
- **Cost**
  - Personnel Cost
  - Infrastructure Cost

From the literature, we also identified 18 measures for measuring product process and cost, as reported in Tables 1, 2, and 3.

Product-related measures. We identified 13 measures (Table 1) for the four identified sub-characteristics (performance, scalability, availability, and maintenance).

From the obtained results, we can see that the highest number of measures is related to performance and scalability, where we identified a total of nine studies referring to them. Among them, response time,

number of requests per minute or second, and waiting time are the most commonly addressed measures. For availability, we derived only three measures and for maintainability only two.

Process-related measures. Seven studies investigated the migration process using three factors: development independence between teams, usage of continuous delivery, and reusability (Table 2). These three factors can be considered as "Boolean measures" and can be used by companies to understand whether their process can be easily adapted to the development of Microservices-based systems.

Existing independent teams could easily migrate and benefit from the independence freedom provided by Microservices. Continuous delivery is a must in Microservices-based systems. The lack of a continuous delivery pipeline eliminates most of the benefits of Microservices. Reusability is amplified in Microservices. Therefore, systems that need to reuse the same business processes can benefit more from Microservices, while monolithic systems in which there is no need to reuse the same processes will not experience the same benefits.

Besides the analyzed characteristics, the papers also discuss several process-related benefits of the migration. Technological heterogeneity, scalability, continuous delivery support, and simplified maintenance are the most frequently mentioned benefits. Furthermore, the need for recruiting highly skilled developers and software architects is considered as a main motivation for migrating to Microservices.

Cost-comparison-related measures. As for this characteristic, three studies include it in their analysis and consider three measures for the comparison (Table 3).

3.2.1. Microservices migration effects

The analysis of the characteristics and measures adopted in the empirical studies considered by the related works allowed us to classify a set of measures that are sensitive to variations when migrating to Microservices. The detailed mapping between the benefits and issues of each measure is reported in Table 1.

Product Characteristics. Regarding product characteristics, performance is slightly reduced in Microservices.

When considering the different measures adopted to measure **performance**, the usage of containers turned out to decrease performance. This is also confirmed by the higher number of CPU instructions needed to process a client request (path length), which is at least double that of monolithic systems and therefore results in high CPU utilization. However, the impact of the usage of different programming languages in different services is negligible. Even if different protocols have different interpreters for different languages, the computational time is comparable.

When considering high **scalability** requirements, Microservices-based systems in general outperform monolithic systems in terms of resources needed. If a monolith is using all the resources, the way to handle more connections is to bring up a second instance. If a single microservice uses all the resources, only this service will need more instances. Since scaling is easy and precise, this means only the necessary amount of resources is used. As a result, for the same amount of money spent on resources, microservices deliver more throughput.

The **availability** of Microservices-based system can be affected by the higher number of moving parts compared to monolithic systems. However, differently than in monolithic systems, in the event of the failure of one Microservice, the remaining part of the system will still be available [1,3]. It is important to mention that Microservices do not provide high availability by default and maintaining high availability for microservices is not a simple task [28]. In order to investigate the practices to maintain high-availability in Microservices-based systems, Marquez et al. [29] conducted a survey among 40 practitioners, highlighting 12 practices. Examples of these practices are "Prevent remote procedure calls from waiting indefinitely for a response" or "Efficiently distributing incoming network traffic among groups of backend servers".

Maintenance is considered more expensive in the selected studies. The selected studies agree that the maintenance of a single Microservice is easier than maintaining the same feature in Microservices. However, testing is much more complex in Microservices [18], and the usage of different programming languages, the need for orchestration, and the overall system architecture increase the overall maintenance effort. Moreover, Microservices-based systems, should also take into account maintenance-related metrics between services, trying to reduce coupling and increase cohesion between services. For this purpose, the Structural Coupling (SC) [30] might be used to easily identify the coupling between services.

Cost-related measures The development effort of Microservices-based systems is reported to be higher than the development of monolithic systems [16]. However, [13] and [24] report that infrastructure costs are usually lower for Microservices than for monolithic systems, mainly because of the possibility to scale only the service that need more resources instead of scaling the whole monolith.

4. The survey

In this section, we present the survey on migration metrics that we performed as well as its results. We describe the research questions, the study design, the execution, and the data analysis, as well as the results of the survey.

4.1. Goal and research questions

We conducted a case study among developers and professionals in order to identify in practice which metrics they considered important before and after migration.

Based on our goal, we derived the following research questions (RQs):

RQ1. Why did companies migrate to Microservices?
RQ2. Which information/metrics was/were collected before and after the migration?
RQ3. Which information/metrics was/were considered useful by the practitioners?

With **RQ1**, we aim to understand the main reasons why companies migrated to Microservices, i.e., to understand whether they considered only metrics related to these reasons or other aspects as well. For example, we expect that companies that migrate to increase velocity considered velocity as a metric, but we also expect them to consider other information not related to velocity, such as maintenance effort or deployment time.

With **RQ2**, we want to understand the information/metrics that companies considered as decision factors for migrating to Microservices. However, we are also interested in understanding whether they also collected this information/these metrics during and after the development of Microservices-based systems.

With **RQ3,** we want to understand which information/metrics practitioners considered useful to collect the migration process, and which they did not collect but now believe they should have collected.

4.2. Study design

The information was collected by means of a questionnaire composed of five sections, as described in the following:

- ***Demographic information***: In order to define the respondents' profile, we collected demographic background information. This information considered predominant roles and relative experience. We also collected company information such as application domain, organization's size via number of employees, and number of employees in the respondents' own team.

- ***Project information***: We collected the following information on the project migrated to Microservices: creation and migration dates of the project.
- ***Migration motivations*** *(RQ1)*: In this section, we collected information on the reasons for migrating to microservices.
- ***Migration information/metrics*** *(RQ2)*: This section was composed of two main questions:
  - Which information/metrics were considered **before** the migration, to decide if migrate or not?
  - Which information/metrics were considered **after** the migration, to decide if migrate or not?
- ***Perceived usefulness of the collected information/metrics*** *(RQ3)*: In this section, we collected information on the usefulness of an assessment framework based on the metrics identified and ranked in the previous section. The goal was to understand whether the set of metrics could be useful for deciding whether to migrate a system or not in the future.
  This section was based on three questions:
  - Here we ask to rank how useful is each metric proposed in the Literature (Table 1) and mentioned by the interviewee to decide if migrate to microservices or not. The ranking is based on a 6-point Likert scale, where 1 means absolutely not and 6 absolutely.
  - How easy are the factors and measures to collect and use?
  - Which factor or measure is not easy to collect?
  - How useful is a possible discussion of the factors and measures reported in the previous questions before the migration? The ranking is based on a 6-point Likert scale, where 1 means absolutely not and 6 absolutely.
  - Do you think the factors or measures support a reasoned choice of migrating or not? (if not, please motivate)
  - Would you use this set of factors and measures in the future, in case of migration of other systems to Microservices? If not, please motivate.

The questionnaire adopted in the interviews is reported in Appendix.

4.3. Study execution

The survey was conducted over the course of five days, during the 19th International Conference on Agile Processes in Software Engineering, and Extreme Programming (XP 2018). We interviewed a total of 52 practitioners. We selected only experienced participants that successfully developed the microservices-based system and deployed in production. We did not consider any profiles coming from academia, such as researchers or students.

4.4. Data analysis

Two authors manually produced a transcript of the answers of each interview and then provided a hierarchical set of codes from all the transcribed answers, applying the open coding methodology [31]. The authors discussed and resolved coding discrepancies and then applied the axial coding methodology [31].

Nominal data was analyzed by determining the proportion of responses in each category. Ordinal data, such as 5-point Likert scales, was not converted into numerical equivalents since using a conversion from ordinal to numerical data entails the risk that any subsequent analysis will yield misleading results if the equidistance between the values cannot be guaranteed. Moreover, analyzing each value of the scale allowed us to better identify the potential distribution of the answers. Open questions were analyzed via open and selective coding [31]. The answers were interpreted by extracting concrete sets of similar answers and grouping them based on their perceived similarity.

Table 4
Role.

Role	#Answers
Developer	31
Project Manager	11
Agile Coach	2
Architect	2
Upper Manager	2
Other	5

Table 5
Experience (in years).

Experience in years	# Answers
Years ≤ 2	2
$2 <$ years ≤ 5	23
$5 <$ years ≤ 8	12
$8 <$ years ≤ 10	11
$10 <$ years ≤ 15	3
(no answer)	1

Table 6
Organization domain.

Organiz. Domain	# Answers
IT consultant	10
Banking	6
Software house	6
E-commerce	8
Other	9
(no answer)	13

Table 7
Team size.

# Team members	# Answers
$\# \leq 10$	14
$10 < \# \leq 20$	12
$20 < \# \leq 50$	7
$\# > 50$	1
(no answer)	18

4.5. Replication

In order to allow replication and extension of our work, we prepared a replication package with the results obtained.[4] The complete questionnaire is reported in Appendix.

4.6. Results

In this section, we will report the obtained results, including the demographic information regarding the respondents, information about the projects migrated to Microservices, and the answers to our research questions.

Demographic information. The respondents were mainly working as developers (31 out of 52) and project managers (11 out of 52), as shown in Table 4. The majority (23 out of 52) of them had between 2 and 5 years of experience in this role (Table 5). Regarding company information, out of the 52 respondents, 10 worked in IT consultant companies, 6 in software houses, 8 in e-commerce, and 6 in banks. The remaining 9 respondents who provided an answer worked in different domains (Table 6). The majority of the companies (15 out of 52 respondents) were small and medium-sized enterprises (SMEs) with a number of employees between 100 and 200, while 9 companies had less than 50 employees. We also interviewed people from 3 large companies with more than 300 employees (see Table 8). Regarding the team size, the vast majority of the teams had less than 50 members (33

[4] Raw data available at https://figshare.com/s/cb8314fb66163d9fcdc9.

Table 8
Organization size.

# Employees in organization	# Answers
# organization employees ≤ 50	9
50 < # organization employees ≤ 100	0
100 < # organization employees ≤ 200	15
200 < # organization employees ≤ 300	3
# organization employees > 300	8
(no answer)	19

Table 9
Application age.

Application age	# Answers
years < 5	18
5 < years ≤ 10	18
10 < years ≤ 15	9
15 < years ≤ 20	3
years > 20	5

Table 10
Migration time.

Migration time	# Answers
year ≤ 2	23
2 < year ≤ 4	20
4 < year	3
(no answer)	6

Table 11
Information/metrics considered before the migration mentioned at least three times.

Information/Metrics	# Answers
Number of bugs	16
Complexity	11
Maintenance effort	10
Velocity	6
Response time	6
Lines of code	3
Performance	3
Extensibility	3
Change frequency	3
Scalability	3

Table 12
Information/metrics considered after the migration, mentioned at least three times.

Information/Metrics	# Answers
Number of bugs	12
Complexity	9
Maintenance effort	7
Velocity	5
Scalability	5
Memory consumption	3
Extensibility	3

out of 52 respondents). 14 teams had less than 10 members, 12 teams had between 10 and 20 members, and 7 teams had between 20 and 50 members. Only one team was composed of more than 50 members (see Table 7).

Project information As for the project's age (see Table 9), about 69% of the respondents (36 out of 52) started the development less than 10 years ago, while 9 interviewees created the project between 10 and 15 years ago. Another 8 interviewees referred to projects with an age between 15 and 20 years, while 5 respondents started the development more than 20 years ago. As for the migration to Microservices, 23 respondents reported that the process started 2 years ago or less, while for 20 interviewees the process started between 2 and 4 years ago (see Table 10).

4.6.1. Migration motivations (RQ1)

In the answers to the question about the interviewees' motivation to migrate from their existing architecture to Microservices, a total of 97 reasons were mentioned. The open coding of the answers classified the 97 reasons into 22 motivations. In Fig. 2, all motivations that were mentioned three or more times are presented. The three main motivations are maintainability, deployability, and team organization.

The most commonly mentioned motivation was to improve the maintainability of the system (19 out of 97). They reported, among other things, that the maintenance of the existing system had become too expensive due to increased complexity, legacy technology, or size of the code base.

Deployability was another important motivation for many interviewees (12 out of 97). They expected improved deployability of their system after the migration. The improvement they hoped to achieve with the migration was a reduction of the delivery times of the software itself as well as of updates. Moreover, some interviewees saw the migration as an important enabler for automated deployment (continuous deployment).

The third most frequently mentioned motivation was not related to expected technical effects of the migration but was organizational in nature, namely team organization (11 out of 97). With the migration to Microservices, the interviewees expected to improve the autonomy of teams, delegate the responsibility placed on teams, and reduce the need for synchronization between teams.

The remaining motivations like cost, modularity, willingness to adopt microservices because other companies are also adopting them, or the reduction of the overall system complexity seem to be motivations that are part of the three main motivations discussed above, or at least influence one of them. For example, complexity was often mentioned in combination with maintenance, or scalability together with team organization. Thus, it appears that these three motivations are the main overall motivations for the migration from monoliths to Microservices.

4.6.2. Information/metrics collected before and after the migration (RQ2)

We collected 46 different pieces of information/metrics, which were considered a total of 107 times by the interviewees **before** the migration to Microservices. The three most commonly mentioned ones were the number of bugs, complexity, and maintenance effort (see Table 11), followed by the velocity, and response time. Other five motivations were mentioned less frequently.

Considering the information/metrics that collected **after** migration to Microservices, 26 clearly distinguishable types were identified that were mentioned a total of 66 times by the participants. Again, the number of bugs, complexity, and maintenance effort were the most frequently mentioned ones. (see Table 12).

As expected, the vast majority of the considered information/metrics was aimed at measuring characteristics related to the migration motivations. As maintainability was the most important reason to migrate to Microservices, maintainability-related metrics turned out to be the most important metrics considered before the migration. It is interesting to note that in some cases, companies collected this information before the migration but stopped collecting it during and after the migration (e.g., 4 interviewees out of 16 who had collected the number of bugs in their monolithic system did not collect the same information in the Microservices-based system).

The results suggest that the most important information needs remain the same from the start of the migration until its completion. Thus, there may be a set of migration information/metrics that is fundamentally important for the process of migration and that should be collected and measured throughout the migration.

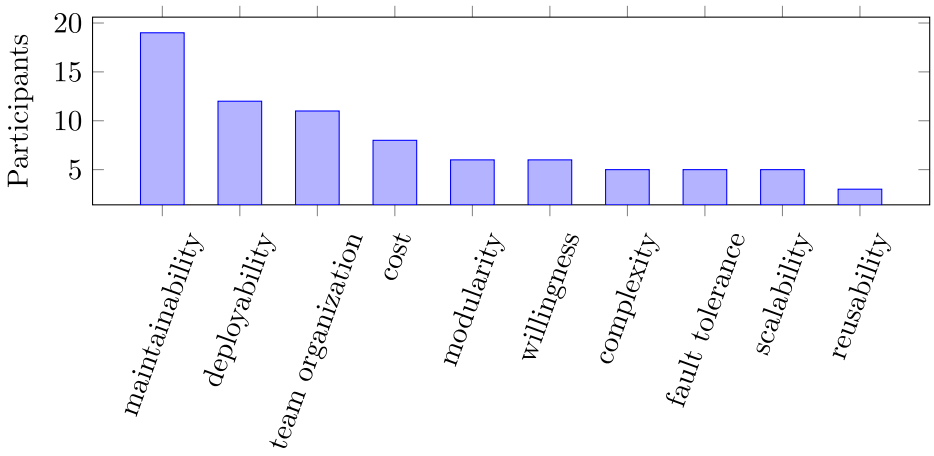

Fig. 2. Migration motivations mentioned by more than three participants.

Table 13
Information/metrics not easy to collect mentioned by more than one interviewee.

Metric	# Answers
Complexity	6
Testability	2
Response time	2
Benchmark data	2
Availability	2

4.6.3. Information/metrics considered useful (RQ3)

In this section, we will report the results on the perceived usefulness of the metrics collected.

Asking the interviewees how **easy** they think it is **to collect** the factors and measures proposed, 41 answered that they considered them as easy, while 10 did not consider them as easy (one interviewee did not provide an answer to this question). While entering into the details of the metrics not easy to collect, only a limited number of interviewees mentioned some metrics as complex. 20 different metrics were reported, but only complexity was mentioned by six interviewees while four metrics (testability, response time, benchmark data, and availability) were considered as complex by only two interviewee and the remaining 15 metrics were mentioned only by one participant (see Table 13).

Almost all interviewees categorized the usefulness of the metrics as very useful (24 out of 52) or extremely useful (25 out of 52). Table 14 reports the medians for the usefulness of each metric reported by the interviewees. Considering the **usefulness** of a possible discussion of the factors and measures reported in RQ2 before the migration, the majority of the interviewees considered it as very useful to understand the importance of the migration. (see Table 15). Furthermore, all but three interviewees confirmed that they believe that the metrics support a rational choice on whether to migrate or not.

Table 14
How useful did the interviewees consider each metric before migration.

Usefulness	Median
Response time (the time between sending a request and receiving the corresponding response)	4
Cpu utilization (the percentage of time the cpu is not idle)	4
Path length (the number of cpu instructions to process a client request)	4
Waiting time (the time a service request spends in a waiting queue before it get processed)	4
Impact of programming language (communication between microservices are network based)	4
Usage of containers (the usage of containers can influence the performance, since they need additional computational time compared to monolithic applications deployed in a single container)	4
Number of features per microservices	4.5
Number of requests per minute or second (also referred as throughput or average latency)	5
Downtime	5
Mean time to recover (the mean time it takes to repair a failure and return back to operations)	5
Mean time to failure (the mean time till the first failure)	5
Testability	5
Complexity	5
Development independence between teams (the migration from a monolithic architecture to a microservice oriented changes the way in which the development team is organized)	5
Continuous delivery	5
Reusability	5
Personnel cost (development cost)	4
Infrastructure cost (cost per hour)	4
Infrastructure cost (cost per million of requests)	4

Likert scale: 1—Absolutely not, 2—Little, 3—Just enough, 4—More than enough, 5—Very/a lot, 6—Extremely useful.

Table 15
How useful did the interviewees consider discussion of the set of information/metrics before migration.

Usefulness	# Answers
Absolutely not	0
Little	0
Just enough	1
More than enough	2
Very/a lot	24
Absolutely	25

Finally, 65% (34 out of 52) of the interviewees stated that they would consider the set of information/metrics proposed in the future, before migrating to microservice.

5. The assessment framework

In this section, we propose an evidence-based assessment framework based on the characteristics that should be considered before migration to identify and measure potential benefits and issues of the migration. The framework is evidence-based in the sense of evidence-based software engineering [32] as it has not been derived based purely on subjective experience of the authors, but rigorously based on a systematic literature study and a survey.

The goal of the framework is to support companies in reasoning about the usefulness of migration and make decisions based on real facts and actual issues regarding their existing monolithic systems. The framework is not aimed at prescribing a specific decision, such as recommending to migrate based on a specific metric, but it is aimed at helping companies to not miss important aspects and to reason on the most complete set of information before deciding to migrate or not. The framework therefore has to be tailored to the specific contexts of companies that apply it.

Based on the results obtained in our survey (Section 4), we grouped the different pieces of information and metrics into homogeneous categories, based on the classification proposed by the ISO/IEC 25010 standard [33]. However, we also considered two extra categories not included in ISO/IEC 25010, which focus on product characteristics, namely cost and processes.

The framework is applied in four steps:

Step 1 Motivation reasons identification
Step 2 Metrics identification
Step 3 Migration decisions
Step 4 Migration

In the next sub-sections, we will describe each of the four steps in detail.

5.1. Motivations reasons identification

Before migrating to Microservices, companies should clarify why they are migrating and discuss their motivation. As highlighted by previous studies [1,3], companies migrate to Microservice for various reasons and often migrate to solve some issues that need to be solved differently. Moreover, sometimes the migration can have negative impacts, for instance when companies do not have enough expertise or only have a small team that cannot work on different independent projects. The quality characteristics listed in Table 16 could be used as a checklist to determine whether there is some common problem in the system that the company intends to solve with the migration.

Based on the motivation, companies should reason – optimally including the whole team in the process – on whether the migration could be the solution to their problems or whether it could create more issues than benefits. If, for any reason, it is not possible to include the whole team in this discussion, we recommend including at least the project manager and a software architect, ideally with knowledge about Microservices.

In case the team still wants to migrate to Microservices after this initial discussion, it could start discussing how to collect the metrics (Step 2).

5.2. Step 2 - metrics identification

In order to finalize the decision on whether or not to migrate to Microservices, teams should first analyze their existing monolithic system. The system should be analyzed by considering the metrics reported in Table 16.

We recommend starting by considering the information and metrics related to the motivation for the migration. However, for the sake of completeness, we recommend discussing the whole set of metrics. For example, if a team needs to migrate to Microservices because of maintenance issues, they should not only consider the block "maintenance" but should also consider the remaining metrics, since other related information such as the independence between teams (process-related) could still be very relevant for maintenance purposes.

The list of metrics reported in Table 16 is not meant to be complete for each characteristic, but is rather to be used as a reference guide for companies to help them consider all possible aspects that could affect the migration. For example, a company's monolithic system might suffer from performance issues (characteristic "Functional Suitability"). The analysis of the sub-characteristics will help them to reason about "Overall performance", but they could also consider whether it is a problem related to "Time behavior" by analyzing the metric "Response time" and also considering the other sub-characteristics listed. However, if the motivation of the performance issue is different, the company will also be able to reason about it.

5.3. Migration decisions

After a thorough discussion of the collected metrics, the team can decide whether to migrate or not based on the results of the discussion performed in the previous step.

For example, there will be cases where a company may decide not to migrate after all. If the company realizes that the reason for the low performance is due to the inefficient implementation of an algorithm, they might decide to implement it better. If the main issue is cost of maintenance and the company wants to migrate mainly to reduce this cost, they might think of better team allocation or reason about the root causes of the high costs, instead of migrating with the hope that the investment will enable them to save money.

5.4. Migration

The team can then start the migration to Microservices. During this phase, we recommend that companies automate measurement of the relevant metrics and set up measurement tools to continuously collect relevant information as identified in Step 2.

6. Discussion

In this section, we will discuss the implications of this work.

The results of our survey are in line with the characteristics identified by the related work. The vast majority of the interviewees migrated to Microservices in order to improve maintainability [1,3]. However, deployability, team organization (such as the independence between teams), and cost are also important characteristics mentioned frequently in the interviews and not considered as important by previous work. Modularity, complexity, fault tolerance, scalability, and reusability were mentioned several times as well.

The proposed framework therefore covers (sub-)characteristics that take the results of the survey into account and are aligned with the

Table 16
The proposed assessment framework.

Characteristic	Sub-characteristic	Measure	Metric
Functional suitability			
	Appropriateness		System requirements understandability
Performance efficiency			
	Overall		
	Time behavior		Response time, throughput, …
	Resource utilization		Memory, disk space, nodes, …
	Compliance	Scalability	
	Other		#Requests
Reliability			
	Overall		Mean Time to Failure Mean Time to Repair Mean Time Between Failure
	Availability		%Availability Mean Time Between Downtimes
	Fault tolerance		#Bugs Code coverage
		Impact of failures	#Feature blocked, …
	Other	Backups	
Maintainability			
	Overall		
	Modularity	Code complexity Adopted patterns	
	Reusability		
	Testability		Code coverage
	Analyzability		#Microservices Complexity (code, data, …) Interactions between services
	Modifiability	Code size Change frequency Coupling Service responsibilities	#Lines of code, …
	Changeability		Extensibility
Cost			
	Overall		Development, testing, deployment
	Infrastructure		Cloud/On-Premise infrastructural costs
	Effort		Overall development Testing, deployment, maintenance, …
Process related			
			Independence between teams #User stories done per sprint Data management Delivery time Deployment frequency Feature priorities Roadmap Service responsibilities Team alignment Velocity (lead time/time to release)

established ISO/IEC 25010 standard. The top-level characteristics are functional suitability, reliability, maintainability, cost, and process. The characteristics cover all the relevant sub-characteristics and metrics identified in the survey. For instance, modularity is a sub-characteristic of maintainability and scalability is a metric for performance efficiency.

Finally, the framework suggests concrete metrics for measuring the characteristics. Given that all discussed characteristics are covered by metrics identified in the papers, the metrics can be used as an initial tool set to measure the main influencing factors for migrating a monolithic system to Microservices. Some characteristics are not easy to quantify, however. For instance, testability has effectiveness and efficiency aspects that can only be approximated by different metrics [34], like the degree of coverage or the number of defects covered. The survey was used to confirm the metrics found and to identify additional ones. The metrics most commonly mentioned in the survey are the number of bugs, complexity, and maintenance effort. It turns out that for the characteristics that are most relevant for migration, these metrics are also mentioned more often than for other characteristics. Maintainability is mentioned as the most important reason for migration, and maintainability-related metrics are also highlighted as the most important metrics.

In our study, we discovered that practitioners often do not properly measure their product, process, and cost before migrating to Microservices and realize only later (during or after the migration) that relevant information is missing. Our proposed assessment framework should not only help to identify the most relevant characteristics and metrics for migration, but also make professionals aware of the importance of measurement before, during, and after migration to Microservices. In

addition, there has not been a clear understanding what to measure before migrating to Microservices. Our proposed assessment framework intends to fill this gap. However, evaluation and refinement of the framework in industrial case studies is required as part of future work.

7. Threats to validity

We applied the structure suggested by Yin [35] to report threats to the validity of this study and measures for mitigating them. We report internal validity, external validity, construct validity, and reliability. As we performed a mixed-methods approach comprising a Systematic Mapping Study and a survey, we will identify in this section different threats to validity regarding both parts of our study.

7.1. Threats to validity regarding the survey

Internal Validity. One limitation that is always a part of survey research is that surveys can only reveal the perceptions of the respondents which might not fully represent reality. However, our analysis was performed by means of semi-structured interviews, which gave the interviewers the possibility to request additional information regarding unclear or imprecise statements by the respondents. The responses were analyzed and quality-checked by a team of four researchers.

External Validity. Overall, a total of 52 practitioners were interviewed at the 19th International Conference on Agile Processes in Software Engineering, and Extreme Programming (XP 2018). We considered only experienced respondents and did not accept any interviewees with an academic background. XP 2018 covers a broad range of participants from different domains who are interested in Microservices and the migration to Microservices. We therefore think that threats to external validity are reasonable. However, additional responses should be collected in the future.

The questions are aligned with standard terminology and cover the most relevant characteristics and metrics. In addition, the survey was conducted in interviews, which allowed both the interviewees and the interviewer to ask questions if something was unclear.

Reliability. The survey design, its execution, and the analysis followed a strict protocol, which allows replication of the survey. However, the open questions were analyzed qualitatively, which is always subjective to some extent, but the resulting codes were documented.

8. Conclusion

In this paper, we proposed an assessment framework to support companies in reasoning on the usefulness of the migration to Microservices.

We identified a set of characteristics and metrics that companies should discuss when they consider migrating to Microservices. The identification of these characteristics was performed by means of an industrial survey, where we interviewed 52 practitioners with experience in developing Microservices. The interviews were based on a questionnaire in which we asked the respondents to identify which metrics and characteristics had been adopted when they migrated to Microservices, which of these were useful, and which had not been adopted but should have been. The metrics were collected by means of open questions so as to avoid any bias of the results due to a set of predefined answers. After the open questions, we also asked the practitioners to check whether they had also collected some of the metrics proposed in the literature, and whether they believed it would have been useful to collect them.

The result of this work is an assessment framework that can support companies in discussing whether it is necessary for them to migrate or not. The framework will help them avoid migration if it is not necessary, especially when they might get better results by refactoring their monolithic system or re-structuring their internal organization.

Future work include the validation of the framework in industrial settings, and the identification of a set of automatically applicable measure, that could easily provide a set of meaningful information, reducing the subjectivity of the decisions. Another interesting future direction is the extension of this framework for different cloud-native technologies, including serverless [36,37] and Micro-Frontends [38]. It would be interesting to investigate frameworks to enable practitioners to understand when it is beneficial to migrate from monolithic to serverless functions, and in particular, which serverless pattern to adopt [39] to create microservices based on serverless functions without decreasing productivity or increase technical debt [40]

CRediT authorship contribution statement

Florian Auer: Conceptualization, Methodology, Writing - original draft. **Valentina Lenarduzzi:** Conceptualization, Methodology, Writing - original draft. **Michael Felderer:** Supervision, Reviewing and Editing, Funding acquisition. **Davide Taibi:** Supervision, Reviewing and Editing, Funding acquisition.

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 was partially supported by the Austrian Science Fund (FWF): I 4701-N and by the Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK), Austria, the Federal Ministry for Digital and Economic Affairs (BMDW), Austria, and the Province of Upper Austria in the frame of the COMET - Competence Centers for Excellent Technologies Programme managed by Austrian Research Promotion Agency FFG.

Appendix. The survey

In this Section we report the questionnaire adopted in the interviews.

Demographic information

- Company name
- Respondent name
- Respondent email address
- Role in the organization
  - Upper Manager
  - Manager
  - Developer
  - Other
- How many years have you spent in your role?
- Number of employees of your team
- Number of employees of your organization
- Organization's domain(s)

Project Information

- Which microservices-based application is your company developing?
- When was the application first created?
- When did your company decide to migrate to microservices?

Migration Motivations

- Why did your company decide to migrate?

Migration Information/Metrics

- Which information/metrics were considered **before** the migration?
- Which information/metrics were considered **after** the migration?

Perceived usefulness of the collected Information/Metrics

- We developed a set of factors and measures to support companies in evaluating the migration to microservices before they start, based on the assessment of a set of information to support them in reasoning about the needs of migrating.
- Which of the following information/metrics do you consider useful to collect and discuss before the migration?

Scalability/Performance	Absolutely not	Little	Just enough	More than enough	Very/a lot	Absolutely
- Response time (The time between sending a request and receiving the corresponding response)						
- CPU utilization (The percentage of time the CPU is not idle)						
- Path length (The number of CPU instructions to process a client request)						
- Waiting time (The time a service request spends in a waiting queue before it get processed)						
- Impact of programming language (Communication between microservices are network based)						
- Usage of containers (The usage of containers can influence the performance, since they need additional computational time compared to monolithic applications deployed in a single container)						
- Number of features per microservices						
- Number of requests per minute or second (Also referred as throughput or average latency)						
Availability						
- Downtime						
- Mean time to recover (The mean time it takes to repair a failure and return back to operations)						
- Mean time to failure (The mean time till the first failure)						
Maintenance						
- Testability						
- Complexity						
Process related benefits						
- Development independence between teams (The migration from a monolithic architecture to a microservice oriented changes the way in which the development team is organized)						
- Continuous delivery						
- Reusability						
Personnel Cost						
- Development Cost						
Infrastructure Cost						
- Cost per hour						
- Cost per million of requests						
Which other factors or measures should be considered? (please list and rank them)						

- How useful would you consider a discussion of the previous information before migration?
- Do you think the factors or measures support a reasoned choice of migrating or not? (if not, please motivate)
- How easy is the set of factors and measures to collect and use?
- Is there any measure that is not easy to collect?
- Would you use this set of factors and measures in the future, in case of migration of other systems to microservices? If not, please motivate.

References

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

[2] J. Lewis, M. Fowler, Microservices, 2014, www.martinfowler.com/articles/microservices.html.

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

[4] A. Balalaie, A. Heydarnoori, P. Jamshidi, Microservices architecture enables DevOps: Migration to a cloud-native architecture, IEEE Softw. 33 (3) (2016) 42–52.

[5] D. Taibi, V. Lenarduzzi, C. Pahl, Microservices anti-patterns: A taxonomy, in: Microservices - Science and Engineering, Springer, 2019.

[6] D. Taibi, V. Lenarduzzi, On the definition of microservice bad smells, IEEE Softw. 35 (3) (2018) 56–62.

[7] P.D. Francesco, I. Malavolta, P. Lago, Research on architecting microservices: Trends, focus, and potential for industrial adoption, in: 2017 IEEE International Conference on Software Architecture, ICSA, 2017, pp. 21–30.

[8] P.D. Francesco, P. Lago, I. Malavolta, Architecting with microservices: A systematic mapping study, J. Syst. Softw. 150 (2019) 77–97.

[9] D. Taibi, V. Lenarduzzi, C. Pahl, Microservices architectural, code and organizational anti-patterns, in: Cloud Computing and Services Science. CLOSER 2018 Selected papers. Communications in Computer and Information Science, 2019, pp. 126–151.

[10] D. Taibi, V. Lenarduzzi, C. Pahl, Architectural patterns for microservices: A systematic mapping study, in: 8th International Conference on Cloud Computing and Services Science, CLOSER2018, 2018.

[11] C. Pahl, P. Jamshidi, Microservices: A systematic mapping study, in: Proceedings of the 6th International Conference on Cloud Computing and Services Science - Volume 1 and 2, CLOSER 2016, SCITEPRESS - Science and Technology Publications, Lda, Portugal, 2016, pp. 137–146.

[12] F. Auer, M. Felderer, V. Lenarduzzi, Towards defining a microservice migration framework, in: Proceedings of the 19th International Conference on Agile Software Development: Companion, XP '18, ACM, New York, NY, USA, 2018, pp. 27:1–27:2.

[13] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, S. Gil, Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud, in: Computing Colombian Conference, 10CCC, 2015, pp. 583–590.

[14] T. Ueda, T. Nakaike, M. Ohara, Workload characterization for microservices, in: International Symposium on Workload Characterization, IISWC, 2016, pp. 1–10.

[15] V. Heorhiadi, S. Rajagopalan, H. Jamjoom, M.K. Reiter, V. Sekar, Gremlin: Systematic resilience testing of microservices, in: International Conference on Distributed Computing Systems, ICDCS, 2016, pp. 57–66.

[16] A. de Camargo, I. Salvadori, R.d.S. Mello, F. Siqueira, An architecture to automate performance tests on microservices, in: International Conference on Information Integration and Web-Based Applications and Services, 2016, pp. 422–429.

[17] H. Khazaei, C. Barna, N. Beigi-Mohammadi, M. Litoiu, Efficiency analysis of provisioning microservices, in: International Conference on Cloud Computing Technology and Science, CloudCom, 2016, pp. 261–268.

[18] N. Kratzke, P.-C. Quint, Investigation of impacts on network performance in the advance of a microservice design, in: Cloud Computing and Services Science, 2017, pp. 187–208.

[19] N.H. Do, T. Van Do, X. Thi Tran, L. Farkas, C. Rotter, A scalable routing mechanism for stateful microservices, in: Conference on Innovations in Clouds, Internet and Networks, ICIN, 2017, pp. 72–78.

[20] M. Gribaudo, M. Iacono, D. Manini, Performance evaluation of replication policies in microservice based architectures, Electron. Notes Theor. Comput. Sci. 337 (2018) 45–65, International Workshop on the Practical Application of Stochastic Modelling (PASM).

[21] S. Klock, J.M.E.M. van der Werf, J.P. Guelen, S. Jansen, Workload-based clustering of coherent feature sets in microservice architectures, in: International Conference on Software Architecture, ICSA, 2017, pp. 11–20.

[22] T. Salah, M.J. Zemerly, C.Y. Yeun, M. Al-Qutayri, Y. Al-Hammadi, Performance comparison between container-based and VM-based services, in: Conference on Innovations in Clouds, Internet and Networks, ICIN, 2017, pp. 185–190.
[23] H. Harms, C. Rogowski, L. Lo Iacono, Guidelines for adopting frontend architectures and patterns in microservices-based systems, in: Joint Meeting on Foundations of Software Engineering, in: ESEC/FSE 2017, 2017, pp. 902–907.
[24] M. Villamizar, O. Garcés, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, M. Lang, Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures, in: International Symposium on Cluster, Cloud and Grid Computing, CCGrid, 2016, pp. 179–182.
[25] D. Taibi, K. Systä, A decomposition and metric-based evaluation framework for microservices, in: Cloud Computing and Services Science, 2020, pp. 133–149.
[26] D. Taibi, K. Systä, From monolithic systems to microservices: A decomposition framework based on process mining, in: CLOSER2019, 2019, pp. 153–164.
[27] V. Lenarduzzi, F. Lomio, N. Saarimäki, D. Taibi, Does migrating a monolithic system to microservices decrease the technical debt? J. Syst. Softw. 169 (2020) 110710.
[28] V. Saquicela, G. Campoverde, J. Avila, M.E. Fajardo, Building microservices for scalability and availability: Step by step, from beginning to end, in: J. Mejia, M. Muñoz, Á. Rocha, Y. Quiñonez (Eds.), New Perspectives in Software Engineering, Springer International Publishing, Cham, 2021, pp. 169–184.
[29] G. Márquez, J. Soldani, F. Ponce, H. Astudillo, Frameworks and high-availability in microservices: An industrial survey, in: C.P. Ayala, L. Murta, D.S. Cruzes, E. Figueiredo, C. Silva, J.L. de la Vara, B. de França, M. Solari, G.H. Travassos, I. Machado (Eds.), Proceedings of the XXIII Iberoamerican Conference on Software Engineering, CIbSE 2020, 2020, pp. 57–70.
[30] S. Panichella, M.R. Imranur, D. Taibi, Structural coupling for microservices, in: 11th International Conference on Cloud Computing and Services Science, 2021.
[31] B. Wuetherick, Basics of qualitative research: Techniques and procedures for developing grounded theory, Canad. J. Univ. Contin. Educ. 36 (2010).
[32] B.A. Kitchenham, T. Dyba, M. Jorgensen, Evidence-based software engineering, in: Proceedings. 26th International Conference on Software Engineering, IEEE, 2004, pp. 273–281.
[33] ISO/IEC, ISO/IEC 25010:2011 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models, 2011.
[34] V. Garousi, M. Felderer, F.N. Kılıçaslan, A survey on software testability, Inf. Softw. Technol. (2018).
[35] R. Yin, Case Study Research: Design and Methods, fourth ed., in: Applied Social Research Methods, vol. 5, SAGE Publications, Inc, 2009.
[36] J. Nupponen, D. Taibi, Serverless: What it is, what to do and what not to do, in: 2020 IEEE International Conference on Software Architecture Companion, ICSA-C, 2020, pp. 49–50.
[37] D. Taibi, J. Spillner, K. Wawruch, Serverless computing-where are we now, and where are we heading? IEEE Softw. 38 (1) (2021) 25–31.
[38] S. Peltonen, L. Mezzalira, D. Taibi, Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review, Inf. Softw. Technol. 136 (2021) 106571.
[39] D. Taibi, N. El Ioini, P. Claus, J.R.S. Niederkofler, Patterns for serverless functions (function-as-a-service): A multivocal literature review, in: Proceedings of the 10th International Conference on Cloud Computing and Services Science, 2020, pp. 181–192.
[40] V. Lenarduzzi, J. Daly, A. Martini, S. Panichella, D.A. Tamburri, Toward a technical debt conceptualization for serverless computing, IEEE Softw. 38 (1) (2021) 40–47.