Enterprise Modelling and Information Systems Architectures
Vol. 14, No. 1 (2019). DOI:10.18417/emisa.14.1

Drivers and Barriers for Microservice Adoption – A Survey among Professionals in Germany

Holger Knoche[*,a], Wilhelm Hasselbring[a]

[a] Software Engineering Group, University of Kiel, 24118 Kiel, Germany

Abstract. *Microservices are an architectural style for software which currently receives a lot of attention in both industry and academia. Several companies employ microservice architectures with great success, and there is a wealth of blog posts praising their advantages. Especially so-called Internet-scale systems use them to satisfy their enormous scalability requirements and to rapidly deliver new features to their users. But microservices are not only popular with large, Internet-scale systems. Many traditional companies are also considering whether microservices are a viable option for their applications. However, these companies may have other motivations to employ microservices, and see other barriers which could prevent them from adopting microservices. Furthermore, these drivers and barriers possibly differ among industry sectors. In this article, we present the results of a survey on drivers and barriers for microservice adoption among professionals in Germany. In addition to overall drivers and barriers, we particularly focus on the use of microservices to modernize existing software, with special emphasis on implications for runtime performance and transactionality. We observe interesting differences between early adopters who emphasize scalability of their Internet-scale systems, compared to traditional companies which emphasize maintainability of their IT systems.*

Keywords. Microservice architecture • Survey • Software modernization • Microservice adoption

Communicated by S. Strecker. Received 2017-08-05. Accepted after 3 revisions on 2018-11-28.

1 Introduction

Microservices are an architectural style which has gained much attention in the last few years. Broad interest in microservices started around 2014 and has been steadily rising ever since (Balalaie et al. 2016). However, implementations of this architectural style have been around for a much longer time, although the term itself was coined only in 2012 (Lewis and Fowler 2014). For instance, one of the best-known early adopters, the video streaming provider NETFLIX, started introducing a microservice architecture in 2008 in order to leverage the advantages of cloud computing.[1] As of today, numerous well-known companies, such as AMAZON,[2] SPOTIFY,[3] and UBER,[4] use microservices. With this architectural style, these companies claim to have achieved the scalability required for providing their services to millions of users all over the world. In addition to scalability (Hasselbring 2016), microservices may furthermore enable both agility and reliability (Hasselbring and Steinacker 2017).

What is particularly noteworthy about the development of microservices is the fact that many companies are making their knowledge and tools

* Corresponding author.
E-mail. hkn@informatik.uni-kiel.de

[1] See Ruslan Meshenberg's talk at GOTO 2016, available at https://www.youtube.com/watch?v=57UK46qfBLY

[2] See Chris Munns's talk at I Love APIs 2015, slides available at https://www.slideshare.net/apigee/i-love-apis-2015-microservices-at-amazon-54487258

[3] See Kevin Goldsmith's talk at GOTO 2015, available at https://www.youtube.com/watch?v=7LGPeBgNFuU

[4] See https://eng.uber.com/soa

publicly available. For instance, Netflix[5] and Otto[6] publish technology blogs, on which current ideas and experiences are presented. In addition, many libraries and infrastructure components to develop and run microservices at scale are published free of charge as open source software. Examples of such software include the Archaius library for configuration management,[7] the Zuul edge gateway,[8] or the container instrumentation platform Kubernetes.[9] Finally, cloud providers such as Amazon Web Services allow to obtain the required computing resources quickly and with little effort. Thus, the entry threshold for adopting microservices is quite low.

As a consequence, many companies are currently considering whether microservices are a viable option for their software systems. However, many of these systems are not "Internet-scale"; instead, they are used by a known, limited, and stable number of users. Therefore, these companies may consider microservices for other reasons than the early adopters. Even more interesting are expected barriers which may prevent these companies from adopting microservices. Several authors warn against considering microservices as viable for every software system, as there are numerous trade-offs to consider (Killalea 2016; Singleton 2016). Furthermore, the reasons for and against microservices may vary considerably between different industries.

Although microservice adoption is discussed extensively in blog posts and other online media, there is yet little research data on the subject. While some studies on microservice adoption in practice exist, many of them have only been conducted with few participants, and several open questions still remain. In order to gain insight into the reasons why "traditional" companies are considering the adoption of microservices, we conducted a survey among software development professionals in Germany. Since many companies already have existing software assets, we furthermore investigated to what extent microservices are perceived as a tool for software modernization, which goals are pursued by introducing microservices into existing software, and how the potential impact on runtime performance and transactionality is rated.

The remainder of this paper is structured as follows. In Sect. 2, background information about microservices and related concepts is presented. Related work is discussed in Sect. 3, and our research approach and design is outlined in Sect. 4. Results and findings are presented in Sect. 5. Threats to validity and implications of the results are discussed in Sect. 6. Sect. 7 concludes the paper.

2 Background

2.1 Properties of Microservice Architectures

There is no commonly accepted definition of microservice architectures. Instead, they are usually characterized by a set of common properties,[10] which are described in detail by Lewis and Fowler (2014). These properties are summarized briefly below. Afterwards, a working definition of the term for the scope of our study is presented.

As the term "microservice" suggests, services are the building blocks and main means of modularization in microservice architectures. Services run in separate process contexts, and can be individually deployed, replaced, and retired. Each microservice focuses on providing a single business function, following the single responsibility principle (SRP). The services are built around business capabilities by cross-functional teams, which are responsible for every aspect of the service from development to productive operation. The teams furthermore keep this responsibility for the entire lifetime of the service, as opposed to traditional projects where the responsibility is usually handed over to another team after the project's completion. This difference is commonly summarized as *"products, not projects"*.

[5] https://medium.com/netflix-techblog
[6] https://dev.otto.de
[7] https://github.com/Netflix/archaius
[8] https://github.com/Netflix/zuul
[9] http://kubernetes.io/

[10] As mentioned in Martin Fowler's talk at GOTO 2014, available at https://www.youtube.com/watch?v=wgdBVIX9ifA

In turn, the teams are granted a high degree of autonomy. In particular, centralized governance and data management are abandoned to the extent possible and feasible. This allows the team to choose the right tools for the job, such as, for instance, appropriate programming languages or databases. Furthermore, decisions can be changed or reverted without affecting other teams. A high degree of deployment and infrastructure automation allows the teams to deploy new versions of their services at their own decision.

This team autonomy, however, is only possible due to strict technical isolation and loose coupling. Each microservice is *technically self-contained*, which means that its deployment unit contains everything to run the service, often even including the operating system. Unlike so-called Self-Contained Systems,[11] microservices do not need to be *functionally self-contained*, i. e. they may invoke other microservices to provide their business function. The required inter-service communication occurs only using defined interfaces based on platform-independent data formats and technologies. Microservice implementations commonly rely on lightweight communication technologies, mostly web technologies such as HTTP/REST or messaging solutions such as Kafka or RabbitMQ. Feature-rich solutions such as enterprise service buses are usually discouraged, as they may tempt to move business logic from the services into the communication infrastructure. Instead, microservices advocate to keep the business logic completely inside the services (*“smart endpoints and dumb pipes”*).

Being a highly distributed architecture, microservices are particularly suspectible to partial failures. Therefore, microservices must be designed to cope gracefully with the unavailability of required services to prevent cascading failures, a property commonly referred to as *resilience*. Several patterns have emerged for this purpose (Nygard 2007), and well-tested implementations are available in libraries such as Hystrix.[12] Examples of such patterns are circuit-breaker[13] and bulkhead[14] , which are commonly used to alleviate the effects of unavailable or slow dependencies. Another important advantage of this resiliency is that it allows to deploy the services in an arbitrary order.

To ensure that their microservices are sufficiently resilient, some companies even deliberately inject failures into their productive environments, a discipline sometimes referred to as “Chaos Engineering”.[15] For instance, Netflix’s Simian Army[16] consists of a number of tools to cause different types of errors. The best-known of these is the Chaos Monkey tool, which randomly terminates virtual machines and application containers.

Microservice implementations are essentially stateless, with the exception of short-time caches for performance improvement and resilience. Together with the arbitrary deployment order, this allows automated deployment infrastructures (see Sect. 2.3) to start, stop, and move service instances with little restrictions. Databases and distributed caches, sometimes referred to as “stateful services”, are usually separated into dedicated containers. Often, especially databases are even operated traditionally.

To summarize, we use the following definition for the term “microservice” for the remainder of this paper.

Definition 1 (Microservice) A microservice is a technically self-contained and independently deployable software component that runs in its own process context and has its own means of persistency. It is developed and run by a cross-functional, autonomous team responsible for its

[11] http://scs-architecture.org

[12] https://github.com/Netflix/Hystrix

[13] https://docs.microsoft.com/en-us/azure/architecture/pa↩tterns/circuit-breaker

[14] https://docs.microsoft.com/en-us/azure/architecture/pa↩tterns/bulkhead

[15] http://principlesofchaos.org/

[16] https://github.com/Netflix/SimianArmy

entire lifecycle. A microservices provides a single business function, which is exposed using platform-independent interfaces.

2.2 Microservices, DevOps, and Continuous Delivery

It is obvious that the previously described properties cannot be provided by only technological means, such as a particular software architecture. The requirement of autonomous, cross-functional teams able to quickly and automatically deploy their services at their own discretion shows that microservices are tightly connected to the notions of DevOps (Hüttermann 2012) and Continuous Delivery (Humble and Farley 2011). However, there are different opinions on how precisely these notions relate. Fowler (2014) sees DevOps culture as a prerequisite for microservice adoption, while Wolff (2016) argues that microservices do not necessitate DevOps adoption. Bass et al. consider Continuous Delivery as a DevOps practice and remark that many companies which already have adopted Continuous Delivery are now moving towards a microservice architecture (Bass et al. 2015, p. 66). Balalaie et al. (2016), on the other hand, see microservices as an enabler for DevOps.

We agree most with the last opinion, as, to our experience, the organizational changes required for DevOps are almost impossible to implement without concrete proof of the technological benefits for the particular organization. We therefore also consider organizational implications of microservices in our survey, such as the role of microservices as an enabler for DevOps or the change of tasks for both development and operations teams.

2.3 Deployment Infrastructure for Microservices

The popularity of the microservice architecture has also led to some important technological developments in the field of deployment and operations. One of these developments is the rise of container-based virtualization, often used synonymous with its best-known implementation, DOCKER.[17]

[17] https://www.docker.com/

This lightweight virtualization mechanism allows for the provisioning of a new service instance in seconds, whereas setting up a traditional virtual machine usually takes several minutes. Therefore, container-based virtualization has become an important tool to achieve high elasticity.

In order to fully leverage this elasticity, automated cluster management tools have emerged. Well-known examples of such tools are KUBERNETES,[18] DOCKER SWARM, and DC/OS.[19] These tools provide features such as autoscaling, service discovery, and redeployment in case of node failure, and thus greatly facilitate the operation of microservice-based applications. But as mentioned before, such automated management usually requires the microservices to be deployable in an arbitrary order.

2.4 Consistency Implications of Microservices

As previously noted, the autonomy granted to microservice teams includes the choice of the underlying database(s). While this *polyglot persistence* approach allows to leverage the strengths of different database types such as relational or graph databases, it also has some important downsides, especially regarding consistency.

One major downside is that consistent updates across service boundaries are very difficult to achieve. In traditional, centralized databases, such updates are implemented using database transactions, which guarantee the well-known ACID (Atomicity, Consistency, Isolation, and Durability) properties.

Unfortunately, these properties are notoriously difficult to uphold for distributed persistency, where multiple different databases can be affected by a single transaction. Such cases are usually handled using a two-phase commit (2PC) protocol, where the participants first vote on whether the transaction can be committed before – provided that there are no objections – proceeding to the actual commit. Due to the high amount of coordination and synchronization required for this

[18] http://kubernetes.io/
[19] https://dcos.io/

protocol, two-phase commits can severely limit scalability.

Not every database provides transactions, not to mention the ability to participate in distributed transactions. Especially many so-called NoSQL databases have little to no transaction support in exchange for high scalability. Since strong consistency cannot be guaranteed, systems using such databases are often built to provide *eventual consistency*, meaning that if no further updates to a data item occur, all reads to this item will eventually return the value of the last update (Vogels 2009).

As a consequence, transactionless coordination between services is preferred for microservices (Lewis and Fowler 2014). In these settings, strategies like explicit compensation or the Try-Cancel-Confirm (TCC) approach (Pardon and Pautasso 2014) are employed to address data consistency. With explicit compensation, changes are made optimistically, and explicitly reverted using a compensating operation if necessary (e. g., a previously created customer is deleted). TCC makes use of entities explicitly representing temporary locks (e. g., a flight *reservation* valid for 5 minutes) which needs to be confirmed in case of success (e. g., the *reservation* is turned into a *booking*).

Another downside is that it can be very difficult to create consistent backups of the different data stores. This can have a significant impact on disaster recovery strategies, since the system cannot be reverted to a consistent state just by restoring backups. Therefore, appropriate measures must be taken to ensure the safety of the stored data. Technically, replication can decrease the risk of data loss due to failures, thus reducing the probability of having to restore backups. Conceptionally, temporal versioning of the data allows to reconstruct consistent states at least up to a specific point in time.

2.5 Microservices as a Means for Software Modernization

A particularly interesting aspect of microservices is that several authors, such as Newman (2015) and Wolff (2016), consider them as a viable means for modernizing existing, monolithic software applications. Furthermore, there are experience reports of several companies who have successfully replaced (parts of) their existing software by microservices, or are in the process of doing so. Although some companies, for instance German online retailer Otto, have succeeded in a complete re-write of their application (Hasselbring and Steinacker 2017), most authors recommend using an incremental approach (Knoche and Hasselbring 2018); some authors even consider a non-incremental approach as bad practice (Carrasco et al. 2018). To provide a concrete example, Stine (2015) proposes an approach comprising three major phases:

1. New features are implemented only as microservices. No new features are added to the monolith.
2. An interface layer is created which allows the newly created microservices to access the monolith's functionality. This interface layer serves as an anti-corruption layer (see Evans 2007) to clearly separate the old and new domain models.
3. Functionality is gradually removed from the monolith and re-implemented as microservices. Stine suggests to start with the services with the highest need for change. This process continues until the monolith is either completely replaced or contains only stable functionality which does not justify the extraction effort.

A closer look at this approach reveals several important challenges, and highlights that such a modernization is far from trivial. In particular, the monolith itself starts depending on the new microservices with the third phase. This can have important consequences:

- The anti-corruption layer is now used bidirectionally.
- Formerly native calls inside the monolith may have to be replaced by service calls, and can become significantly slower due to invocation overhead.

- Database accesses inside the monolith may have to be restructured or replaced by service calls.
- The monolith must now also be able to cope with unavailable dependencies.

Furthermore, the previously discussed restrictions on transactionality and consistency now also apply to the monolith itself, as it can, for instance, no longer assume that all its changes are contained within ACID transactions. These restrictions may pose a significant challenge for modernizations to microservices, and are therefore also investigated in our survey.

3 Related Work

Being a discipline that has only recently received academic attention, there is yet little empirical research that particularly targets microservices. Systematic mapping studies of the existing literature on microservices have been conducted by Pahl and Jamshidi (2016), Alshuqayran et al. (2016), and Di Francesco et al. (2017). The latter two studies also list implementation challenges and problems targeted by microservices that are commonly addressed in the literature. However, these challenges and problems are not discussed in detail. Furthermore, they may not be representative of the actual situation in practice, as Di Francesco et al. find that the majority of publications was written only by authors from academia.

Schermann et al. (2016) conducted a survey with 42 professional participants on the current state of practice in service-based applications. This study, however, aimed primarily at implementation details such as used protocols and data formats, collected monitoring metrics, and preferred programming languages. Motivations and barriers for adopting microservices are not addressed.

Taibi et al. (2017) report on a study based on interviews with 21 professional participants on motivations, issues, and processes for adopting microservices. Although this study has some overlap with ours, it does not address the aspect of migrating existing software towards a microservice architecture. Furthermore, there are fundamental differences regarding the methodology. While Taibi et al. chose a more qualitative approach with few participants, our study uses a quantitative approach with a significantly larger number of participants.

A survey with 25 professional participants on challenges in microservice design as well as suggestions on how to resolve them has been conducted by Ghofrani and Lübke (2018). While this study addresses drivers and challenges for microservice adoption, the results are only discussed very shortly, and especially the challenges are based on only a few responses.

Di Francesco et al. (2018) report on a survey with 18 professional participants regarding the activities and challenges while migrating an existing piece of software to microservices using an adapted variant of the *horseshoe model* (Razavian and Lago 2010). Although this study also addresses modernization using microservices, we see some important differences to our study. While Di Francesco et al. investigate concrete actions and issues while applying a given modernization strategy, we focus on identifying reasons that lead to the decision to modernize. The issue of changes in transactionality and data consistency, which we investigate in our study, is not addressed.

In addition to these academic studies, there are also industrial surveys partially concerned with microservices. Surveys conducted by NGINX (2016) and Lightbend, Inc. (2016) report on the usage of microservices in production. The latter study also briefly investigates drivers and barriers to microservice adoption, but only in very little detail.

To summarize, while there is already some empirical research on microservice adoption in practice, we still see many opportunities as well as need for further research in this area. Many of the existing studies have been conducted with quite few participants, and should therefore be complemented by further studies to verify the findings (see Sect. 6.2). Furthermore, the implications for transactionality and consistency, which differ significantly between traditional monoliths and microservices (see Sect. 2.4), have not yet been investigated.

4 Research Design and Method

4.1 Research Questions

In this study, we aim at investigating drivers and barriers for microservices in general as well as goals for using microservices as a means of software modernization. This resulted in the selection of the following research questions:

RQ1: What are primary drivers for companies to adopt microservices?

RQ2: What are primary barriers preventing companies from adopting microservices?

RQ3: What are important goals for using microservices to modernize existing software assets and do they differ from the overall drivers for adoption?

RQ4: How is the potential impact on runtime performance and transactionality/data consistency in modernization settings rated?

4.2 Research Method

To answer the presented research questions, we conducted a survey among professionals from Germany. In order to increase the relevance of the answers, we decided to primarily target professionals who were already concerned with microservices.

Therefore, we visited industry meetings and conferences concerned with microservices to acquire respondents personally, and contacted speakers of such conferences by mail and professional social networks. We also advertised our survey on a well-known German developer news site, HEISE DEVELOPER.[20] Furthermore, the respondents were asked to forward the survey to other professionals concerned with microservices. An attempt to acquire respondents from microservice-themed groups in the professional social network XING was unsuccessful due to very poor response rates.

For the survey, we used both paper and web-based questionnaires. The paper questionnaire was used at meetings where we were able to recruit respondents personally, while the web questionnaire was used for contact via electronic media. The questionnaires comprised a total of 19 questions (see Appendix A), and were designed to take 10 to 15 minutes to complete. The questions were selected by one of the authors of the study based on findings in the literature, experience from discussions with professionals, and his own professional experience in software modernization. The questionnaire was then reviewed by another researcher and a professional software developer, both already concerned with microservices. After the review, the questionnaire was tested by two professional software developers for usability and understandability.

[20] https://www.heise.de/developer/

4.3 Methodological Remarks

Several questions of our questionnaire asked the respondents to rank the importance of different aspects for particular decisions. Since we expected some aspects to be decisive, i. e., considered so important that they alone might lead to a decision, we used a four-point rating scale capable of capturing such extreme responses. We chose to use four points as this allowed for clearly separated, easy-to-understand labels and, being an even number, avoided central tendencies.

For comparing the importance of the items, we calculated a score as the weighted mean of responses for each item, as well as the standard deviation of the score. Thereby, we implicitly interpreted these ordinal items as interval-scaled. We are aware that this interpretation of such items is disputed and, from a statistical point of view, not fully sound (see Knapp 1990). As pointed out by Stevens (1946), this is mainly due to the intervals between the options being of unequal size. This is particularly true for our scale due to the extremal options, which are, as a consequence, underweighted in the scores. We therefore use the scores and standard deviations only for a rough ordering of the aspects as well as indications for the degree to which responses differed for a given item.

For checking hypotheses regarding the items, we employ statistical tests appropriate for ordinal data (sign tests and Wilcoxon rank sum tests). In order to keep the text coherent, we report only the

p-values in the text; details on the tests (hypotheses and test statistics) can be found in Appendix B. A test label (e. g., T_1) is provided for each p-value to facilitate the navigation in the appendix.

Furthermore, a few numbers in the stacked bar plots had to be adjusted by one percentage point to account for rounding as the plots required the percentages to sum up to exactly 100%.

5 Results and Findings

5.1 Demographical Information

In total, 71 respondents took part in our survey. As shown in Tab. 1, about half the respondents were concerned with microservices for less than a year, one third for one to two years, and one fifth for more than two years. The remaining respondents provided no or invalid answers.

Table 1: Time concerned with microservices

Experience	**# of respondents**	**% of respondents**
Up to 6 months	17	24%
6 months to 12 months	14	20%
12 to 24 months	20	28%
More than 24 months	12	17%
Invalid / no answer	8	11%

As for job roles and departments, 70% of the respondents stated working as architects, 13% as consultants, 56% as developers, and 15% as team leads. The vast majority of the respondents, almost 96%, worked in development departments, 21% worked in operations departments. The detailed numbers are shown in Tab. 2 and Tab. 3. The respondents were allowed to give multiple answers to both questions.

Table 2: Respondents and job roles

Role	**# of respondents**	**% of respondents**
Architect	50	70%
Consultant	9	13%
Developer	40	56%
Team Lead	11	15%

The respondents were furthermore asked to name the industry they work in.

Table 3: Respondents and departments

Department	**# of respondents**	**% of respondents**
Development	68	96%
Operations	15	21%

The responses were grouped manually into the following industry categories:

- *Software Development and Consulting*: Respondents from companies whose primary product is software. Since many such companies also do software-related consulting, this industry was also added to this category.
- *Energy and Manufacturing*: This category represents companies from traditional branches of industry, such as car manufacturers and energy suppliers.
- *Financial Services*: In this category, respondents from banks and insurance companies are grouped.
- *Retailing and E-Commerce*: Respondents from online retailing and other E-Commerce companies.
- *Other or no answer*: Respondents working in other industries and respondents that did not answer this question.

The grouping was done as to avoid small groups with only few participants. The category "Retailing / E-Commerce" was established despite having very few participants, since especially online retailing is one of the pioneering industries of microservices. These categories and the respective figures are shown in Tab. 4.

Table 4: Respondents and industries

	# of respondents	**% of respondents**
Development / Consulting	16	23%
Energy / Manufacturing	11	15%
Financial Services	20	28%
Retailing / E-Commerce	6	8%
Other / No answer	18	25%

5.2 Current Usage of Microservices

At the beginning of our questionnaire, we were concerned with the current extent of microservice usage in the respective companies and industries. As evident from Fig. 1, almost one third (27%) of all respondents stated that they or their customers already use microservices to a large extent.[21] The remaining respondents stated using microservices to a lesser extent, or not using microservices at all, in equal parts (37%).

A look at the industry-specific figures reveals that the degree of usage differs considerably between industries. While microservices seem to be heavily used by the retailing and e-commerce industry (median: *usage to a large extent*), they are only sparingly used by the financial services industry (median: *no usage*). When weighting the answers from -1 (*no usage*) to 1 (*usage to a large extent*), these differences are statistically significant (T_1: $p = 0.002$ and T_2: $p = 0.010$, respectively).

5.3 Drivers for Microservice Adoption

To identify important drivers for microservice adoption, we asked the respondents to rate nine properties commonly attributed to microservices with respect to their importance on the decision to adopt microservices. These potential drivers are discussed below; the actual properties are highlighted in italics.

One of the most commonly cited property of microservices is *high scalability and elasticity* (see, for instance, Hasselbring and Steinacker 2017) due to the services being independently deployable. Thus, instances can be automatically added and removed by cluster management tools like Kubernetes. A property closely related to scalability is that microservices are considered a cloud-native architecture, i. e., they allow to *leverage the advantages of the Cloud and container-based virtualization*. As noted in the introduction, well-known adopter Netflix introduced microservices for this very reason, and we therefore expect companies to consider microservices as a tool for their endeavor towards the Cloud. Due to the strong component separation, microservices are expected to be less prone than traditional monoliths to develop entangled dependencies, thus improving *maintainability* (see Newman 2015). This is particularly important for business applications, which are typically in use for several years, and may therefore be an important driver for adopting microservices. We were furthermore interested in whether the free choice of programming language and persistence (see Lewis and Fowler 2014), often referred to as *polyglot programming* and *polyglot persistence*, are actually considered as a driver for microservice adoption, as they also have several downsides as discussed later on.

As described in Sect. 2.2, microservices are also closely related to the more organizational notions of DevOps and Continous Delivery. Companies might embrace microservices to serve as an *enabler for DevOps and Continuous Delivery* (see Chen 2018). In particular, these approaches promise a *short "time to market"* for new features, which can be crucial for companies facing intense competition (Chen 2015). A further organizational advantage attributed to microservices is a better alignment of the development teams and the software artifacts (see Newman 2015; Wolff 2016), referred to as *organizational improvement* below. Since organizational misalignment is known to be detrimental to quality (Nagappan et al. 2008), improving this alignment is expected to cause an increase in quality. Finally, being a state-of-the-art architecture, microservices might be adopted by companies to increase their *attractiveness as an employer*.

The rating of these drivers was done using our four-point rating scale, with labels *crucial*, *relevant*, *hardly relevant*, and *irrelevant*. The respondents were asked to choose the *crucial* option for properties which, in their opinion, would on their own suffice to adopt microservices.

Tab. 5 summarizes the answers from the different industries, ordered by the overall score. The score is calculated as the weighted average of

[21] Note that customers were included to account for consultants who might only be concerned with microservices in customer projects.

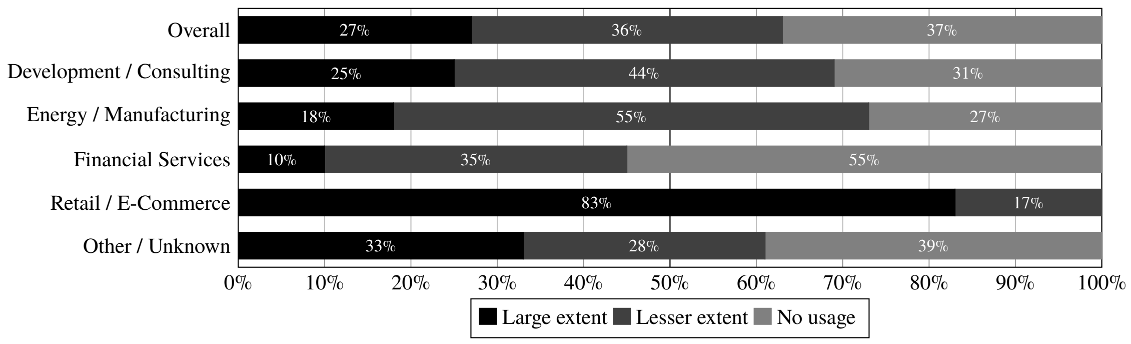

Figure 1: Current microservice usage in different industries (% of respondents)

the responses, with the weights ranging from 3 (*crucial*) to 0 (*irrelevant*).

As evident from the table, the primary overall drivers for microservice adoption were found to be scalability, maintainability, and time to market, which were each rated as crucial by roughly one third of the respondents. The difference between the primary and secondary drivers is statistically significant (T_3: $p < 0.001$). Secondary drivers are the potential role as an enabler for Continuous Delivery and DevOps, the suitedness for cloud environments and container-based virtualization, and organizational improvement. Polyglot persistence and polyglot programming are rated a bit less relevant. The potential effect of increasing the attractiveness as an employer is perceived as least important; the gap to the secondary drivers is significant (T_4: $p = 0.004$).

The industry-specific figures from Tab. 5 show that for development and consulting companies, the suitedness for virtualization platforms is rated significantly higher than in the other industries (T_5: $p = 0.014$). These companies also appear to perceive maintainability as the key driver for microservices; however, the difference to the other primary drivers is not significant (T_{12}). Polyglot programming is rated considerably higher in the energy and manufacturing (T_6: $p = 0.012$) and e-commerce industries (T_7: $p = 0.027$). The latter also rates the attractiveness as an employer much higher than the rest (T_8: $p = 0.018$), which is particularly interesting due to the high degree of adoption in this industry. There is, however, no significant difference regarding this question between "heavy users" and others across all industries (T_{13}).

Some drivers are also rated notably low in certain industries. Particularly striking are time to market and suitedness for virtualization in the energy and manufacturing industry, which are rated considerably lower than in other industries (T_{10}: $p = 0.023$ and T_9: $p = 0.012$, respectively).

Cross-correlating the items using Spearman's rank correlation coefficient revealed some notable, but medium to low correlations. The highest was between polyglot programming and persistence ($r_s = 0.47$, $p < 0.001$), which we found to be surprisingly low as both notions seem closely related. All cross-correlation results are shown in Tab. 10 in the appendix. However, the figures from Tab. 5 suggest that these two items may be perceived differently in different industries. While polyglot programming is ranked significantly higher than polyglot persistence in the energy and manufacturing industry (T_{11}: $p = 0.029$), the polyglot persistence is rated notably, but not significantly higher the financial services industry as well as the other industries (T_{14}).

5.4 Barriers for Microservice Adoption

In order to identify important barriers for microservice adoption, we proceeded in the same way as for the drivers. However, barriers to adoption are far less discussed in the literature. Therefore, we had

Table 5: Drivers for microservice adoption in different industries

	Overall		**Development / Consulting**	**Energy / Industry**	**Financial Services**	**Retail / E-Commerce**	**Other / Unknown**
	Score (mean)	SD	Score (mean)	Score (mean)	Score (mean)	Score (mean)	Score (mean)
High Scalability and Elasticity	2.14	0.72	2.19	2.18	2.15	2.67	1.89
High Maintainability	2.11	0.75	2.31	2.10	2.15	2.17	1.89
Short Time to Market	2.07	0.82	2.12	1.45	2.10	2.67	2.17
Enabler for CD and DevOps	1.61	0.89	1.69	1.27	1.70	1.83	1.56
Suitedness for Cloud and Docker	1.55	0.89	2.00	1.09	1.35	1.50	1.67
Organizational Improvement	1.37	0.81	1.31	1.18	1.50	1.83	1.22
Polyglot Programming	1.28	0.90	1.00	1.82	1.15	2.00	1.11
Polyglot Persistence	1.27	0.83	1.06	1.09	1.40	1.33	1.39
Attractiveness as Employer	0.87	0.86	1.00	0.55	0.85	1.67	0.72

to rely more on personal experience from an industrial project and discussions with practicioners as for the drivers. Since the items contain abstract barriers, such as resistance to change, as well as challenges implementing concrete measures, we decided to separate them into two separate questions. The items we asked the respondents to rate are discussed below. Again, the items themselves are highlighted in italics.

Based on our experience, *resistances* by both *operations staff* and *developers* can be a great barrier to microservice adoption, as microservices are in some ways very different from what the developers and operators are used to. For instance, we observe that the concept of eventual consistency is difficult to accept for developers who have been working with strict consistency for many years. Furthermore, especially the operations staff are often skeptical of the *maturity of the new technologies*, as well as the *compatibility with existing software systems*.

In addition to resisting to change, both developers and operators may *lack the skills* for adopting microservices. The high degree of team autonomy comes with a high degree of responsibility. The teams may now have to take care of numerous cross-cutting concerns that were previously done by specialized teams, such as security, data protection, monitoring, database administration, consistency, and backups. Especially the latter can be challenging with distributed persistence, as *consistent backups* – which are simple in centralized databases – are difficult to achieve. Notions like polyglot programming and persistence require developers to be proficient in multiple programming languages and persistence solutions. And due to their highly distributed nature, microservices yield more *complex deployments* than traditional monoliths (see Wolff 2016), i. e., a higher number of running instances as well as more frequent deployments and more complex interactions.

Another issue with teams autonomously choosing their tools is that companies must ensure sufficient *licensing and support contracts*. For microservice architectures, software components that are licensed per running instance, such as some commercial monitoring solutions, can be particularly costly. The organizational implications of adopting microservices may further be *incompatible with compliance and regulations*. For instance, companies may be required to adhere to given release processes, perform obligatory code reviews, or have change advisory boards (see Chen 2015) which can be difficult to reconcile with teams releasing software at their own decision. Furthermore, responsibility issues may arise, especially when running microservices in the Cloud (see Esposito et al. 2016).
The rating of these items was done similar to the drivers, with the difference that the relevance for *refraining* from adopting microservices was to be rated.

As evident from Tab. 6, insufficient skills of both operations and development staff as well as resistance by operations are seen as the premier overall barriers to microservice adoption, followed by deployment complexity. However, the overall scores of the items are quite low, with only insufficient skills of the operations staff being rated significantly as *relevant* (T_{15}: $p = 0.036$). The remaining items are rated quite low overall. However, compliance and regulations as well as resistances by developers are seen very differently in different industries, as shown by the high standard deviations for these items.

The cross-correlation of the items revealed several significant, but medium to low correlations. The highest were between the skill set and resistance of development and operations ($r_s = 0.42$, $p < 0.001$ and $r_s = 0.40$, $p < 0.001$), which come at no surprise.

The industry-specific figures suggest that the lack of consistent backups is especially important to financial services companies, where it received the third-highest score of all barriers. It was rated *relevant* by 60% of the respondents, and received a significantly higher score than in the other industries (T_{16}: $p = 0.003$).

Compliance and regulations might be an issue in the development and consulting industry, where it was rated *crucial* by 25% of the respondents, and in the financial services industry, where it received the highest score in all industries with a median of *relevant*. However, the score differences in both industries are not statistically significant (T_{18} and T_{19}, respectively). In contrast, this item received by far the lowest overall in the energy and manufacturing industry, and significantly lower than in the other industries (T_{17}: $p = 0.005$).

The role- and department-specific figures, which are not included for brevity, reveal that consultants and leads ranked lack of skills particularly high. For instance, 44% of the consultants rated insufficient operations skills as *crucial* (score: 2.11). Leads were particularly sceptical about insufficient developer skills, which were rated *crucial* by 36% (score: 1.82). A remarkable observation from the department-specific figures is that both departments agree on the insufficient skill sets being the most important barriers, with very similar scores (1.71, 1.70, 1.71, and 1.64).

The concrete implementation challenges were also rated on a four-point scale. However, since the respondents were to rate the difficulty of the implementation, the labels were *simple*, *medium*, *difficult*, and *impossible*. The challenges to be rated are described below.

In order to achieve the desired degree of team autonomy, delivery speed, and time to market, *automated deployment pipelines* are required. As observed by Leppänen et al. (2015), such pipelines prove to be difficult to implement in practice and may therefore pose a significant implementation challenge. As noted by Gmeiner et al. (2015), deployment pipelines require a high degree of test automation due to their frequent execution. This does not only apply to unit tests, but *integration tests and further test stages* also need to be *automated*. To our experience, many companies still rely heavily on manual testing in these stages.

For running the pipelines and tests as well as achieving elasticity in production, it is necessary to *provide resources quickly on demand*. This often means a significant technological and organizational change, as we observe that providing new resources and creating test environments is often a bureaucratic process involving manual work and a considerable amount of red tape. Similar to test automation, the creation and configuration of the necessary infrastructure needs to be automated as well, leading to formal descriptions of environments commonly referred to as *Infrastructure as Code*.

The second set of challenges we investigated is concerned with the shift of operations tasks to the development teams. On the one hand, this entails that *operations tasks are done by development teams*, including unpopular tasks such as pager duty. On the other hand, there is a *change of tasks for operations teams*, since several of their common tasks are automated as part of the deployment pipelines or taken over by the cross-functional teams. A closely related challenge is that *running distributed applications* is not easy, and can be very different from running traditional

Table 6: Barriers for microservice adoption in different industries

	Overall		**Development / Consulting**	**Energy / Industry**	**Financial Services**	**Retail / E-Commerce**	**Other / Unknown**
	Score (mean)	SD	Score (mean)	Score (mean)	Score (mean)	Score (mean)	Score (mean)
Insufficient Ops Skills	1.71	0.82	2.06	1.27	1.53	1.67	1.89
Resistance by Ops	1.66	0.87	1.88	1.64	1.68	1.33	1.56
Insufficient Dev Skills	1.63	0.95	1.69	1.00	1.79	1.67	1.78
Deployment Complexity	1.41	0.79	1.62	0.73	1.35	1.00	1.83
Compliance and Regulations	1.21	1.11	1.38	0.45	1.50	1.17	1.22
Compatibility Issues	1.13	0.96	1.06	1.09	1.05	0.50	1.50
Consistent Backups	1.13	0.88	1.00	0.73	1.60	1.00	1.00
Maturity of Technology	0.97	0.78	1.06	0.73	1.05	0.83	1.00
Resistance by Devs	0.97	1.06	1.06	1.18	1.10	0.50	0.78
Support Contracts / Licenses	0.89	0.77	0.56	0.73	1.20	0.50	1.06

monoliths. Therefore, the tasks do not just shift, but change as well.

We furthermore selected a few technological challenges we considered as potential barriers based on our experiences from industrial projects. Many companies have employed centralized persistence for several years. Therefore, *implementing decentralized and polyglot persistence* can be a major paradigm change. Similarly, *supporting polyglot programming* may require additional tooling, processes, and knowledge. And as microservices are suspectible to partial failure, sufficient *runtime monitoring* is required, which allows to detect and locate problems quickly. However, such monitoring requires careful preparation in order to gather the required information without significant degradation of runtime performance. As shown in Tab. 7 below, running distributed applications and the change of tasks for operations teams received the highest overall score from the respondents, followed by monitoring, ad-hoc resource provisioning, and decentralized persistence. But none of the challenges seems to be perceived as a "show stopper", since the overall scores are low and none of the items received a notable amount of *impossible* ratings. There are, however, considerable differences between the industries regarding particular challenges. For instance, ad-hoc provisioning received a much higher score in the financial services industry than in the other industries (T_{20}: $p = 0.037$), in particular, the development and consulting industry (T_{21}: $p = 0.003$). Supporting polyglot programming is also rated notably more difficult by respondents from the financial services industry (T_{22}: $p < 0.001$).

Cross-correlating the items did not reveal any particularly notable correlations. In order to analyze whether the participants attributed challenges to specific barriers, we furthermore cross-correlated the items of both questions. Surprisingly, the strongest correlation was found between consistent backups and polyglot programming ($r_s = 0.49$, $p < 0.001$), which may indicate that some respondents confused polyglot programming with polyglot persistence.

5.5 Applicability of Microservices

Business applications are often run as on-premise installations, and not in the Cloud. This is due to multiple reasons, a prominent one being concerns or regulations regarding the storage of personal data potentially anywhere in the world. As microservices are usually considered a cloud-native software architecture, the respondents were asked to rate the suitedness of microservices for on-premise installations for self-developed as well as licensed software, i. e. software developed by a third party. The major intention behind this question was to investigate to what extent microservices are considered viable for software products that are sold or licensed to be run at and by the respective customer. As noted by Olsson et al. (2012), continuously delivering software to customers may require new engagement models. Therefore, the

Table 7: Implementation challenges for microservice adoption in different industries

	Overall		Development / Consulting	Energy / Industry	Financial Services	Retail / E-Commerce	Other / Unknown
	Score (mean)	SD	Score (mean)	Score (mean)	Score (mean)	Score (mean)	Score (mean)
Running Distributed Apps	1.49	0.63	1.62	1.36	1.35	1.50	1.61
Change of Ops Tasks	1.46	0.73	1.75	1.09	1.45	1.67	1.39
Establishing Monitoring	1.38	0.64	1.38	1.00	1.40	1.17	1.67
Resource Provisioning	1.31	0.77	0.88	1.27	1.55	1.00	1.56
Decentralized Persistence	1.31	0.73	1.25	1.00	1.30	1.50	1.50
Test Automation	1.22	0.65	1.31	1.09	1.15	1.17	1.28
Ops Tasks by Dev Teams	1.13	0.78	1.25	1.00	1.16	1.17	1.06
Building Pipelines	1.10	0.74	0.81	1.27	1.15	0.83	1.28
Infrastructure as Code	1.04	0.79	0.81	0.82	1.10	0.83	1.41
Polyglot Programming	0.87	0.84	0.69	0.73	1.45	0.33	0.67

participants were hinted at elasticity and frequent deployments, and could rate microservices to be suited *perfectly*, *well*, *moderately*, or *not at all* for the respective scenario.

As shown in Tab. 8, the large majority (81%) of respondents considers microservices well-suited for self-developed software running on-premise. On the contrary, only 39% of the respondents rate them as such for licensed software. Based on discussions and the hints given to the participants, we assume that this difference is largely due to expected difficulties of running and frequently updating complex, distributed applications that are developed and delivered by a third party.

5.6 Modernization Goals for Existing Applications

As previously noted, microservices are also considered as a viable option for modernizing existing software. Two thirds of the respondents stated that there were plans or projects to introduce microservices to existing applications; the detailed numbers are shown in Fig. 2.

Particularly notable was that 92% of the respondents who stated to use microservices to a lesser extent answered this question with *yes*, as opposed to “heavy users” (79%) and non-users (32%). These numbers suggest that companies who are already using microservices to some extent are planning to increase their usage of microservices, while those who have not yet adopted microservices at all are unlikely to do so in the near future.

Asked whether they would only add new functionality or also replace existing functionality by microservices, 85% of the respondents stated that they would also replace existing functionality.

Similar to the drivers for microservice adoption, we intended to identify important modernization goals pursued by companies. For this purpose, we asked the respondents whether they would pursue the following modernization goals *primarily*, *secondarily*, or *not at all* by introducing microservices. The selection of goals is based on the drivers from Sect. 5.3 to allow comparisons between the results:

1. Improve the maintainability of the applications
2. Improve the time to market for new features
3. Improve the scalability of the applications
4. Improve the overall quality of the applications
5. Make preparations for Continuous Delivery or DevOps
6. Introduce new technology to the applications
7. Improve the team motivation

The answers are summarized in Tab. 9; the weights for the score range from 2 (*primarily*) to 0 (*not at all*). As obvious from the table, the premier overall modernization goal is improving the maintainability, followed by improving the time to market for new features and scalability. The score difference between maintainability and time to market is statistically significant (T_{23}: $p = 0.003$). Quality improvement and preparing Continuous Delivery

Table 8: Applicability of microservices for on-premise installations

	Overall		**Development / Consulting**	**Energy / Industry**	**Financial Services**	**Retail / E-Commerce**	**Other / Unknown**
	Score (mean)	SD	Score (mean)	Score (mean)	Score (mean)	Score (mean)	Score (mean)
Self-developed software on-premise	1.95	0.57	2.13	1.80	2.05	1.83	1.73
Licensed software on-premise	1.36	0.68	1.31	1.36	1.45	1.33	1.34

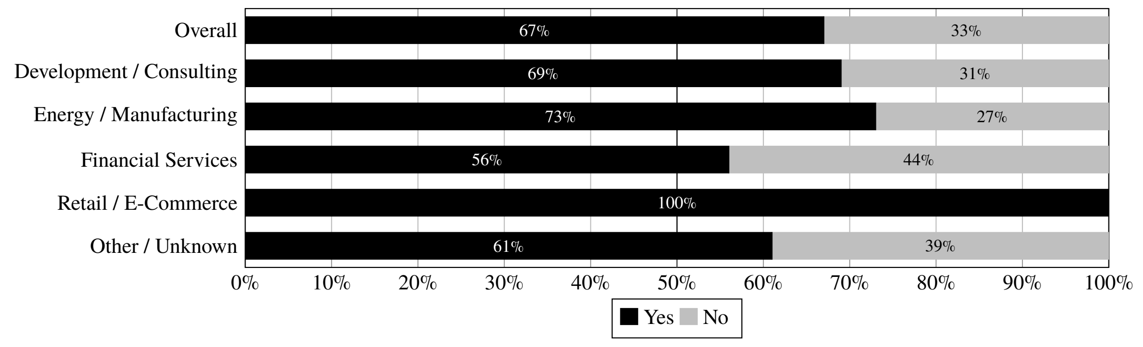

Figure 2: Plans for introducing microservices to existing applications in different industries (% of respondents)

and DevOps are also considered important. The introduction of new technology and motivational improvement are rated significantly less important than the other goals (T_{24}: $p = 0.002$). An accompanying open question as to which new technologies the respondents wanted to introduce yielded no notable results.

Cross-correlation revealed a notable correlation between improving maintainability and improving quality ($r_s = 0.42$, $p < 0.001$), which comes at no surprise. Further notable, but lower correlations were found between improving team motivation and the introduction of new technology ($r_s = 0.34$, $p = 0.003$) as well as improving the time to market ($r_s = 0.33$, $p = 0.005$).

As shown in Tab. 9, the relevance of goals differs slightly among industries. This is true for the roles as well. The most notable observation from the role-specific figures, which are not included for space reasons, is that leads rank quality and preparation for Continuous Delivery / DevOps (both 73% *primarily*, score: 1.73) significantly higher (T_{25}: $p = 0.030$ and T_{26}: $p = 0.042$, respectively) than the remaining roles.

5.7 Performance and Transactionality

One fundamental property of microservices is that each service runs in its own process context. Thus, service calls imply at least inter-process, in most cases even network communication.

Especially network communication is known to be orders of magnitude slower than in-process method calls (Litoiu 2004). Containers and overlay networks, which are commonly used in microservice architectures, can add an additional performance penalty (Kratzke 2015). Therefore, moving existing functionality into microservices might cause a considerable performance degradation of the existing software.

However, only 36% of the respondents expected this degradation to be serious or even critical. 51% expected only a minor degradation, 13% no degradation at all. Detailed numbers are shown in Fig. 3.

The introduction of remote service calls can also affect application performance in a second way. As shown by Knoche (2016), prolongating existing transaction contexts can increase the degree of lock contention inside a database and reduce overall transaction throughput. As noted in

Table 9: Modernization goals for existing applications

	Overall		**Development / Consulting**	**Energy / Industry**	**Financial Services**	**Retail / E-Commerce**	**Other / Unknown**
	Score (mean)	SD	Score (mean)	Score (mean)	Score (mean)	Score (mean)	Score (mean)
Improve Maintainability	1.79	0.48	1.81	2.00	1.85	1.50	1.67
Improve Time to Market	1.54	0.63	1.88	1.09	1.60	1.83	1.33
Improve Scalability	1.46	0.58	1.44	1.64	1.60	1.50	1.22
Improve Quality	1.39	0.64	1.06	1.64	1.50	1.50	1.39
Prepare CD and DevOps	1.39	0.69	1.56	1.45	1.25	1.67	1.28
Introduce New Technology	1.04	0.75	0.88	1.09	0.90	1.50	1.17
Improve Team Motivation	0.97	0.68	0.94	0.82	1.10	1.50	0.78

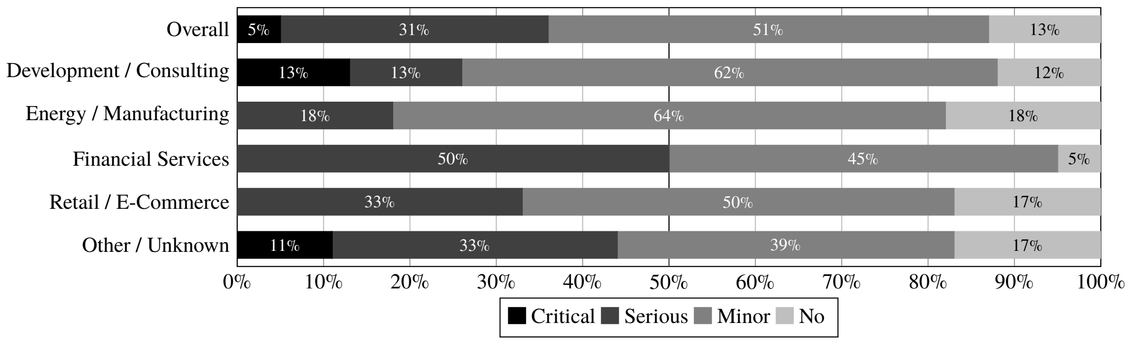

Figure 3: Expected performance degradation due to inter-process and network communication (% of respondents)

the exemplary modernization process in Sect. 2.5, service invocations are incrementally inserted into the modernized monolith, which still contains its original transactional contexts. Even if the invoked services do not participate in the transaction, the invocation overhead may lead to a prolongation of the monolith's transaction contexts.

The majority of the respondents (66%) considered this a potential problem, but expected it to occur only in specific cases. 18% of the respondents saw this as a serious problem, especially in the financial services industry, where 30% of the respondents rated it as such. The least concern was expressed by respondents from the energy and manufacturing industry. The detailed numbers are shown in Fig. 4.

As discussed earlier, cross-service ACID transactionality is discouraged in microservice architectures and usually not available. Since ACID transactions are ubiquitous in business software, this can be perceived as a loss, especially in modernization settings. Overall, 13% of the respondents rated the unavailability of such transactions as a critical, and 45% as a serious issue. 32% considered this a minor problem, and 10% did not see a problem at all. As obvious from Fig. 5, the issue is seen particularly critical in the energy and manufacturing sector, where it was rated as such by 36% of the respondents.

Unlike cross-service transactions, service-internal transactions are not discouraged at all in microservice architectures. 79% of the respondents considered it very important (27%) or important (52%) to incorporate transactional boundaries during service design. No respondent considered this issue unimportant. As evident from Fig. 6, the numbers vary among industry sectors, with a notably low rating in the retailing and e-commerce sector.

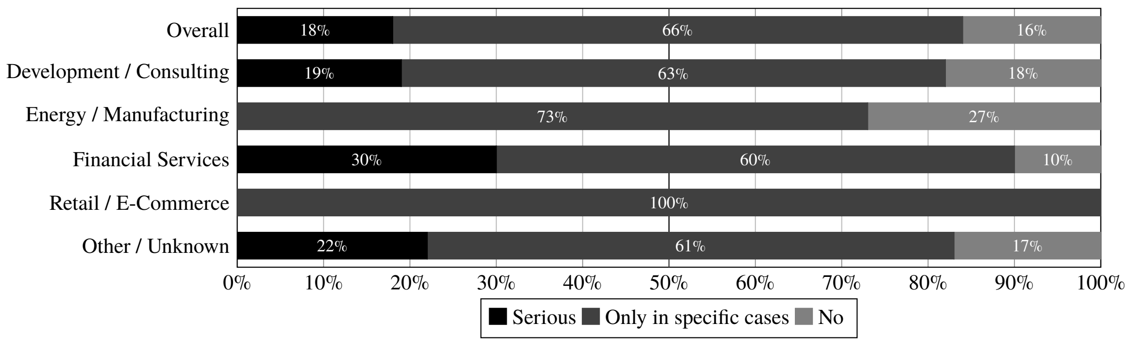

Figure 4: Expected performance degradation in existing transactional contexts (% of respondents)

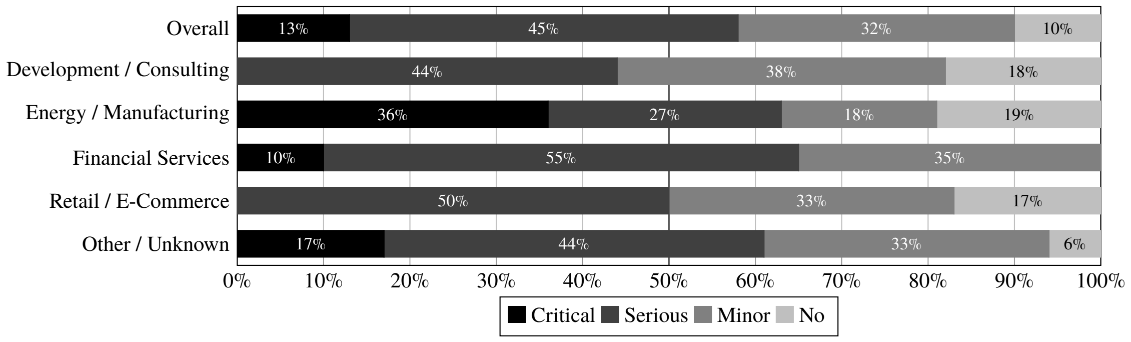

Figure 5: Relevance of the unavailability of cross-service ACID transactionality (% of respondents)

5.8 Answers to the Research Questions

To conclude the findings of the survey, we present the answers to the research questions introduced in Sect. 4.1 below.

RQ1: The primary drivers for companies to adopt microservices were found to be scalability, maintainability, and time to market. The ability to serve as an enabler for Continuous Delivery and DevOps, the suitedness for virtualization, and organizational improvements were rated as secondary drivers.

RQ2: Insufficient skill sets of both developers and operations as well as resistance of the operations staff are perceived as the most important barriers to microservice adoption. Depending on the industry, backup and compliance issues are considered important as well. Furthermore, the deployment scenario of the application may prevent the adoption of microservices. In particular, microservices are considered not well-suited for third-party software running as an on-premise installation.

RQ3: The primary goal for software modernization using microservices was found to be improved maintainability, followed by shorter time to market and better scalability. Modernization appears to be a relevant scenario for microservice adoption, as two thirds of the respondents stated that there are plans or projects to introduce microservices to existing applications.

RQ4: While the potential performance impact in modernization settings was not considered very relevant by the respondents, the impact on transactionality was found to be an important issue. The majority of the respondents rated the loss of cross-service ACID transactionality as a serious limitation, and more than three fourths consider it important to incorporate the transactional boundaries into the service design.

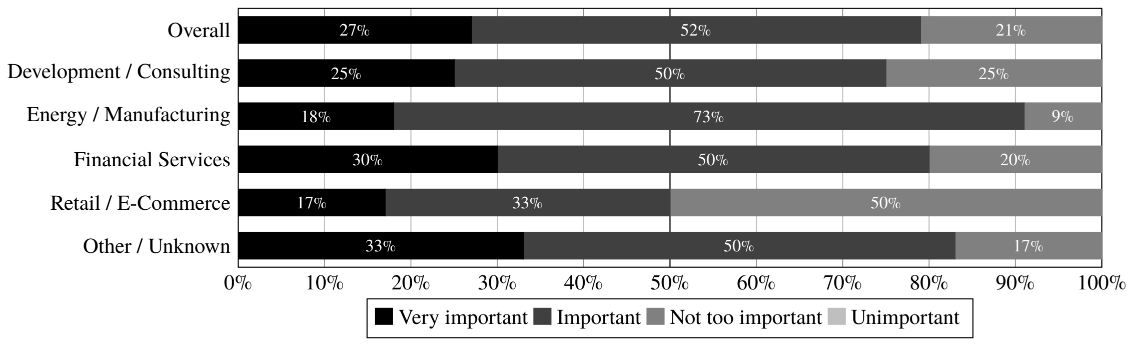

Figure 6: Importance of transaction boundaries for service design (% of respondents)

6 Discussion

6.1 Threats to Validity

In order to discuss the threats to the validity of this survey, we rely on the scheme presented by Runeson and Höst (2008). We see the greatest threat to *construct validity* in misinterpretation of the survey questions, in particular, different understandings of the items that the participants were to rate. This threat was mitigated by careful questionnaire design, internal discussion, and testing. Furthermore, the questionnaire contained several terms that may be considered as “buzz words” (including the term “microservices” itself), which may be particularly suspectible to misinterpretation. However, we decided to use these terms as we felt that these terms were what the participants were most familiar with and using different terms might have caused additional confusion.

As for *external validity*, we see the most important threat in the size and structure of the sample. Since the sample only contained practicioners from Germany, the generalizability of the results to other countries may be limited. As the respondents were acquired by a combination of convenience and snowball sampling, the sample was not representative. We think, however, that due to the very specific target population, this sampling method was the best choice with respect to accuracy and response rate. A comparison with the sample structure of a large, but also non-representative survey[22] by developer portal Stack Overflow indicates that in particular the financial services industry may be overrepresented in our sample. Furthermore, different degrees of company heterogeneity in the different industry sectors must be expected, but cannot be quantified due to the anonymity of the study. It is therefore possible that some results are biased by a large number of participants from the same company.

A further threat may result from the size of the sample. Although we are very content with the overall number of respondents, some groups were too small to get statistically significant results. In particular, the “Retail / E-Commerce” group, which was only established due to the pioneering role of this industry for microservices, had very few participants, and the results may therefore be skewed.

As we perform multiple statistical tests on the same data set, the results may furthermore be subject to *alpha error accumulation*, i. e., the probability of incorrectly rejecting the null hypothesis in one of these tests is actually higher than the individual alpha error rate for each test. This effect is particularly important when conclusions are drawn from multiple tests at the same time (e. g., a drug is considered efficacious when it leads to a significant improvement of one of several diseases). However, as no conclusions of this

[22] See https://www.stackoverflowbusiness.com/de/talent/ressourcen/die-stack-overflow-entwicklerumfrage-2017 (in German)

sort are drawn in this study, the effect of falsely significant findings is limited to that particular result.

Another threat results from the decision to primarily target practicioners already concerned with microservices. While their existing knowledge on microservices may reduce the risk of misinterpretation and increase the soundness of their answers, it may also lead to a positive bias towards this architectural style. We therefore also included data from seven respondents who reported not to have significant knowledge of microservices. However, as microservices are currently a "hype" topic, a certain bias in favor of microservices could be expected.

A threat to *reliability* may result from the fact that we acquired a significant number of respondents by speaking to them personally. Although we took great care not to suggest anything regarding the questions to the respondents, this risk cannot be completely ruled out.

6.2 Comparison with Results from other Studies

As discussed in Sect. 3, there are several other studies whose results complement the findings of this study. In the following paragraphs, we discuss the similarities and differences between the findings of those studies and ours.

Similar to our study, Taibi et al. (2017) find maintainability and scalability to be the primary drivers for microservice adoption. According to their results, an improvement in maintainability is also the most important benefit that is actually achieved. Short time to market, the third primary driver identified by our study, is not investigated. Suitedness for DevOps as well as improved team organization are identified as secondary drivers by their study as well. As for barriers, moving functionality from the monolith to microservices and splitting the database are found to be most important. People-related barriers such as resitances or lack of skills, which received the highest score in our study, were only marginally investigated.

Ghofrani and Lübke (2018) identify scalability and agility as the primary drivers, while the distributed nature of microservices is rated as the primary challenge. Response time and performance, together with security, raised the greatest concerns with respect to non-functional properties. This is a notable difference to our results, as our respondents did not expect performance to be an issue when introducing microservices, and also differs from other reports from practice such as (Gouigoux and Tamzalit 2017). The aspect of modernization using microservices is not investigated in this survey.

Di Francesco et al. (2018) report on several challenges with pre-existing software systems that can be considered as drivers for the modernization. While long time to market is found to be the top challenge, three of the top five challenges refer directly or indirectly to a lack of maintainability (high coupling, unexpected side effects of changes). As for microservice implementation, setting up the required infrastructure, the change to the developer's mindset and distributed monitoring are found to be the top three challenges. This maps reasonably well to our findings that insufficient developer skills, running distributed applications as well as monitoring are significant barriers to microservice adoption. Barriers due to the operations staff, which were rated highest by our participants, are not named in this study.

The industrial survey by NGINX (2016) reports that roughly one third of their respondents are using microservices in production, one third are investigating microservices, and one third is not using them at all. This matches quite well with our numbers on microservice adoption, but due to the different sampling method and target population, these similarities must be considered with care.

A study by Lightbend, Inc. (2016) reports similar numbers on plans for microservice usage. It also reports on drivers for microservice adoption and names development agility and velocity as well as elasticity as the most important ones, and states that the use of microservices for modernization is particularly important for large companies. Cultural change is named as one of the major barriers to microservice adoption, especially for large companies. Other barriers are not named.

The importance of cultural change, which may result in resistance, is also highlighted by empirical research on DevOps (Riungu-Kalliosaari et al. 2016; Smeds et al. 2015) as well as experience reports on the adoption of Continuous Delivery (Neely and Stolt 2013).

When comparing our study with earlier studies on the adoption of service-oriented architectures (SOA), it becomes apparent that the motivations for SOA adoption were quite different from the motivations to employ microservices. A study by MacLennan and Van Belle (2014) reports important drivers to be improved organizational agility, reuse, standardized data representation, legacy system integration, and improved business processes. While improved agility is closely related to notions like short time to market (Becker et al. 2009), legacy integration and business processes are scarcely discussed in the context of microservices. Similarly, several of the challenges in the study refer to (centralized) governance, which is avoided as much as possible in microservice settings. The strong focus on (business) integration is also supported by the findings of Joachim (2011).

6.3 Implications of the Results

The results of our study indicate several implications for companies considering the adoption of microservices. Based on our results on barriers and implementation challenges presented in Sect. 5.3 and Sect. 5.4, companies should pay particular attention to the cultural and people-related changes implied by microservices. Potential resistance by both developers and operations staff should be investigated and addressed early. And care must be taken that the development teams are actually capable of performing the new tasks assigned to them, which may now include aspects of operation such as ensuring safety and security.

Furthermore, we observe that the issue of compliance and regulations may be underrated. Although this item did not receive a particularly high score (1.21), the high standard deviation (1.11) indicates that there are different opinions on this topic.

As for modes of operation, the results from Sect. 5.5 imply that microservices are not deemed very viable for traditionally licensed software, i. e. software that is developed by a third party and run by the customer. Vendors of such software might therefore want to contemplate offering it as a service (SaaS) when moving to microservices instead of providing it for operation by the customer.

The results from Sect. 5.7 further indicate that consistency and transactionality should be carefully considered during service design. Especially when polyglot persistency is to be employed, it must be decided whether and where ACID transactions are required, and where eventual consistency is sufficient. For the latter case, it is furthermore necessary to choose and implement strategies for dealing with inconsistencies (see Sect. 2.4).

7 Conclusions and Future Work

In this paper, we have identified important drivers and barriers to microservice adoption in the German software industry, and highlighted some notable differences among industry sectors. The premier drivers were found to be scalability, maintainability, and time to market, while the skill set of both development and operations staff was identified as the main barrier. Although several implementation challenges were considered difficult by the respondents, no “show stoppers” were identified.

Concerning drivers, we observe interesting differences between early adopters who emphasize *scalability* of their “Internet-scale” systems, compared to traditional companies who emphasize *maintainability* of their IT systems. In particular, for the adoption of microservices as a means for software modernization, maintainability appears to be the leading driver. Performance degradation due to remote invocations is considered a minor issue, while the lack of cross-service transactionality appears to be a serious concern.

As future work, a replication of a similar study with the software industry in other countries, and comparison with the German situation would be valuable. The study material is available on Zenodo,[23] so that other researchers can replicate and extend our work. A technical report in German on the results of this study is also available.[24]

Further opportunities for future work include a more detailed investigation of the drivers and barriers identified in this study. For instance, further insight on the precise reasons for resistances would be valuable in order to be able to address them properly. Another interesting topic would be to investigate whether the use of modern approaches like microservices may indeed be a way to increase the attractiveness as an employer.

References

Alshuqayran N., Ali N., Evans R. (2016) A Systematic Mapping Study in Microservice Architecture. In: Proceedings of the 9th International Conference on Service-Oriented Computing and Applications (SOCA). IEEE, pp. 44–51

Balalaie A., Heydarnoori A., Jamshidi P. (2016) Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. In: IEEE Software 33(3), pp. 42–52

Bass L., Weber I., Zhu L. (2015) DevOps: A Software Architect's Perspective. Addison-Wesley, New York

Becker A., Buxmann P., Widjaja T. (2009) Value Potential and Challenges of Service-Oriented Architectures – A User and Vendor Perspective. In: Proceedings of the 17th European Conference on Information Systems (ECIS). 88. AIS eLibrary https://aisel.aisnet.org/ecis2009/88

Carrasco A., van Bladel B., Demeyer S. (2018) Migrating Towards Microservices: Migration and Architecture Smells. In: Proceedings of the 2nd International Workshop on Refactoring (IWoR). ACM, New York, pp. 1–6

Chen L. (2015) Continuous Delivery: Huge Benefits, but Challenges Too. In: IEEE Software 32(2), pp. 50–54

[23] https://doi.org/10.5281/zenodo.820146
[24] see http://eprints.uni-kiel.de/38682/

Chen L. (2018) Microservices: Architecting for Continuous Delivery and DevOps. In: Proceedings of the 2018 IEEE International Conference on Software Architecture (ICSA). IEEE, pp. 39–397

Di Francesco P., Lago P., Malavolta I. (2017) Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption. In: Proceedings of the 2017 IEEE International Conference on Software Architecture (ICSA). IEEE, pp. 21–30

Di Francesco P., Lago P., Malavolta I. (2018) Migrating towards Microservice Architectures: an Industrial Survey. In: Proceedings of the 2018 IEEE International Conference on Software Architecture (ICSA). IEEE, pp. 29–2909

Esposito C., Castiglione A., Choo K. R. (2016) Challenges in Delivering Software in the Cloud as Microservices. In: IEEE Cloud Computing 3(5), pp. 10–14

Evans E. (2007) Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, Upper Saddle River

Fowler M. (2014) Microservice Prerequisites http://martinfowler.com/bliki/MicroservicePrerequi↩ sites.html Last Access: 2018-11-30

Ghofrani J., Lübke D. (2018) Challenges of Microservices Architecture: A Survey on the State of the Practice. In: Proceedings of the 10th Central European Workshop on Services and their Composition (ZEUS). CEUR Workshop Proceedings, Aachen, pp. 1–8

Gmeiner J., Ramler R., Haslinger J. (2015) Automated Testing in the Continuous Delivery Pipeline: A Case Study of an Online Company. In: Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, pp. 1–6

Gouigoux J. P., Tamzalit D. (2017) From Monolith to Microservices: Lessons Learned on an Industrial Migration to a Web Oriented Architecture. In: Proceedings of the 2017 IEEE International Conference on Software Architecture Workshops (ICSAW). IEEE, pp. 62–65

Hasselbring W. (2016) Microservices for Scalability: Keynote Talk Abstract. In: Proceedings of the 7th ACM/SPEC International Conference on Performance Engineering (ICPE). ACM, pp. 133–134

Hasselbring W., Steinacker G. (2017) Microservice Architectures for Scalability, Agility and Reliability in E-Commerce. In: Proceedings of the 2017 IEEE International Conference on Software Architecture Workshops (ICSAW). IEEE, pp. 243–246

Humble J., Farley D. (2011) Continuous Delivery. Addison-Wesley, Upper Saddle River

Hüttermann M. (2012) DevOps for Developers. Apress, New York

Joachim N. (2011) A Literature Review of Research on Service-Oriented Architectures (SOA): Characteristics, Adoption Determinants, Governance Mechanisms, and Business Impact. In: Proceedings of the 7th Americas Conference on Information Systems (AMCIS). 339. AIS eLibrary https://aisel.aisnet.org/amcis2011_submissions/339

Killalea T. (2016) The Hidden Dividends of Microservices. In: Communications of the ACM 59(8), pp. 42–45

Knapp T. R. (1990) Treating Ordinal Scales as Interval Scales: An Attempt to Resolve the Controversy. In: Nursing Research 39(2), pp. 121–123

Knoche H. (2016) Combining Application-Level and Database-Level Monitoring to Analyze the Performance Impact of Database Lock Contention. In: Softwaretechnik-Trends 36(4), pp. 25–27

Knoche H., Hasselbring W. (2018) Using Microservices for Legacy Software Modernization. In: IEEE Software 35(3), pp. 44–49

Kratzke N. (2015) About Microservices, Containers and their Underestimated Impact on Network Performance. In: Proceedings of the 6th International Conference on Cloud Computing, GRIDs and Virtualization. IARIA, pp. 165–169

Leppänen M., Mäkinen S., Pagels M., Eloranta V. P., Itkonen J., Mäntylä M. V., Männistö T. (2015) The Highways and Country Roads to Continuous Deployment. In: IEEE Software 32(2), pp. 64–72

Lewis J., Fowler M. (2014) Microservices http://martinfowler.com/articles/microservices.html Last Access: 2018-11-30

Lightbend, Inc. (2016) Enterprise Development Trends 2016 https://info.lightbend.com/COLL-20XX-Enterprise-Development-Trends-2016-Report_RES-LP.html Last Access: 2018-11-30

Litoiu M. (2004) Migrating to Web Services: A Performance Engineering Approach. In: Journal of Software Maintenance and Evolution: Research and Practice 16(1–2), pp. 51–70

MacLennan E., Van Belle J.-P. (2014) Factors Affecting the Organizational Adoption of Service-Oriented Architecture (SOA). In: Information Systems and e-Business Management 12(1), pp. 71–100

Nagappan N., Murphy B., Basili V. (2008) The Influence of Organizational Structure on Software Quality. In: Proceedings of the 30th ACM/IEEE International Conference on Software Engineering (ICSE). IEEE, pp. 521–530

Neely S., Stolt S. (2013) Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy). In: Proceedings of the 2013 Agile Conference. IEEE, pp. 121–128

Newman S. (2015) Building Microservices. O'Reilly, Sebastopol

NGINX (2016) The Future of Application Development and Delivery Is Now https://www.nginx.com/resources/library/app-dev-survey/ Last Access: 2018-11-30

Nygard M. T. (2007) Release It! – Design and Deploy Production-Ready Software. The Pragmatic Bookshelf, Raleigh

Olsson H. H., Alahyari H., Bosch J. (2012) Climbing the "Stairway to Heaven" – A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software. In: Proceedings of the 38th Euromicro Conference on Software Engineering and Advanced Applications. IEEE, pp. 392–399

Pahl C., Jamshidi P. (2016) Microservices: A Systematic Mapping Study. In: Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER). ACM, pp. 137–146

Pardon G., Pautasso C. (2014) Atomic Distributed Transactions: A RESTful Design. In: Proceedings of the 23rd International Conference on World Wide Web. ACM, pp. 943–948

Razavian M., Lago P. (2010) Understanding SOA Migration Using a Conceptual Framework. In: Journal of Systems Integration 1(3), pp. 33–44

Riungu-Kalliosaari L., Mäkinen S., Lwakatare L. E., Tiihonen J., Männistö T. (2016) DevOps Adoption Benefits and Challenges in Practice: A Case Study. In: Proceedings of the International Conference on Product-Focused Software Process Improvement (PROFES). Springer, Berlin, pp. 590–597

Runeson P., Höst M. (2008) Guidelines for Conducting and Reporting Case Study Research in Software Engineering. In: Empirical Software Engineering 14(2), p. 131

Schermann G., Cito J., Leitner P. (2016) All the Services Large and Micro: Revisiting Industrial Practice in Services Computing. In: Proceedings of the International Conference on Service-Oriented Computing (ICSOC) Workshops. Springer, Berlin, pp. 36–47

Singleton A. (2016) The Economics of Microservices. In: IEEE Cloud Computing 3(5), pp. 16–20

Smeds J., Nybom K., Porres I. (2015) DevOps: A Definition and Perceived Adoption Impediments. In: Proceedings of the International Conference on Agile Processes in Software Engineering and Extreme Programming (XP). Springer, Berlin, pp. 166–177

Stevens S. S. (1946) On the Theory of Scales and Measurement. In: Science 103(2684), pp. 677–680

Stine M. (2015) Migrating to Cloud-Native Application Architectures. O'Reilly, Beijing

Taibi D., Lenarduzzi V., Pahl C. (2017) Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation. In: IEEE Cloud Computing 4(5), pp. 22–32

Vogels W. (2009) Eventually Consistent. In: Communications of the ACM 52(1), pp. 40–44

Wolff E. (2016) Microservices – Flexible Software Architectures. Addison-Wesley, Boston

A Survey Questions

A translation of the questions used in the survey with their answer options are listed below. Questions whose results are not used in this paper are marked with an asterisk (∗).

Question 1
Since when are you concerned with microservices?

- Since ☐ months / ☐ years
- Not yet

Question 2
Is your company or customer already employing microservices?

- Yes, to a large extent
- Yes, to a lesser extent
- No

Question 3
How relevant do you think the following properties commonly attributed to microservices are for the decision to employ microservices? (Answer options for each item: *irrelevant, hardly relevant, relevant, crucial*)

1. Easy and specific scalability / elasticity
2. Short "time to market" (i. e. time from development to productive deployment)
3. High maintainability
4. Service-specific choice of programming language ("Polyglot Programming")
5. Service-specific choice of database / persistence solution ("Polyglot Persistence")
6. Enabler for Continuous Delivery / DevOps
7. Suitedness for Cloud deployment and container-based virtualization
8. Improvement of organizational structures
9. Improvement of attractiveness as an employer

Question 4
How relevant do you think the following barriers are for the decision to employ microservices? (Answer options for each item: *irrelevant, hardly relevant, relevant, crucial*)

1. Resistance by development teams
2. Insufficient developer skills
3. Resistance by operations teams
4. Insufficient operations skills
5. Increased effort for supporting and licensing databases, etc
6. Increased deployment complexity
7. Incompatibility with compliance and regulations
8. Insufficient maturity of technologies
9. Difficulty of consistent backups due to distributed persistence
10. Incompatibilities with existing software

Question 5
How difficult do you think it is to implement the following aspects related to microservices? (Answer options for each item: *simple, medium, difficult, impossible*)

1. Creation of an automated deployment pipeline
2. Ad-hoc provisioning of resources (e. g., for automated integration tests)
3. Decentalized / distributed persistence
4. Have ops tasks performed by development teams
5. Change of tasks and responsibilites for ops teams
6. Running heavily distributed applications in production
7. Automation of integration tests and further test stages
8. Formal description of infrastructure ("Infrastructure as Code")
9. Support for several programming languages ("Polyglot Programming")
10. Establishment of sufficient monitoring

Question 6 (∗)
Microservices are commonly related to web applications, but are not limited to them. How well

do you think microservices are suited for the following types of applications? (Answer options for each item: *not at all, moderately, well, perfectly*)

1. Public web applications
2. Internal web applications
3. Client-server applications
4. Legacy applications
5. Batch applications

Question 7
Most well-known microservice applications are run in the Cloud. However, many companies run self-developed or licensed software on their own hardware. How well do you think microservices are suited for such situations (e. g., with respect to elasticity and frequent deployments)? (Answer options for each item: *not at all, moderately, well, perfectly*)

1. Running self-developed software on own hardware
2. Running licensed software on own hardware

Question 8
Are there already plans or projects in your company to introduce microservices to existing applications?

- Yes
- No

Question 9
Which goals would you pursue by introducing microservices to an existing application? (Answer options for each item: *primarily, secondarily, not at all*)

1. Improve scalability
2. Improve "time to market"
3. Improve maintainability
4. Improve quality
5. Gain access to new technologies
6. Pave the way for Continuous Delivery / DevOps
7. Improve employee motivation

Question 10
(If you chose "Gain access to new technologies" above): To which technologies would you like to gain access to? (Open question)

Question 11
Would you only add new functionality or would you also replace existing functionality by microservices?

- No, I would only add new functionality
- Yes, I would also replace existing functionality

Question 12 (∗)
(If you would also replace existing functionality): Do you expect to be able to re-use parts of the existing implementation?

- Yes, to a large extent
- Yes, to a lesser extent
- No, but I would like to
- No, I do not want to even if I could

Question 13
Microservices lead to an increase in inter-process or network communication. Do you expect this to cause a decrease in runtime performance?

- Yes, critical
- Yes, serious
- Yes, but minor
- No

Question 14
The distributed persistence of microservices commonly requires to eschew ACID transactionality across services. How do you rate this eschewal?

- Critical
- Relevant
- Hardly relevant
- Irrelevant

Question 15
Introducing service invocations to existing transaction contexts may elongate their runtime, thus reducing throughput. Do you think this is a problem?

- Yes, a serious problem
- Yes, but only in specific cases
- No

Question 16
How important do you think it is to incorporate transactional boundaries into the service design?

- Very important
- Important
- Not too important
- Unimportant

Question 17
In what line are you / is your company in (e. g., Banking, Consulting)? (Open question)

Question 18
What position(s) do you hold?

- Management
- Software Architect
- Software Developer
- Consultant
- Other: (Room for answer)

Question 19
In what department(s) / area(s) do you work?

- Development
- Operations
- Specialist Department
- Other: (Room for answer)

B Details on Statistical Tests

Details on the individual statistical significance tests in the paper are provided below. For each test, the test type, the alternative hypothesis, the relevant data and the p-value of the test are provided. For specifying the alternative hypothesis, we use the following notation:

- ls(p_1, p_2) refers to the location shift between the distributions of the subsamples defined by the selection predicates p_1 and p_2
- median(p) refers to the median of the score of the subsample defined by the selection predicate p

Selection predicates for responses are specified as follows:

- Groups (e. g., "finance" for "Financial Services"), are written with non-capital letters. If only a group is given (only valid for yes/no questions), it refers to the positive responses
- Complements of groups are written with an overline (e. g., "$\overline{\text{finance}}$" for all responses not from the financial services domain)
- Aspects (e. g., "Time to Market") are written with capitalized first letters. If no group is given, it refers to all responses for this aspect
- Aspects in a group are written as "Aspect@Group", e. g., "Time to Market@finance"

For brevity, we use the following abbreviations for groups and aspects:

Abbreviation	**Full Name**
devcons	Development / Consulting
enermfg	Energy / Manufacturing
finance	Financial Services
heavyuser	"heavy users"
retail	Retailing / E-Commerce

Abbreviation	**Full Name**
Attr. Employ.	Attractivenes as Employer
Backups	Consistent Backups
CD / DevOps	Enabler for CD and DevOps
Compliance	Compliance and Regulations
Docker/Cloud	Suitedness for Cloud and Docker
Maintainability	High Maintainability
New Technology	Introduce New Technology
Ops Skills	Insufficient Ops Skills
Poly. Persist.	Polyglot Persistence
Poly. Program.	Polyglot Programming
Provisioning	Resource Provisioning
Quality	Improve Quality
Scalability	High Scalability and Elasticity
Time to Market	Short Time to Market

B.1 Tests on Question 2

Test T_1

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	$\text{ls}(\text{retail}, \overline{\text{retail}}) > 0$
Data	$n_1 = 6$; $n_2 = 65$ $m_1 = 1$; $m_2 = 0$ $W = 328.5000$
p-Value	0.001656

Test T_2

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	$\text{ls}(\text{finance}, \overline{\text{finance}}) < 0$
Data	$n_1 = 20$; $n_2 = 51$ $m_1 = -1$; $m_2 = 0$ $W = 339$
p-Value	0.009998

B.2 Tests on Question 3

Test T_3

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Time to Market, CD / DevOps) > 0
Data	$n_1 = 71$; $n_2 = 71$ $m_1 = 2$; $m_2 = 2$ $W = 3261$
p-Value	0.000597

Test T_4

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Attr. Employ., Poly. Program.) < 0
Data	$n_1 = 71$; $n_2 = 71$ $m_1 = 1$; $m_2 = 1$ $W = 1898$
p-Value	0.003724

Test T_5

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	$\text{ls}(\text{Docker/Cloud@devcons}, \text{Docker/Cloud@}\overline{\text{devcons}}) > 0$
Data	$n_1 = 16$; $n_2 = 55$ $m_1 = 2$; $m_2 = 1$ $W = 592$
p-Value	0.013572

Test T_6

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Poly. Program.@enermfg, Poly. Program.@$\overline{\text{enermfg}}$) > 0
Data	$n_1 = 11$; $n_2 = 60$
	$m_1 = 2$; $m_2 = 1$
	$W = 463.5000$
p-Value	0.012416

Test T_7

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Poly. Program.@retail, Poly. Program.@$\overline{\text{retail}}$) > 0
Data	$n_1 = 6$; $n_2 = 65$
	$m_1 = 2$; $m_2 = 1$
	$W = 283$
p-Value	0.027161

Test T_8

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Attr. Employ.@retail, Attr. Employ.@$\overline{\text{retail}}$) > 0
Data	$n_1 = 6$; $n_2 = 65$
	$m_1 = 2$; $m_2 = 1$
	$W = 290.5000$
p-Value	0.017768

Test T_9

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Time to Market@enermfg, Time to Market@$\overline{\text{enermfg}}$) < 0
Data	$n_1 = 11$; $n_2 = 60$
	$m_1 = 2$; $m_2 = 2$
	$W = 200$
p-Value	0.012007

Test T_{10}

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Docker/Cloud@enermfg, Docker/Cloud@$\overline{\text{enermfg}}$) < 0
Data	$n_1 = 11$; $n_2 = 60$
	$m_1 = 1$; $m_2 = 2$
	$W = 211$
p-Value	0.022894

Test T_{11}

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Poly. Program.@enermfg, Poly. Persist.@enermfg)
Data	$n_1 = 11$; $n_2 = 11$ $m_1 = 2$; $m_2 = 1$ $W = 87.5000$
p-Value	0.029442

Test T_{12}

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Maintainability@devcons, Scalability@devcons) > 0
Data	$n_1 = 16$; $n_2 = 16$ $m_1 = 2$; $m_2 = 2$ $W = 134$
p-Value	0.401466

Test T_{13}

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Attr. Employ.@heavyuser, Attr. Employ.@$\overline{\text{heavyuser}}$) > 0
Data	$n_1 = 19$; $n_2 = 52$ $m_1 = 1$; $m_2 = 1$ $W = 515.5000$
p-Value	0.383101

Test T_{14}

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Poly. Persist.@finance, Poly. Persist.@$\overline{\text{finance}}$) > 0
Data	$n_1 = 20$; $n_2 = 51$ $m_1 = 1$; $m_2 = 1$ $W = 574$
p-Value	0.191551

B.3 Tests on Question 4

Test T_{15}

	One-sample Test
Test Type	Sign test
Alternative	median(Ops Skills) > 1
Data	$n_p = 43$; $n = 70$
p-Value	0.036119

Test T_{16}

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Backups@finance, Backups@$\overline{\text{finance}}$) > 0
Data	$n_1 = 20$; $n_2 = 51$ $m_1 = 2$; $m_2 = 1$ $W = 716$
p-Value	0.002539

Test T_{17}

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Compliance@enermfg, Compliance@$\overline{\text{enermfg}}$) > 0
Data	$n_1 = 11$; $n_2 = 60$ $m_1 = 0$; $m_2 = 1$ $W = 175.5000$
p-Value	0.005322

Test T_{18}

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Compliance@devcons, Compliance@$\overline{\text{devcons}}$) > 0
Data	$n_1 = 16$; $n_2 = 55$ $m_1 = 1$; $m_2 = 1$ $W = 483.5000$
p-Value	0.266709

Test T_{19}

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Compliance@finance, Compliance@$\overline{\text{finance}}$) > 0
Data	$n_1 = 20$; $n_2 = 51$ $m_1 = 1.5$; $m_2 = 1$ $W = 627.5000$
p-Value	0.059079

B.4 Tests on Question 5

Test T_{20}

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Provisioning@finance, Provisioning@$\overline{\text{finance}}$) > 0
Data	$n_1 = 20$; $n_2 = 51$ $m_1 = 2$; $m_2 = 1$ $W = 639$
p-Value	0.037406

Test T_{21}

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Provisioning@finance, Provisioning@devcons) > 0
Data	$n_1 = 20$; $n_2 = 16$
	$m_1 = 2$; $m_2 = 1$
	$W = 240.5000$
p-Value	0.002504

Test T_{22}

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Poly. Program.@finance, Poly. Program.@$\overline{\text{finance}}$)
Data	$n_1 = 20$; $n_2 = 51$
	$m_1 = 2$; $m_2 = 0$
	$W = 783.5000$
p-Value	0.000010

B.5 Tests on Question 9

Test T_{23}

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Maintainability, Time to Market) < 0
Data	$n_1 = 71$; $n_2 = 71$
	$m_1 = 2$; $m_2 = 2$
	$W = 3057.5000$
p-Value	0.002817

Test T_{24}

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(New Technology, CD / DevOps) < 0
Data	$n_1 = 71$; $n_2 = 71$
	$m_1 = 1$; $m_2 = 2$
	$W = 1873$
p-Value	0.002166

Test T_{25}

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	ls(Quality@leads, Quality@$\overline{\text{leads}}$) > 0
Data	$n_1 = 11$; $n_2 = 60$
	$m_1 = 2$; $m_2 = 1$
	$W = 436$
p-Value	0.030351

Test T_{26}

	Two-sample Test
Test Type	Wilcoxon rank sum test
Alternative	$\text{ls}(\text{CD / DevOps@leads}, \text{CD / DevOps@}\overline{\text{leads}}) > 0$
Data	$n_1 = 11$; $n_2 = 60$ $m_1 = 2$; $m_2 = 1$ $W = 428$
p-Value	0.042106

C Cross-Correlation Tables

The cross-correlaton tables of Questions 3 to 5 and 9 are shown below.

	High Scalability and Elasticity	Short Time to Market	High Maintainability	Polyglot Programming	Polyglot Persistence	Enabler for CD and DevOps	Suitedness for Cloud and Docker	Organizational Improvement	Attractiveness as Employer
High Scalability and Elasticity	—	-0,13	0,04	0,21	0,13	-0,06	0,11	-0,09	0,02
Short Time to Market	-0,13	—	0,01	-0,17	0,05	0,20	-0,13	0,07	0,01
High Maintainability	0,04	0,01	—	0,07	0,11	0,18	0,04	0,33	0,10
Polyglot Programming	0,21	-0,17	0,07	—	0,47	-0,06	-0,05	0,05	0,02
Polyglot Persistence	0,13	0,05	0,11	0,47	—	-0,08	-0,04	0,05	0,11
Enabler for CD and DevOps	-0,06	0,20	0,18	-0,06	-0,08	—	0,38	0,03	0,07
Suitedness for Cloud and Docker	0,11	-0,13	0,04	-0,05	-0,04	0,38	—	0,08	0,15
Organizational Improvement	-0,09	0,07	0,33	0,05	0,05	0,03	0,08	—	0,28
Attractiveness as Employer	0,02	0,01	0,10	0,02	0,11	0,07	0,15	0,28	—

Table 10: Cross-correlation table for Question 3

	Resistance by Devs	Insufficient Dev Skills	Resistance by Ops	Insufficient Ops Skills	Support Contracts / Licenses	Deployment Complexity	Compliance and Regulations	Maturity of Technology	Consistent Backups	Compatibility Issues
Resistance by Devs	—	0,42	0,16	-0,04	-0,01	-0,20	0,11	0,02	-0,04	0,03
Insufficient Dev Skills	0,42	—	0,11	0,23	-0,08	-0,03	0,20	0,13	0,07	-0,09
Resistance by Ops	0,16	0,11	—	0,40	0,13	-0,08	0,24	0,29	-0,17	-0,01
Insufficient Ops Skills	-0,04	0,23	0,40	—	0,05	0,31	0,32	0,21	-0,07	-0,07
Support Contracts / Licenses	-0,01	-0,08	0,13	0,05	—	0,30	0,16	0,13	0,28	0,07
Deployment Complexity	-0,20	-0,03	-0,08	0,31	0,30	—	0,31	0,12	0,18	0,09
Compliance and Regulations	0,11	0,20	0,24	0,32	0,16	0,31	—	0,21	0,14	0,08
Maturity of Technology	0,02	0,13	0,29	0,21	0,13	0,12	0,21	—	0,15	0,15
Consistent Backups	-0,04	0,07	-0,17	-0,07	0,28	0,18	0,14	0,15	—	0,05
Compatibility Issues	0,03	-0,09	-0,01	-0,07	0,07	0,09	0,08	0,15	0,05	—

Table 11: Cross-correlation table for Question 4

	Building Pipelines	Resource Provisioning	Decentralized Persistence	Ops Tasks by Dev Teams	Change of Ops Tasks	Running Distributed Apps	Test Automation	Infrastructure as Code	Polyglot Programming	Establishing Monitoring
Building Pipelines	—	0,39	0,12	-0,01	-0,29	-0,02	0,49	0,36	0,13	0,15
Resource Provisioning	0,39	—	0,08	0,01	-0,07	-0,04	0,26	0,27	0,07	0,17
Decentralized Persistence	0,12	0,08	—	0,24	-0,09	0,09	0,17	0,06	0,03	0,21
Ops Tasks by Dev Teams	-0,01	0,01	0,24	—	0,32	0,00	-0,02	0,18	0,09	0,14
Change of Ops Tasks	-0,29	-0,07	-0,09	0,32	—	0,08	-0,10	0,08	-0,04	0,07
Running Distributed Apps	-0,02	-0,04	0,09	0,00	0,08	—	0,04	-0,07	-0,10	0,19
Test Automation	0,49	0,26	0,17	-0,02	-0,10	0,04	—	0,09	-0,01	0,22
Infrastructure as Code	0,36	0,27	0,06	0,18	0,08	-0,07	0,09	—	0,11	0,15
Polyglot Programming	0,13	0,07	0,03	0,09	-0,04	-0,10	-0,01	0,11	—	0,00
Establishing Monitoring	0,15	0,17	0,21	0,14	0,07	0,19	0,22	0,15	0,00	—

Table 12: Cross-correlation table for Question 5

	Resistance by Devs	Insufficient Dev Skills	Resistance by Ops	Insufficient Ops Skills	Support Contracts / Licenses	Deployment Complexity	Compliance and Regulations	Maturity of Technology	Consistent Backups	Compatibility Issues
Building Pipelines	-0,02	0,07	0,11	-0,16	-0,08	-0,03	-0,07	-0,08	0,02	-0,04
Resource Provisioning	-0,01	0,10	-0,11	-0,14	0,13	0,07	-0,06	-0,09	0,01	0,01
Decentralized Persistence	-0,18	-0,09	-0,13	-0,22	-0,09	-0,09	-0,21	0,18	0,21	0,01
Ops Tasks by Dev Teams	-0,22	-0,05	0,10	0,27	0,29	0,26	0,05	0,20	0,31	0,31
Change of Ops Tasks	-0,01	0,14	0,23	0,45	0,12	0,00	0,21	0,20	0,04	0,24
Running Distributed Apps	-0,04	-0,06	0,29	0,17	0,02	0,12	0,18	-0,07	-0,02	-0,09
Test Automation	0,05	0,13	0,16	0,01	-0,25	0,11	0,21	0,07	-0,07	0,05
Infrastructure as Code	-0,03	0,12	-0,07	-0,02	0,23	0,28	0,03	0,08	0,08	0,02
Polyglot Programming	0,12	0,02	-0,09	-0,14	0,26	-0,01	-0,02	0,02	0,49	0,14
Establishing Monitoring	-0,05	0,14	-0,04	0,02	0,10	0,31	0,21	-0,05	0,19	0,05

Table 13: Cross-correlation table for Questions 4 and 5

	Improve Scalability	Improve Time to Market	Improve Maintainability	Improve Quality	Introduce New Technology	Prepare CD and DevOps	Improve Team Motivation
Improve Scalability	—	-0,05	0,07	-0,22	-0,01	0,14	0,12
Improve Time to Market	-0,05	—	-0,18	-0,10	0,20	0,03	0,33
Improve Maintainability	0,07	-0,18	—	0,42	-0,02	0,28	0,03
Improve Quality	-0,22	-0,10	0,42	—	0,21	0,28	0,05
Introduce New Technology	-0,01	0,20	-0,02	0,21	—	0,14	0,34
Prepare CD and DevOps	0,14	0,03	0,28	0,28	0,14	—	0,16
Improve Team Motivation	0,12	0,33	0,03	0,05	0,34	0,16	—

Table 14: Cross-correlation table for Question 9