DAVIDE TAIBI

VALENTINA LENARDUZZI

CLAUS PAHL

Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation

Davide Taibi, Valentina Lenarduzzi, and Claus Pahl, Free University of Bozen-Bolzano

 2325-6095/17/$33.00 © 2017 IEEE

Microservices have been getting more and more popular in recent years, and several companies are migrating monolithic applications to microservices. Microservices allow developers to independently develop and deploy services, and ease the adoption of agile processes. However, many companies are still hesitant to migrate because they consider microservice as a hype or because they are not aware of the migration process and the benefits and issues related to migration. For this purpose, we conducted a survey among experienced practitioners who already migrated their monoliths to microservices. In this paper, we identify a process framework based on the comparison of three different migration processes adopted by the interviewed practitioners, together with the common motivations and issues that commonly take place during migrations.

In this work, we describe the results and provide an analysis of our survey, which includes a comparison of the migration processes, a ranking of motivations, and issues and some insights into the benefits achieved after the adoption. Maintainability and scalability were consistently ranked as the most important motivations, along with a few other technical and nontechnical motivations. Although return on investment was expected to take longer, the reduced maintenance effort in the long run was considered to highly compensate for this.

Microservices and Monolithic Applications

Identification of the most suitable software architecture is always a complex and important task since it impacts the future of the development of the application itself. Nowadays, the microservice architectural style is becoming increasing popular, and is being widely adopted by various large and small companies such as Amazon, Netflix, LinkedIn, SoundCloud, and many others.[1,2,3,4] However, microservices are not silver bullets and some companies are still hesitant to migrate because they cannot clearly evaluate the pros and cons.

The term "microservice" has been widely used since March 2012 to refer to applications developed

as a set of relatively small, consistent, isolated, and autonomous services deployed independently, with a single and clearly defined purpose.[5] Microservices are language-agnostic and can be developed independently by different development teams. Microservices are the opposite of monolithic architectures, where applications are usually deployed as a single package on a web container such as Tomcat (Apache Tomcat, http://tomcat.apache.org/) or JBoss (JBoss, http://www.jboss.org).

Numerous software architects are pushing this architectural style, but considering the migration costs, some practitioners are still hesitant to adopt microservices because they are not fully aware of the pros and cons.

So, software developers often choose to adopt one architecture over another based on their experience in previous projects or based on the perceived benefits of the new architecture. Therefore, it is important to study why microservices are adopted, to check the current motivation for their adoption, and to investigate whether specific issues are believed to require more improvement than others.

In order to elicit these motivations, we conducted an empirical study in the form of a survey, interviewing 21 practitioners who adopted a microservices-based architectural style at least two years ago.

Regarding the profile of our interviewees, they differ in several aspects:

- *Roles in their companies:* 31.82% of our participants were software architects, 27.27% project managers, 22.73% senior developers, 9.09% agile coaches, and 9.09% company chief executive officers (CEOs). All the interviewees had at least five years of experience in software development, including the CEOs.
- *Organization domain:* 28.57% of our interviewees work for banks, 28.57% for companies that produce and sell only their own software as a service (e.g., website builders, mobile app generators, and others), 23.81% in consultancy companies specializing in migration to microservices, 9.52% in the IT department of public administrations, and 9.52% in telecommunication companies.
- *Type of migration:* In order to address the most common pros and cons of migration to microservices, we considered both completed and ongoing migrations, also taking into account the experiences of software consultants who have supported several migrations and the experience of single companies that have already migrated or are still in the migration process.

Because of the complexity of finding participants experienced in migration to microservices-based architectural styles, we did not plan the sample of interviewees in advance. However, to avoid wasting the time of the practitioners and the interviewers, before the beginning of the interview we asked whether the participant had at least two years of experience in migrations. We therefore skipped ten participants who had only two months of experience; nobody else reported less than two years of experience.

The Questionnaire and the Survey

The goal of our survey was to analyze the motivations as well as the pros and cons of migrating from monolithic to microservices-based architectures. We structured the questions according to the information we needed to collect:

Company and Personal Information.

- *Interviewee's role:* We wanted to classify the role of our interviewees.
- *Company information:* We aimed to identify the company domain.
- *Role of the company with respect to software:* We aimed to understand whether the company produces software for other companies, for internal purposes, or if they are consultants.

Migration.

- *Motivations*. We aimed to understand the reason for the migration.
- Number of migrated applications
- Number of developed microservices
- *Migration process*. We aimed to characterize the migration processes adopted by experienced practitioners to understand the actual benefits and issues related to specific processes.
- *Costs*. We aimed to understand the costs overhead for the migration

We collected the information both with closed-answer questions and with open-ended questions to allow the participants to provide their opinion on the migration process. One of the most important goals of the questionnaire was to assess the importance of the motivations for the migration. Therefore, we asked the participants to rank their motivations based on a five-point Likert scale, where 0 meant "totally irrelevant" and 4 meant "fundamental". Moreover, we explained that the values of the scale had the sole purpose of ranking the motivations. As an example, a value of 5 for maintenance and 3 for

TABLE 1. The motivation drivers elicited in the survey.

Motivations	Entire Dataset		Migration Consultants		Others	
	#	Median	#	Median	#	Median
Maintainability	15	4	5	2	10	4
Scalability	15	2	3	3	12	2
Delegation of team responsibilities	11	3	1	3	10	3
DevOps support	8	3	2	1	6	3
Because everybody does it	6	4	2	3	4	4
Fault tolerance	2	4	2	4	/	/
Easy technology experimentation	2	3	2	3	/	/
Delegation of software responsibilities	1	4	1	4	/	/

separation of concerns indicates that maintenance is considered more important than separation of concerns, but the single values 5 and 3 have no meaning in themselves.

We ran the survey during the 17th International Conference on Agile Software Development (XP) among practitioners experienced in microservices. Each participant was interviewed by two authors at the same time. Then the two researchers separately interpreted and classified the answers. Finally, all disagreements were discussed and clarified. Even though this approach requires great effort, we believe it to be the most reliable way to collect subjective data based on open answers.

Qualitative and Quantitative Analysis

No noticeable differences emerged among different roles from the statistical analysis of the motivations, issues, and benefits, while several differences were found when we compared participants working in consultancy companies supporting migration to microservices and all other participants. Participants belonging to software development companies and companies whose main business is not software development (i.e., banks, telecommunication operators) reported a very similar set of answers. Therefore, in this section we report all the results by comparing "Migration Consultants" with other participants. Similar behavior is also reported independent of the participant roles. Software architects reported a set of answers comparable to that of developers, project managers, and others. The results are reported in Table 1 with their frequency and medians. Regarding motivations, issues, and benefits, we identified four importance groups based on the Likert scale adopted for ranking them (0 = totally irrelevant, 1 = not very important, 2 = slightly important, 3 = important, and 4 = fundamental). Here we describe the motivations reported by the practitioners interviewed. Please note that the answers are based on the summary of the description of the motivations provided by the participants and may slightly vary from the classical definitions provided in the literature.

Why Companies Migrated to Microservices

As for the motivations driving the adoption of microservices-based architectures, software maintenance was always reported and rated as very important by all the participants. Scalability, delegation of responsibilities to independent teams, and the easy support for DevOps also frequently drive adoption, while other motivations were only reported by migration consultants. One interesting observation is that several practitioners reported adopting microservices-based architectures because a lot of other companies are adopting them.

Maintainability. The modular architecture of microservices allows reducing the complexity of monolithic systems. Breaking a system into independent and self-deployable services enables developer teams to make changes and test their service independent of other developers, which simplifies distributed development. Moreover, the small size of each microservice contributes to increasing code

understandability, and thus to improving the maintainability of the code.

Scalability. Scaling microservices is easier than scaling monoliths. Scaling monolithic systems requires huge investment in terms of hardware and often fine-tuning of the code. If there is a bottleneck in some component, a more powerful piece of hardware can be used, or multiple instances of the same monolithic application can be executed across several services and managed by a load balancer.

In contrast, microservices are not automatically scalable, even if they are commonly deployed in elastic and stateless architectures. However, in a microservices-based system, each microservice can be deployed on a different server, with different levels of performance, and can be written in the most appropriate development language. If there is a bottleneck in one microservice in such a case, the specific microservice can be containerized and executed across multiple hosts in parallel, without the need to deploy the whole system to a more powerful machine.

Delegation of Team Responsibilities. Since microservices do not have external dependencies, they can be developed by different teams independently, reducing communication overhead and the need for coordination among teams. Each team owns the code base and can be responsible for the development of each service; it can maintain independent revisions and build environments based on their needs. The distribution of clear and independent responsibilities among teams allows splitting large project teams into several small and more efficient teams. Moreover, since microservices can be developed with different technologies and with a totally different internal structure, only high-level and technical decisions need to be coordinated among teams, while other technical decisions can be made by the teams themselves.

Because Everybody Does It. Microservices are cool, and a lot of large companies are adopting them. Therefore, some practitioners become interested and start investigating the potential of microservices. Over time, this interest commonly increases in practitioners. Our interviewees report that after using microservices in a testing environment or implementing some noncritical features, they started using it also for other features, always because of the perceived improvement regarding maintainability of the system. The main reason reported for this motivation is that, even if they are not totally aware of the real benefits, practitioners prefer following the mainstream, and want to avoid being out of the market in a few years because of a wrong technology choice.

DevOps Support. The adoption of a DevOps toolchain is supported by microservices. Since each service can be developed and deployed independently, each team can easily develop, test, and deploy their services independent of other teams.

Fault Tolerance. The failure of a microservice does not commonly impact the whole system. In contrast, in a monolithic application, the failure of a component might break the whole system. Moreover, faulty microservices can be quickly restarted. Some of our interviewees also reported that they automatically replace faulty microservices with previous versions of the same service to deliver the correct output without the need to restart the whole application.

Easy Technology Experimentation. Microservices are small by definition. Small components are faster to develop and therefore it is easier to experiment with new technologies or to develop new features. As an example, some participants reported that thanks to the polyglot nature of microservices, they were able to experiment with the introduction of a new service written in a new development language that is more suitable for the new service needs. A monolithic application would not have allowed development in a different language, or at least would not have allowed integrating it as easily and flawlessly into the existing system as the microservices architecture.

Separation of Software Responsibilities. Microservices are responsible for one single task within well-defined boundaries and are self-contained; therefore, development is greatly simplified.

Common Migration Issues

The adoption of a microservices-based architectural style is not always all peaches and roses. Some issues were commonly encountered by the practitioners we interviewed (Table 2). The main issues reported are the complexity to decouple from the monolithic system, followed by migration and splitting of data in legacy databases and communication among services. Effort overhead was only considered by non-consultants. People's minds are another personal reason against migration, followed by concern for the lack of return on investment (ROI) in the long run.

TABLE 2. The migration issues identified in the survey.

Issues	Entire Dataset		Migration Consultant		Others	
	#	Median	#	Median	#	Median
Monolith decoupling	7	3	/	/	7	3
Database migration and data splitting	6	4	/	/	6	4
Communication among services	4	3.5	2	4	2	3
Effort estimation	2	4	2	4	/	/
DevOps infrastructure requires effort	2	4	/	/	2	4
Library conversion effort	2	4	/	/	2	4
People's minds	2	4	1	4	1	4
Expected long-term return on investment (ROI)	2	3	1	3	1	3

Decoupling from the Monolithic System. The general behavior of our participants was to start the development of new non-critical features with a microservices-based architectural style.

Database Migration and Data Splitting. The migration of legacy databases needs to be addressed carefully. On the one hand, all our participants who are not working for consultancy companies reported that they adopted a microservice architecture connected to a legacy database or to existing database clusters even if this would partially reduce the benefits of microservices, as they were not always able to split existing data. On the other hand, the consultants recommended splitting the data in existing databases such that each microservice accesses its own private database.

Communication Among Services. Every microservice needs to communicate. Moreover, they cannot communicate internally but need to communicate on the network, adding complexity to the implementation besides possible network-related issues. Our interviewees reported that they rarely face network latency problems. However, they never had any bandwidth issues, probably because they are all implementing systems on highly reliable cloud infrastructures.

Effort Estimation and Overhead. Estimating the development time of a microservices-based system is considered less accurate than estimating a monolithic system. Despite our initial thoughts—that the effort overhead should be higher at the beginning of the project, but lower once the initial setup of the system is done—the interviewees always reported an effort overhead of nearly 20% more compared to the effort required for developing a monolithic solution. However, they reported that the benefits of increased maintainability and scalability highly compensate for the extra effort.

Effort Required for the DevOps Infrastructure. For all participants, the adoption of microservices required adopting a DevOps infrastructure. However, the adoption of the complete DevOps toolchain requires a lot of effort, which needs to be taken into account in addition to the development effort.

Effort Required for Library Conversion. Existing libraries require more effort for conversion. They cannot be simply reused, but rather need to be converted into one or more microservices, which again requires additional effort.

Service Orchestration Issues. Microservices-based architectural styles require an orchestration layer, which adds complexity to the system and needs to be developed reliably.

People's Minds. Changes in existing architectures are generally an issue for several developers. Our interviewees reported that older developers do not always believe in recent technologies. Moreover, a lot of them consider the legacy system as their own creation and are reluctant to accept such an important change to the software they wrote.

TABLE 3. The migration benefits identified in the survey.

Benefits	Entire Dataset		Migration Consultant		Others	
	#	Median	#	Median	#	Median
Maintainability improvement	9	4	5	4	4	3.5
Scalability improvement	7	2	5	2	2	4
ROI	6	4	/	/	6	4
Architectural complexity reduction	6	3	/	/	6	3
Simplifies distributed work	2	3	2	3	/	/
Performance improvement	2	1	2	1	/	/
Testability	2	3	/	/	2	3
Separation of concerns	2	3	2	3		
Single clear responsibility	2	1	2	1	/	/
Suitability for Scrum	2	3	/	/	2	3
System understandability	1	4	1	4	/	/

ROI Achieved in Longer Time (or Never) Compared to Monolithic Systems. Because of the aforementioned issues, which are mainly related to increased effort, ROI is perceived as an important issue. However, in several cases, practitioners reported that migration was the only choice, independent of the costs, since they were forced to migrate because of the lack of maintainability and the impossibility to scale their legacy systems.

Migration Benefits

In addition to the motivations and issues highlighted for the adoption of microservices, here we want to understand whether the initially perceived motivations and issues are also coupled with real benefits once microservices are actually adopted.

As reported in Table 3, improved maintenance was confirmed as the most important benefit. Practitioners reported that in the beginning they were aware that such a complex architecture might have a negative impact on software maintenance, but in the long run (at least one year, based on participant answers), they discovered that the architecture is less complex to understand and the system requires less maintenance.

Scalability was also named by several participants; however, while migration consultants stressed its importance, the others did not consider it as important to them.

Our participants also reported increased performance of the system, mainly because of the cloud nature of the new system and as a side benefit of increased scalability. A smaller number of participants highlighted as side benefits the separation of concerns due to the delegation of software responsibility to each service, and the single clear responsibility of each service.

Unexpectedly, microservices-based systems are less expensive than monolithic systems in the long run, allowing for good ROI. Our interviewees reported that ROI is achieved during maintenance of the system, as maintenance costs are lower than in monolithic systems. This also means that if quick prototyping is needed or if small projects are planned to be used for a short term, microservices could be more expensive than monoliths.

Migration Process

Our interviewees reported using three different processes for migrating from a monolithic system to a microservices-based one. In Figure 1, we report the process framework adopted by the interviewed practitioners. The three columns clearly identify the common and different steps in the three processes. This supports practitioners in selecting the most suitable migration process.

The aim of the first two processes is to migrate an existing monolithic system to a microservices-based one by reimplementing the system from scratch. The aim of the third approach is to implement only new features as microservices, to replace external services provided by third parties or develop features that need important changes and therefore can be considered as new features, thus gradually eliminating the existing system. However, some participants using the third process are planning to move some of their existing services to microservices as soon as possible as they need to work on their maintenance.

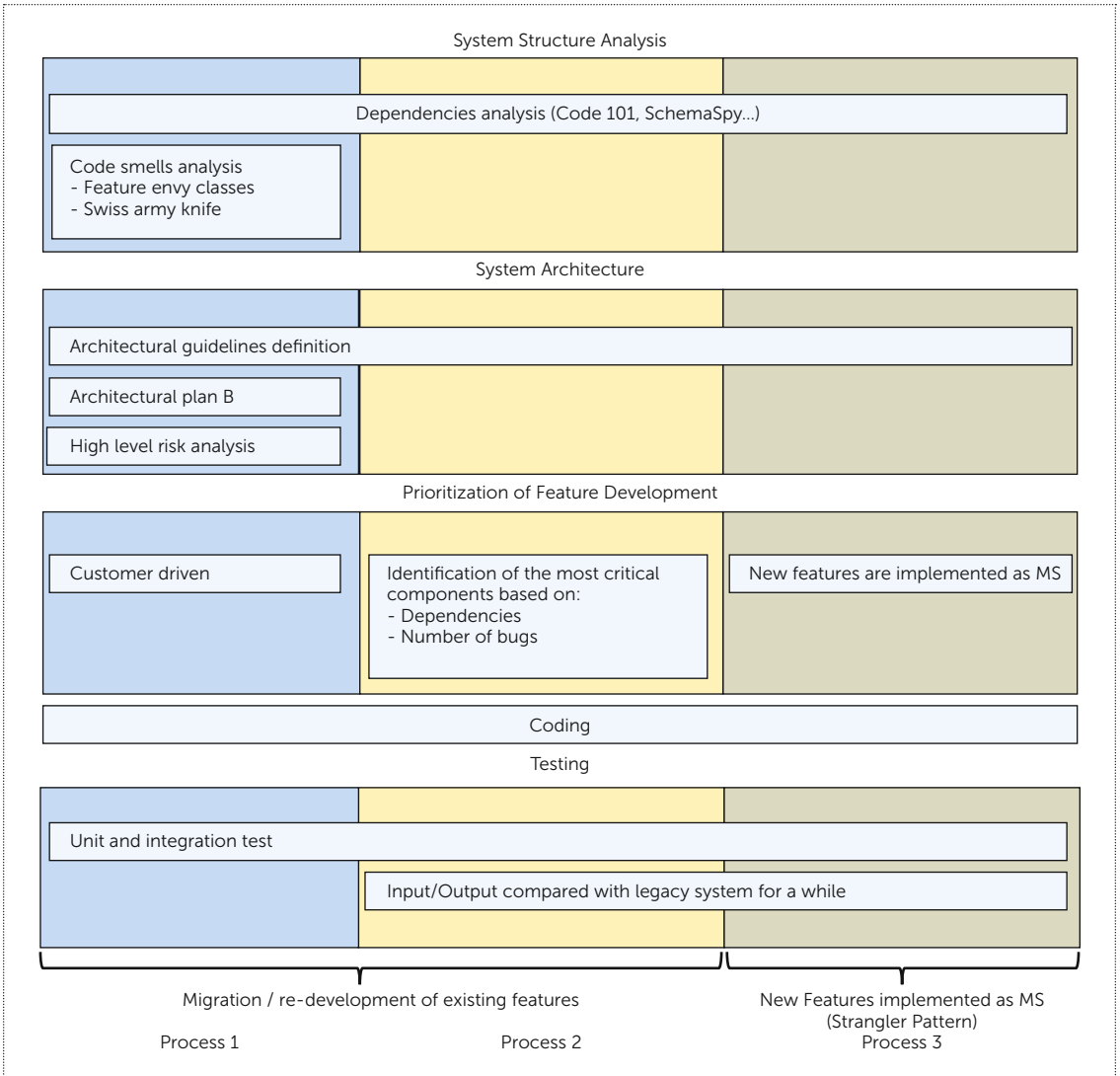

FIGURE 1. The identified migration processes framework.

All the participants working in consultancy companies recommended the first two approaches (24%). Process 2 was also adopted by half of the practitioners working in non-consultancy companies, while Process 3 was adopted by the remaining half. All the processes have some steps in common but differ in the details:

- *System structure analysis*. All processes start by analyzing systems, considering the system of systems and not only the single subsystem or component, so as to be able to clearly identify dependencies. Dependencies are then analyzed mainly with the support of tools (Structure101 (structure101.com), SchemaSpy (http://schemaspy.sourceforge.net/) or others, depending on the development language) or even manually. Process 1 also includes analyzing the presence of some structural code smells, such as "Feature Envy" or "Swiss army knife", to identify possible classes or methods that try to monopolize the system by carrying out most of the tasks.
- *Definition of the system architecture*. In this task, all processes propose defining the architectural structure of the system, collaboratively defining a set of architectural guidelines or principles, and proposing how to partition the system into small microservices. Moreover, they define the tools and frameworks to be used for the architecture, communication protocols, service APIs, and any other decisions that need to be agreed among the different services. Process 1 also includes an analysis of an architectural plan B

in case issues arise with the implementation of the designed architecture, and a high-level risk analysis of the adoption of the different architectures. Examples are analyzing the risk of splitting the database into separate databases or the risk of re-architecting/redeveloping some critical services.
- *Prioritization of feature/service development.* In this step, all the processes identify and prioritize the next microservices to be implemented. Here, the most important differences among the three processes emerge.

P1. Prioritization is only based on the customer value. The higher the customer value, the higher the development priority.

P2. First, the existing services are prioritized, with higher priority given to those with more bugs or to services that need more work done to immediately try to reduce the pain of the customer, always considering services with fewer dependencies. When the customer's priority is similar, the service with the lowest number of dependencies is developed first in order to deliver it as soon as possible.

P3. In this process, only new features are developed as microservices. Some practitioners refer to this approach as the "strangler pattern";[5] here, new microservices are added around the monolith. In the long run, the new ecosystem of microservices will gradually replace each feature of the existing monolith.

- *Coding* is then carried out independent from the process.
- The *Testing* phase is the last step of all processes. All processes prescribe unit and integration testing; P2 and P3 include testing of the microservices in parallel to testing the original component of the legacy system operating in the production environment. The microservice is tested as a black-box; it is deployed in the production environment and fed with the same input as the original component. The output is then compared with the output of the original component for a different time frame. The participants from consultancy companies and half of the remaining participants recommended running these tests for at least two months, while the remaining half of the non-consultancy participants adopted a different time frame depending on the frequency, the actual usage, and the risk of the service. In the case of P3, this approach applies for the development of new services to replace third-party external services or libraries.

Besides the benefits and issues reported in the previous section, the main benefit of the first two processes is the complete rearchitecture of the whole system, decomposing the whole system into microservices without continuing to work on the maintenance of the legacy system. Considering the third process, the main benefit is the lower migration cost, since the system is not completely redeveloped but only new features are added with a microservice architecture. Therefore, the main issue of the third process is he longer time needed to completely abandon the legacy system once all new features have been completely replaced by new ones. In theory, if some features do not need to be re-implemented, the legacy system could persist forever next to the new microservices system. In such a case, companies will need to decide if they want to keep both systems in parallel or if it is more convenient to re-develop the services that are still working in the legacy system.

Cost Overhead

As also highlighted by Singleton, Killalea, and Taibi et al., microservices architectures require extra machinery, which can impose substantial costs.[6,7,8] All the participants confirmed that the development of a microservices-based system has initial costs that are higher than those for developing a traditional system, with 24% reporting increased costs ranging from zero to 10% and the remaining 76% reporting an effort overhead between 20% and 30%.

All the participants also agreed that the higher cost overhead is highly compensated by the reduction of maintenance effort in the long run. They also reported that usually the initial extra effort is compensated after a period of between two years (33%) and three years (66%).

Effort estimation is commonly carried out as for any other project, based on the developers' experience, on analyzing the next development steps (57%), or supported by a work breakdown structure (43%). Moreover, all practitioners reported that they increased their estimation accuracy using a microservices-based architecture as a result of the lower granularity of the estimations and uncertainties due to the lack of external dependencies and the need for team synchronizations.

Data Interpretation

The statistical analysis of the collected data allowed us to collect interesting results, confirming some previous expectations regarding general beliefs and contradicting others.

Among the motivations that confirmed our expectations, technical motivations were the most important ones, while personal motivations also influence adoption. As for the technical motivations, reduced maintenance and increased system scalability were confirmed to be two of the most important drivers. Moreover, they were also confirmed to be not only drivers but also effective benefits reported after the adoption of microservices.

Considering the main issues identified, several indications emerged as well:

- *Technical issues:* Migrating to microservices requires more experienced developers, who must be able to decompose the system and develop new services decoupled from the monolith, migrate data to isolated services, and include new communication and orchestration mechanisms among services that are not needed for monolithic systems.
- *Economic issues:* Because of the aforementioned technical issues, microservices require more effort than monoliths. However, this is not only due to the technical issues but also to the required DevOps infrastructure.
- *Psychological issues:* Adopting a new technology that completely rearchitects the whole system is usually perceived as risky. Microservices are not an exception and, as expected, older developers and architects tend to be more conservative than younger ones in adopting new technologies.

The main benefits highlighted by the participants after the adoption of microservices confirm the reported motivations and issues. Only ROI was initially perceived as an issue, because of the cost overhead of microservices. However, unexpectedly the participants claimed that they experienced a return on their investment within a period of between one and three years.

As for the migration process framework reported in Figure 1, the framework can be easily used by practitioners who need to migrate their systems to allow them to easily identify the most suitable steps based on their requirements.

The processes confirm the power and suitability of DevOps when using a microservices-based architecture.[8,9] Moreover, compared to other works reporting migration processes, we support practitioners in the identification of the high-level process for migrating an existing system throughout the entire process, from requirements elicitation to the testing phase.[10]

Conclusion

The reasons for migrating to microservices are manifold. Several participants migrated to microservices simply because it was the only feasible solution for reducing the growing complexity of their systems. Therefore, maintainability as a consequence of the properties of some microservices is the key driver for migration, whereas the increased initial cost turned out to be one of the main issues hindering adoption, even if such costs were highly compensated after the adoption because of the long-term return on investment. The results match the main benefits identified in the literature, except for the lack of consideration of legal issues and the security threats highlighted by Esposito et al.[4,7,8,11,12]

We reported a migration process framework composed of three processes; two for the redevelopment of the whole system from scratch and one for creating new features with a microservice architecture on top of the existing system. The proposed process framework is based on real processes adopted by practitioners over at least two years, who reported on the main benefits and issues identified in this study, and can be easily used by companies to identify the most suitable migration process based on their needs.

Our study will continue with a set of new interviews in order to continue capturing the evolution of the motivations, benefits, and issues of the adoption of microservices.

References

1. S. Kramer, "GIGAOM—The Biggest Thing Amazon Got Right: The Platform," Oct. 2011; https://gigaom.com/2011110/12/419-the-biggest-thing-amazon-got-right-the-platforml.
2. T. Mauro, "Nginx—Adopting Microservices at Netflix: Lessons for Architectural Design," Feb. 2015; http://nginx.com/blog/microservices-at-netflix-architectural-best-practices.
3. S. Ihde, "InfoQ—From a Monolith to Microservices + REST: The Evolution of LinkedIn's Service Architecture," Mar. 2015; http://www.infoq.com/presentations/linkedin-microservices-urn.
4. P. Calcado, "SoundCloud—Building Products at Sound Cloud—Part I: Dealing with the Monolith," Jun. 2014; https://developers.soundcloud.com/blog/building-products-at-soundcloud-part-1-dealing-with-the-monolith.
5. J. Lewis and M. Fowler, "Microservices," Mar. 2014; www.martinfowler.com/articles/microservices.html.
6. A. Singleton, "The Economics of Microservices," *IEEE Cloud Computing*, vol. 3, no. 5, 2016, pp. 16–20.

7. T. Killalea, "The Hidden Dividends of Microservices," *Comm. ACM*, vol. 59, no. 8, 2016, pp. 42–45.
8. D. Taibi et al., "Microservices in Agile Software Development: A Workshop-Based Study into Issues, Advantages, and Disadvantages," *Proc. XP '17 Workshops*, 2017; doi 10.1145/3120459.3120483.
9. A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," *IEEE Software*, vol. 33, no. 3, 2016, pp. 42–52.
10. A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Migrating to Cloud-Native Architectures Using Microservices: An Experience Report," *Comm. Computer and Information Science*, vol. 567, 2016, pp. 201–215.
11. C. Pahl and P. Jamshidi, "Microservices: A Systematic Mapping Study," *Int'l Conf. Cloud Computing and Services Science*, 2016.
12. C. Esposito, A. Castiglione, and K.-K. R. Choo, "Challenges in Delivering Software in the Cloud as Microservices," *IEEE Cloud Computing*, vol. 3, no. 5, 2016, pp. 10–14.

DAVIDE TAIBI *is an assistant professor of software engineering at the Free University of Bozen-Bolzano. His research activities are focused on software quality and cloud migration, supporting SMEs and micro-enterprises in reducing project failures due to project maintainability issues. Formerly, he worked at the Technical University of Kaiserslautern (Germany), Fraunhofer IESE—Kaiserslautern (Germany), and University of Insubria (Italy) where he has researched for more than eight years. He obtained his PhD in computer science in 2011 working on quality models for Open Source Software. He is a member of the IEEE. Contact him at davide@taibi.it.*

VALENTINA LENARDUZZI *is a research assistant at the Free University of Bolzano-Bozen (Italy). She obtained her PhD in computer science in 2015 working on data analysis in software engineering. Her research activities are focused on empirical software engineering, agile processes, effort estimation, and functional size measurement. She spent eight months as visiting researcher at the Technical University of Kaiserslautern and Fraunhofer Institute for Experimental Software Engineering (IESE) working on Empirical Software Engineering in Embedded Software and Agile projects. Contact her at valentina.lenarduzzi@unibz.it.*

CLAUS PAHL *is an associate professor at the Free University of Bozen-Bolzano. Prior to his current employment he was a principal investigator of the Irish Centre for Cloud Computing and Commerce IC4. His research interests include software engineering in service and cloud computing. He holds a PhD in computing from the University of Dortmund and an MSc from the University of Technology in Braunschweig. Contact him at cpahl@unibz.it.*