MIGRATION FROM MONOLITHIC TO MICROSERVICE ARCHITECTURE: RESEARCH OF IMPACTS ON AGILITY

Josef Dolezal[1], Alena Buchalcevova[2],

Department of Information Technologies
Faculty of Informatics and Statistics
Prague University of Economics and Business
[1] xdolj12@vse.cz [2] alena.buchalcevova@vse.cz

Keywords

monolithic architecture, microservice architecture, agile software development, Scrum

Abstract

In recent years, the microservice architecture has been gaining popularity in software development and is replacing the monolithic architecture. The migration process from monolithic to microservice architecture is achievable more easily for software development companies that successfully adopted the agile approach. Aim of the paper is to identify the benefits and challenges of migrating from monolithic to microservice architecture from the agile software development approach perspective. The research is based on data gained in a software company successfully practicing the Scrum framework.

1. Introduction

In the monolithic architecture, all functionality is encapsulated into one single application, so any part cannot be executed independently and parts are tightly-coupled (Ponce et al., 2019). This type of architecture no longer meets the needs of scalability and rapid development (Tapia et al., 2020).

Microservice architecture represents a distributed approach where all application modules are microservices, i.e., independent processes interacting via messages. These services are highly de-coupled and are enabled for frequent deployment as per user requirements. Microservices can be implemented using various programming languages or databases (Sarita & Sebastian, 2017).

Microservice architecture is not suitable for every use case, and its implementation can be challenging. However, the popularity of microservices is rising, mainly because of its ability to solve maintenance problems and limited scalability of monoliths (Dragoni et al., 2017).

It seems like microservices fit into agile frameworks perfectly, as smaller teams can focus on individual services (Taibi et al., 2017). The nature of microservices increases software agility because each microservice becomes an independent unit of development, deployment, operations, versioning, and scaling (Jamshidi et al., 2018). However, some outcomes from practical experience argue that if

the development process is still a waterfall and software development practices and technologies like DevOps or Docker are not embraced, there could be problems in the development and maintenance of microservices (Kranc, 2017).

There is a lot of studies dealing with different aspects of the migration from monolithic to microservice architecture (Kazanavičius & Mažeika,2019; Mazlami et al.,2017; Blanch,2022; Taibi et al., 2017; Fowler & Lewis, 2014), however they do not examine the impact of the migration process on software development agility. Hence, this paper aims at filling this gap and analyzes the impact of migration from monolithic to microservice architecture on agility in a small agile software development company.

The paper is organized as follows. Section 2 presents the research methodology. Section 3 discusses the theoretical background. Section 4 then describes the results. Finally, Section 5 presents the conclusions.

2. Research Method

The main goal of this paper is to analyze the migration from monolithic to microservice architecture with emphasis on agile values of the Scrum framework. For this purpose, the following research questions were formulated:

RQ1: What are the benefits and drawbacks of the migration from monolithic to microservice architecture from the Product owner's perspective.

RQ2: What are the benefits and drawbacks of the migration from monolithic to microservice architecture from the Scrum master perspective.

RQ3: What are the benefits and drawbacks of the migration from monolithic to microservice architecture from the Developer's perspective in an agile team.

To find the answers to the questions above, we decided to conduct qualitative research based on semi-structured interviews (Wholey et al., 2010) with the people from the Scrum teams involved in the migration process. The following questions were prepared for interviews:

Q1: How has the migration from monolithic to microservice architecture affected your role in the software development process?

Q2: How has the migration from monolithic to microservice architecture affected the traditional Scrum ceremonies, namely Planning, Review, Daily Scrum, and Retrospective?

Q3: Have there been any changes to Scrum artifacts like Product Backlog, Sprint Backlog, or increment definition connected to the migration from monolithic to microservice architecture?

Q4: How has the migration from monolithic to microservice architecture improved or worsened the agile development process?

The research was conducted in a small software company providing logistics solutions. Company began the process of migrating from monolithic to microservice architecture about two years ago. There are three development teams in the company that have been using an agile approach to software development, specifically the Scrum framework, for six years.

The data collection took place in February 2022. In total, four interviews were conducted, participants were one Product owner, one Scrum master, and two Developers.

The first part of the interview was structured, while the second part was not structured to obtain more detailed information. Each interview lasted about 40-60 minutes. The outputs were written down. Although various aspects of migration from monolithic to microservice architecture were discussed during interviews, in this paper we focus on those related to agility. Specifically, we focus on benefits of the migration from monolithic to microservice architecture, disadvantages of the migration, impact of the migration on Scrum events and Scrum artifacts from the Product owner's, Scrum master's and Developer's perspective.

3. Background

In this section the basic concepts of the monolithic architecture, microservice architecture, agile software development, Scrum, and the migration from monolithic to microservice architecture are explained.

3.1. Monolithic Architecture

Monolithic architecture represents a traditional way of creating software. The monolithic application is a software in which different components (such as authorization, business logic, notification module, etc.) are combined into a single program developed on a single platform (Gos & Zabierowski, 2020). It is a single unit, which usually consists of a client, a server-side monolithic application, and a database (Figure 1). All the functions are served and managed in one unit.

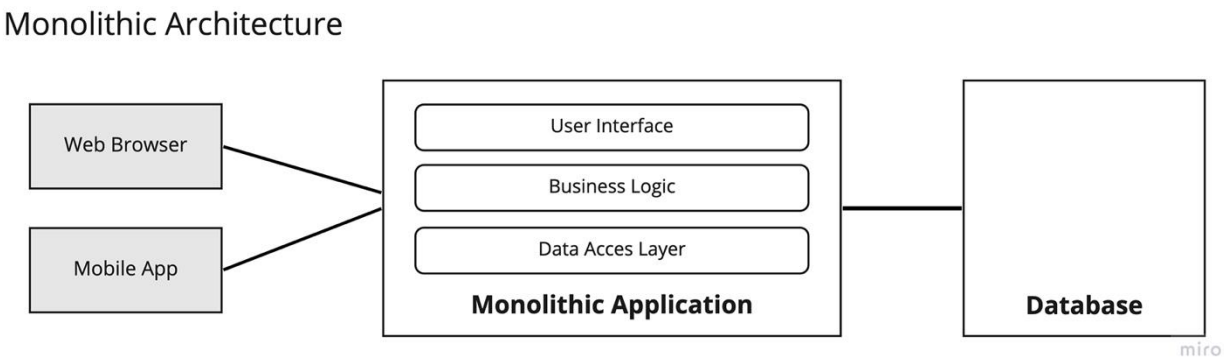

Figure 1. Monolithic Architecture (Kazanavičius & Mažeika, 2019)

Monolithic systems tend to have typically one large codebase. Whenever developers want to upgrade a specific aspect of the system, they must modify the entire application. So, because of the system's unified nature, every minor change affects the system as a whole (Insights, 2021). It is recommended for a small team at the founding stage to start with a monolithic architecture, as it is difficult to manage, e.g., microservices with only 2-4 people. Similarly, the monolithic architecture is recommended when developing an unproven product or a proof of concept. Other benefits that brings monolithic architecture include (Richardson, 2019):

- Ease of development. Monolithic architecture is better known and easier to implement.
- Relatively simple deployment. The entire application or system is uploaded as a single file, so complicated deployment is not needed.
- Easy testing and error tracing.
- Fast performance during the initial stages.

- Easier management. It's easier to set up, monitor, log, test, and deploy one solution than several separate units.

As for the disadvantages of monolithic architecture, the most important ones include (Lytvynenko, 2021):

- Poor scalability. Monolithic applications are easier to manage when they are small, but as it expands and employs new functions, it becomes more difficult to understand and scale.
- Poor stability. A problem in one module can crash the entire application due to the nature of the architecture.
- It can be more difficult to understand. Especially for new team members, it can be challenging to understand a huge monolithic system. With the growth of the application size, this problem gets worse.
- Minor changes are more complicated to implement. The whole system must be deployed even for minor fixes, which is inefficient.
- Future issues with speed. Deployment and launch time increase as an application grows.
- The monolithic application has a single tech stack. The process of implementation of new technology becomes highly complicated.
- Reliability. One error can possibly break down the entire system. This is one of the main disadvantages of monolithic architecture.

Although modern trends are pushing software companies to make their choice in favor of popular microservice architecture, monolithic architecture still has its benefits. However, when the application tends to become more complicated, the monolithic structure grows, becoming a large, hard to manage and scale piece of software.

3.2. Microservice Architecture

According to Lewis and Fowler (2014), the term microservices was first discussed at a May 2011 software architecture workshop to present a new architectural approach. Microservices are being employed by more and more companies around the world now, thanks to the results they are providing in software development processes (Baškarada et al., 2020).

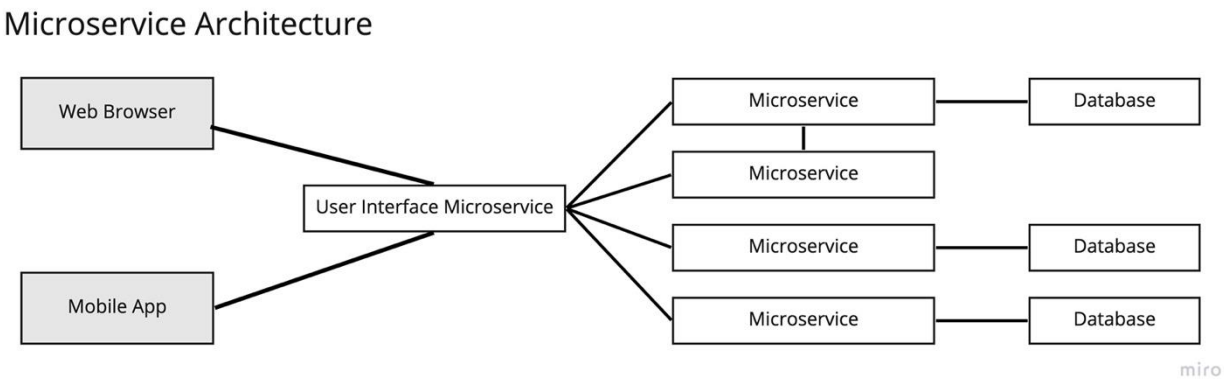

Figure 2. Microservice Architecture (Kazanavičius & Mažeika, 2019)

Microservice architecture builds applications as sets of independently deployable units that represent entities of a particular business or mission domain. As we can see in Figure 2, the client part can be

represented by a web browser or mobile application that communicate with the microservice providing a presentation layer. This User Interface microservice communicates with number of other microservices that provide the business logic.

For example, an e-shop system may include microservices that handle transactions associated with customers, invoices, basket, payments, etc. All data/information associated with each of these entities belongs to its respective microservice. A microservice that needs information outside the boundaries of its own entity (e.g., the warehouse service needs a customer's address) must get it from the corresponding microservice (Ponce et al., 2019).

While there is no precise definition of microservice architecture, there are certain common characteristics around the organization of business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data (Fellah & Bandi, 2021).

The most discussed reasons to start with microservice architecture are (Salah et al., 2016):

- Microservices scalability. Due to the system of individual microservices, the system as whole is flexible and capable of expanding. Including horizontal scaling – if one microservice experiences a significant load, it's the only one that needs a boost.
- Performance. Microservices-based solutions, if organized well, can outperform monolithic ones, especially when more complex software is involved.
- Greater stability. An application can run even if some of its microservices malfunction (fault isolation). This results in reduced downtimes.
- Better security. The relative isolation of microservice units typically means that attacks and data breaches will be more complicated to carry out on a system-wide scale.
- Easy to introduce new technologies to the product - each microservice can use different technology based on business, not technical requirements.
- Error-proof. Like the security, the microservice architecture allows establishing a boundary between certain parts of the system. This helps prevent unwanted mistakes – namely, connecting parts that shouldn't be connected. It also prevents tight couplings between the parts that should be linked.
- Simplified onboarding. Newcomers can jump on a specific microservice and immediately get into work, so they do not need to examine the entire system.

When considering the adoption of microservice architecture, it is necessary to take into account the disadvantages it brings (Lytvynenko, 2021):

- Operational overhead. Microservices are typically deployed on their own containers (docker) or virtual machines, which means lots of handling. These tasks should be automated with container fleet management tools. Independent deployment of each microservice is not effective.
- Complicated deployment. Large number of stand-alone services and connections between them require a more significant effort from developers to deploy the application.
- High initial cost. Complying hosting infrastructures, as well as skilled development teams to maintain the services, are expensive.
- Complicated debugging. It is necessary to trace the source of an error, which can become a challenge when an application consists of a variety of microservices, with each having its own set of logs.

- Greater resource consumption. Microservices architecture often requires not only more effort but more development time and manpower, which may not suit some companies.
- Complicated testing. Of course, a more complex system consisting of disparate services created using different tech stacks requires a more thorough approach to testing. Running a few automated scripts through the entire system will have no effect in such a situation.

3.3. Migration from Monolithic to Microservice Architecture

In general, there are two strategies how to migrate legacy monolithic software to microservice architecture. The first one is rebuilding, which means developing a new application. The second one is modernization, i.e., refactoring of the old application. Not all monolithic applications can be easily refactored to microservice architecture. Sometimes it is more economically beneficial to rebuild the application from scratch instead of refactoring it (Kazanavičius & Mažeika, 2019). As reasons speaking for building a new application we can state: (1) applications are built using very old languages and databases; (2) applications have a poor design, etc.

A key challenge in the process of migration is the extraction of microservices from existing legacy monolithic code bases. Identifying components of monolithic applications that can be turned into cohesive, standalone services is a tedious manual effort that encompasses the analysis of many dimensions of software architecture views and often heavily relies on the experience and know-how of the expert performing the extraction (Mazlami et al., 2017).

Because microservice architecture is a relatively new style and no widely approved way of doing migration exists, different organizations use different migration patterns and techniques (Blanch, 2022). According to Ponce, Márquez, and Astudillo (2019) we can name the Model-Driven approach, which uses design elements as input (using Domain-Driven Design), the Static analysis approach, which require the source code as input, and the Dynamic analysis approach, which analyze the system functionalities at runtime.

The adoption of a microservice architecture style often faces issues. According to Taibi et al. (2017), the main problems are the complexity to decouple from the monolithic system, followed by migration and splitting of data in legacy databases and communication among services. People's attitudes and minds are another reason against migration, followed by concern for the lack of return on investment in the long run. According to Fowler & Lewis, (2014) there is a high overall cost associated with decomposing an existing system to microservices and it may take many iterations.

3.4. Agile Software Development

An agile approach to software development has become very popular in the last ten years. It is represented by various methods, frameworks, and approaches that are based on values and principles of the Agile manifesto (Beck et al., 2001). Agile approach concentrates on a collaboration of teams, which are self-organized and cross-functional.

Without a doubt, the most popular agile framework is Scrum (Digital.ai, 2021). Scrum defines three roles Product owner, Scrum master and Developers. Product owner is responsible for representing the customer's best interest and has the ultimate authority over the final product. Scrum master is a facilitator, responsible for arranging the daily meetings, improving team interactions, and maximizing productivity. Developers are the people in the Scrum team that develop products using so-called Sprints. Sprints are cycles of work, typically one to four weeks each. At the start of each Sprint, within the Sprint planning meeting a cross-functional team selects items from the Product backlog and this way agrees on what they believe they can deliver within the Sprint (Sprint backlog).

Every day the team meets to briefly inspect its progress and adjust the next steps needed to finish the work remaining (Daily Scrum). At the end of the Sprint, the team members review the Sprint with stakeholders (Sprint review) and obtain feedback that can be incorporated in the next iteration. The purpose of the Sprint retrospective is to plan ways to increase quality and effectiveness for the next Sprint. Scrum emphasizes a working product at the end of the Sprint that is integrated, fully tested, and potentially shippable (Product increment) (Deemer et al., 2012; Schwaber & Sutherland, 2020).

4. Research Results

In this section the results of the research in a small software development company are presented structured according to individual interviewed roles.

4.1. The Impacts of Migration to Microservice Architecture on the Product Owner Role

In general, the Product owner role was noticeably affected by the migration from monolithic to microservice architecture. As one who represented the interests of the customer and the product the Product owner placed emphasis primarily on increasing the frequency of software deliveries. The move to microservice architecture has significantly improved the continuous delivery process. New functionality has not been deployed to the customer in fixed deployment cycles as before, not even at the end of each Sprint, but continuously within a Sprint. The delivery to the customer has been defined as a condition for the user story completion and has been incorporated in the Definition of Done. Increasing the frequency of deployments has led to an increase in the number of iterations with the customer and therefore increased the agility of the process.

Another important change that led to faster software delivery was the simplification of development parallelization. Teams could work on different services in parallel without worrying about inter-service dependencies.

Concerning Scrum ceremonies, the Product owner mentioned the biggest changes in the Sprint planning. With microservice architecture it was easier to structure development by features than by components, which among other things, also helped in communication with stakeholders.

As for Scrum artifacts, the Product backlog was mentioned by the Product owner specifically. The breakdown of the application into smaller parts has led to an increase in the number of User stories. On that account several User stories were aggregated into larger units called Features.

The Product owner, like other participants, also pointed out the increase in complexity of both the software development and the related processes.

4.2. The Impacts of Migration to Microservice Architecture on the Scrum Master Role

Compared to Product owner, Scrum master stated smaller number of changes caused by migration from monolithic to microservice architecture. The Scrum master mainly mentioned the areas of communication and coordination between teams and issues of growth in team member motivation.

Scrum master emphasized that breaking the application into smaller independent units simplified the distribution of application responsibilities among teams and helped with the cross-functionality of the teams. It also reduced the issues related to dependencies between teams, which has led to increased efficiency.

The Scrum master also pointed out on the fact that ability to define sprint goals more precisely in relation to microservices has increased the motivation of team members within the Sprint.

As for Scrum ceremonies and Scrum artifacts, the Scrum master has not observed major changes from his point of view. In particular, the Retrospective was addressed in more detail, although according to the Scrum master remained unchanged.

The Scrum master also mentioned the complexity of the transition to microservice architecture.

4.3. The Impacts of Migration to Microservice Architecture on the Developer Role

Naturally, the technological aspects of migration from monolithic to microservice architecture dominated the outputs of the interviews with developers. They did not hide the challenges that accompanied the migration process. They stressed that issues related to deployment, operation, and monitoring should not have been underestimated. Furthermore, DevOps concept and associated automation of processes was marked as crucial. It was necessary to meet the higher requirements for knowledge, experience, and technical proficiency of the development team.

On the other hand, microservices made it easier for development teams to manage their work, and lowering risks associated with creating new functionality. Adding or replacing individual microservices was much easier than redeploying the whole monolith.

The increased technological independence associated with migration from monolithic to microservice architecture was perceived positively. However, the possibility of developing microservices in whatever technologies and still achieving service integration and interoperability has not been fully used yet. The main reason was the fear of introducing non-standard or rare technologies that other developers would not be able to cover if needed.

A major benefit was presented in relation to the reduction of inter-team dependencies, which had caused considerable problems before the migration.

In relation to Scrum events, the most important change was perceived within the Sprint planning and Backlog refinement meeting.

As for Scrum artifacts, the most important presented change was the update of Definition of Done rules for User stories, which helped to increase the frequency of software deliveries in line with the principle of Continuous delivery.

4.4. Summary of the Impacts of Migration to Microservice Architecture

In this section results of the conducted interviews are summarized from the perspective of the Product Owner, the Scrum Master, and the Developer. In Table 1 perceived benefits of the migration from monolithic to microservice architecture are outlined with an indication which role the particular benefit has perceived. Table 2 depicts perceived disadvantages of the migration process again with the indication of which role the particular benefit has perceived. Then, Scrum Events mentioned by individual roles as affected by migration from monolithic to microservice architecture are presented in Table 3, while Scrum artifacts with the indication of which role the particular artifact has mentioned in Table 4.

Table 1. Perceived benefits of the migration from monolithic to microservice architecture from the point of view of the Product Owner, the Scrum Master, and the Developer.

Perceived benefit of the migration from monolithic to microservice architecture	**Product owner**	**Scrum master**	**Developer**
Increased frequency of software deliveries	x		x
Increased number of iterations with the customer	x		
Development parallelization	x		x
Team motivation		x	x
Cross-functionality of teams		x	x
Lower risk of adding new functionality	x		x
Increased technological independence and agility			x
Lower inter-team dependencies	x		x
Easier communication with stakeholders	x		

Table 2. Perceived disadvantages of the migration from monolithic to microservice architecture from the point of view of the Product Owner, the Scrum Master, and the Developer.

Perceived disadvantage of the migration from monolithic to microservice architecture	**Product owner**	**Scrum master**	**Developer**
Need for higher knowledge and experience of developers		x	x
High costs of the migration process	x		x
Need for deployment automation			x
More complicated monitoring			x
More complicated logging			x
More complex Product backlog	x		x

Table 3. Scrum Events affected by migration from monolithic to microservice architecture from the point of view of the Product Owner, the Scrum Master, and the Developer.

Scrum event	**Product owner**	**Scrum master**	**Developer**
Sprint planning	x		x
Daily scrum			
Sprint			x
Sprint review	x		x
Sprint retrospective			

Table 4. Scrum artifacts affected by migration from monolithic to microservice architecture from the point of view of the Product Owner, the Scrum Master, and the Developer.

Scrum artifact	**Product owner**	**Scrum master**	**Developer**
Product backlog	x		x
Sprint backlog			
Increment		x	x
Definition of Done		x	x

5. Conclusion

The aim of the paper was to investigate the impact of migration from monolithic to microservice architecture focusing on software development agility. The research was conducted in the environment of a small software development company that has been following the Scrum framework. In this context, limitation of the paper should be mentioned. The number of participants was not high as the research was conducted in only one software company. Therefore, the validity of the research output can be influenced by the specifics of a local environment.

The findings show that migration from monolithic to microservice architecture had an impact on all roles of the Scrum framework and the associated processes. The most affected by the changes was the Development Team, followed by the Product Owner. The Scrum Master was the least affected one.

Faster and more frequent deliveries of valuable features to customers, and an increase in the frequency of development iterations are perceived as the main benefits concerning development agility. Other described contributions supporting agility include reduced risks associated with the frequent inserting of new functionality to the application, more straightforward parallelization of development, increased developer motivation associated with clearer sprint goal setting, reduced cross-team dependencies, and easier management of the application scope, which is broken down into smaller units.

As for Scrum ceremonies and Scrum artifacts, changes are described mainly for Sprint planning and Backlog refinement meetings and Product backlog and Definition of Done artifacts.

Challenges that have been mentioned are mainly related to the increase of complexity that the microservice architecture has been introducing. This placed higher demands on knowledge and experience of developers. It was stressed that the deployment process, operational issues, and advanced monitoring should have been considered. Moreover, the DevOps concept and the associated automation was emphasized as critical for acceleration of development, testing, packaging, and deployment of microservice-based applications.

Acknowledgment. This work was supported by an internal grant funding scheme (F4/35/2022) administered by the Prague University of Economics and Business.

6. References

Baškarada, S., Nguyen, V., & Koronios, A. (2020). Architecting Microservices: Practical Opportunities and Challenges. *Journal of Computer Information Systems*, *60*(5), 428–436. https://doi.org/10.1080/08874417.2018.1520056

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., C. Martin, R., Mellor, S., Schwaber, K., Shuterland, J., & Thomas, D. (2001). *Manifesto for Agile Software Development*. https://agilemanifesto.org/

Blanch, R. (2022). Microservices: Strategies for Migration in a Brownfield Environment. *Medium*. https://medium.com/@rhettblanch_48135/microservices-strategies-for-migration-in-a-brownfield-environment-6c14335a8069

Deemer, P., Benefield, G., Larman, C., & Vodde, B. (2012). *A Lightweight Guide to the Theory and Practice of Scrum*. Ver, 2, 2012.

Digital.ai. (2021). *15th Annual State Of Agile Report | Digital.ai*. https://digital.ai/resource-center/analyst-reports/state-of-agile-report

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, Today, and Tomorrow. In M. Mazzara & B. Meyer (Eds.), *Present and Ulterior Software Engineering* (pp. 195–216). Springer International Publishing. https://doi.org/10.1007/978-3-319-67425-4_12

Fellah, A., & Bandi, A. (2021). Microservice-based Architectures: An Evolutionary Software Development Model. *CAINE 2020. The 33rd International Conference on Computer Applications in Industry and Engineering*, *75*, 41–48. https://doi.org/10.29007/1gx5

Fowler, M., & Lewis, J. (2014). *Microservices*. Martinfowler.Com. https://martinfowler.com/articles/microservices.html

Gos, K., & Zabierowski, W. (2020). The Comparison of Microservice and Monolithic Architecture. *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, 150–153. https://doi.org/10.1109/MEMSTECH49584.2020.9109514

Insights. (2021, September 7). Monolith vs Microservices: Everything You Need To Know. *Insights*. https://bambooagile.eu/insights/monolith-vs-microservices/

Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The Journey So Far and Challenges Ahead. *IEEE Software*, *35*(3), 24–35. https://doi.org/10.1109/MS.2018.2141039

Kazanavičius, J., & Mažeika, D. (2019). Migrating Legacy Software to Microservices Architecture. *2019 Open Conference of Electrical, Electronic and Information Sciences (EStream)*, 1–5. https://doi.org/10.1109/eStream.2019.8732170

Kranc, M. (2017). Thank you for not adopting microservices. *SD Times*. https://sdtimes.com/agile/thank-not-adopting-microservices/

Lytvynenko, O. (2021). *Monolithic vs Microservices architecture: What's the difference and which to choose?* CodeIT. https://codeit.us/blog/monolithic-vs-microservices-architecture

Mazlami, G., Cito, J., & Leitner, P. (2017). Extraction of Microservices from Monolithic Software Architectures. *2017 IEEE International Conference on Web Services (ICWS)*, 524–531. https://doi.org/10.1109/ICWS.2017.61

Ponce, F., Márquez, G., & Astudillo, H. (2019). Migrating from monolithic architecture to microservices: A Rapid Review. *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*, 1–7. https://doi.org/10.1109/SCCC49216.2019.8966423

Richardson, C. (2019). *Microservices Pattern: Monolithic Architecture pattern*. Microservices.Io. http://microservices.io/patterns/monolithic.html

Salah, T., Jamal Zemerly, M., Yeun, C. Y., Al-Qutayri, M., & Al-Hammadi, Y. (2016). The evolution of distributed systems towards microservices architecture. *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, 318–325. https://doi.org/10.1109/ICITST.2016.7856721

Sarita, & Sebastian, S. (2017). Transform Monolith into Microservices using Docker. *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*, 1–5. https://doi.org/10.1109/ICCUBEA.2017.8463820

Schwaber, K., & Sutherland, J. (2020). *Scrum Guide*. https://scrumguides.org/scrum-guide.html

Taibi, D., Lenarduzzi, V., & Pahl, C. (2017). Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation. *IEEE Cloud Computing*, *4*(5), 22–32. Scopus. https://doi.org/10.1109/MCC.2017.4250931

Taibi, D., Lenarduzzi, V., Pahl, C., & Janes, A. (2017). Microservices in agile software development: A workshop-based study into issues, advantages, and disadvantages. *Proceedings of the XP2017 Scientific Workshops*, 1–5. https://doi.org/10.1145/3120459.3120483

Tapia, F., Mora, M. Á., Fuertes, W., Aules, H., Flores, E., & Toulkeridis, T. (2020). From Monolithic Systems to Microservices: A Comparative Study of Performance. *Applied Sciences*, *10*(17), 5797. https://doi.org/10.3390/app10175797

Wholey, J. S., Hatry, H. P., & Newcomer, K. E. (2010). *Handbook of Practical Program Evaluation*. John Wiley & Sons.