





Serverless Software Engineering – and How to Get There.

Stephen McAleese¹, Jordan Conway McLaughlin¹, Filip Detyna¹, Andrey Murashev¹, Murat Yilmaz² , and Paul M. Clarke^{1,3} 

¹ School of Computing, Dublin City University, Dublin, Ireland
{stephen.mcaleese2, jordan.conwaymclaughlin24, filip.detyna2, andrey.murashev2}@mail.dcu.ie, paul.m.clarke@dcu.ie

² Department of Computer Engineering, Gazi University, Ankara, Turkey
my@gazi.edu.tr

³ Lero, The Science Foundation Ireland Research Center for Software, Dublin, Ireland

Abstract. Serverless computing is on the rise but developing software to exploit this space involves a deep rethink of software architecture, deployment, and operation (perhaps also, software development processes and team structures). Central to this revolution, we find a compelling argument for distributed, services-based software architectures. But converting a large, established monolith architecture system to microservices is non-trivial and fraught with both cost and risk. For the many firms with established software systems, this architectural system conversion might be considered the first stop-off on the journey to serverless computing. In tandem, software deployment and production monitoring also require reinvention. The focus of this paper involves an examination of the advantages of microservices architectures, include techniques for migrating from monolith architectures. Through application of a Multivocal Literature Review (MLR), we find that migrating from a monolith architecture to a microservices architecture is risky and non-trivial, but that there are techniques that can be employed to support the transition. We find also that monoliths have their advantages which might be overlooked to some extent in the race to serverless computing.

Keywords: Serverless software engineering · Monolith · Microservices · Migration

1 Introduction

The microservice architecture has become increasingly popular in the past several years as many companies have migrated their monolith applications to microservices [1]. However, the transition can be difficult and challenging [2]. Arguments for whether to adopt the microservices architecture, how to do so and what to expect during the transition exist in various forms both in the grey and white literature [36], though the focus of such literature varies greatly, and in some cases, there is conflicting information.

The term *microservice* was introduced in 2011 to describe an architectural style for web services where applications are composed of several small independently deployable services which each run in their own process, handle a single business capability and communicate with each other via lightweight mechanisms such as HTTP [3]. Generally, each microservice is owned by a single small “two-pizza” team which manages its development, deployment and testing [4]. Microservices have many advantages over monolithic applications such as better scaling, resilience, technological heterogeneity, replaceability [5] and modifiability [6].

There has been a significant increase in the popularity of microservices in recent years as many businesses have decided to use the microservice architecture over the more traditional monolithic architecture [1]. As of 2020, only 23% of organisations surveyed were not using the microservices architecture (Fig. 1).

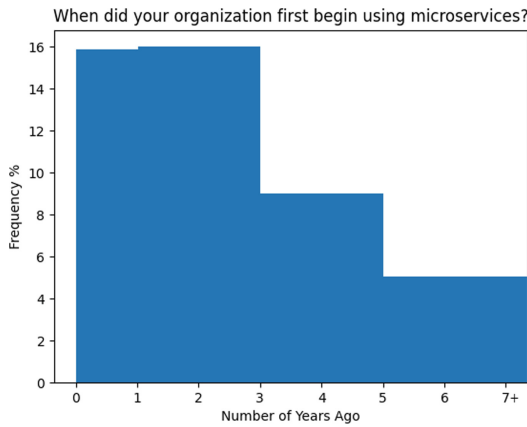


Fig. 1. Microservice adoption rates as of 2020 [1], 23% of respondents said they had not yet adopted microservice architecture.

The number of searches for the term ‘microservices’ has also increased significantly over the past several years. Google trends, an online tool for measuring the search interest of search terms, shows that the worldwide search interest for the search term ‘microservices’ from 2004 onward increased rapidly over the past several years [7].

Netflix and Amazon have been pioneers of the microservice architecture. Amazon launched their “Amazon Elastic Compute Cloud” in 2006 much ahead of their other competitors such as Google or Microsoft [8]. Netflix started to migrate their monolithic application to microservices running on AWS in 2009; two years before the term ‘microservice’ was introduced. Netflix’s adoption of this architecture has been pivotal in facilitating the company’s massive growth. The microservice architecture enables Netflix to regularly update their product for millions (Fig. 2) of users worldwide and provide uninterrupted access to their service [4].



Fig. 2. Worldwide Google search interest for the term ‘microservices’. Source: adapted from [7].

A *monolith* is an application where all the functions of the application are encapsulated in a single running process. The monolithic architecture was the established way of building applications [6]. Monolithic applications have their own strengths such as simple deployment and testing [9]. Therefore, the monolith can be viewed as the “least software” approach to developing an application and is therefore suitable for small companies. However, large monoliths are associated with many problems such as high complexity, slower deployment, scaling difficulties and technology lock-in [10]. There are pros and cons to monoliths and microservices and whether an organisation adopts the monolithic or microservice architecture depends on contextual factors such as the size of the company and the rate at which the software might ideally evolve. In this paper, we examine why microservices are appealing to software firms, what techniques might be employed when transitioning from a monolith architecture to microservices, and what risks are involved in the process. For serverless computing to be fully embraced, it seems that a large part of the initiative is dependent on a distributed software architecture. For this reason, this paper focuses on the benefits of microservices architectures, and later, the techniques that might be adopted when migrating monolith architectures to microservices.

2 Research Methodology

This research paper was written in the context of a Multivocal Literature Review (MLR) involving academic and non-academic literature also known as white and grey literature respectively [36]. Under the guidance of a senior academic, the primary research team was assigned the research topic “Monolith to Microservices Migrations: Techniques and Pitfalls”. The first task when conducting the MLR was to divide the research paper title into subproblems by formulating research questions to guide the research process. Each of the four primary researchers was assigned a single research question. The following research questions were identified:

- RQ1: Why have microservices increased in popularity in recent years?
- RQ2: What are the advantages of migrating from a monolithic to a microservice architecture?
- RQ3: What are some techniques for migrating applications from a monolithic to a microservices architecture?

- RQ4: What are the risks of migrating from a monolithic to a microservices architecture?

Once the research questions had been chosen, we used certain search keywords and search engines to create an initial pool of white and grey literature. The names of the search engines used to find academic literature were Google Scholar and IEEE Xplore and Google Search was used to find non-academic literature. To find relevant academic papers, the following search terms were used in Google Scholar: ‘monolith to microservices’, ‘microservices’ and ‘monolith to microservices migration’. For each search term in Google Scholar, we saved references to all the papers on the first search results page using the Mendeley reference manager. We also used snowballing to add additional white and grey literature to the initial document pool which ultimately contained 79 documents.

2.1 Selection Criteria

We then applied inclusion and exclusion criteria to filter out irrelevant or low-quality documents to create the final literature pool which contained 35 documents. To filter our collection of academic literature, we first excluded and removed duplicate papers, papers not written in English, papers with few citations and papers written more than ten years ago. We then only included papers considered to be relevant to our research questions and relevance was determined by reading the abstract of each document. When selecting grey literature documents such as online articles and blog posts, we included documents which were ranked high enough to be on the first search results page, were written by prominent industry figures and had enough supporting references. Grey literature documents were only included if they were insightful and well-written, and if presented from sources of ostensible credibility.

Before answering the research questions, we read through the papers in the final literature collection to create research notes, identify useful information and references relevant to answering our research questions. In the following analysis section, a subsection is dedicated to answering each research question.

3 Analysis

3.1 RQ1: Why have Microservices Increased in Popularity in Recent Years?

There has been a significant increase in the popularity of microservices over the past several years as more and more businesses choose the microservices architecture over the more traditional monolithic architecture [1]. In this section, explore the reasons for the increase in microservices to answer the first research question: “Why have microservices increased in popularity in recent years?”.

Reasons for the Increase in Popularity of Microservices. We identified several reasons in the white and grey literature for the increase in popularity of microservices: the advantages of the microservice over the monolithic architecture, increasing awareness of microservices, supporting technologies which have increased the viability of microservices and the changing software development culture.

Advantages of Microservices. One of the reasons why microservices have increased in popularity is the advantages they have over the monolithic architecture. Large monoliths have many downsides such as slower deployment speed, higher coupling, complexity and technology lock-in and many of these problems can be eliminated by migrating to a microservice architecture [10]. In recent years, an increasing number of companies have been motivated by these benefits to replace their monolithic applications with microservices causing an increase in the popularity of microservices.

Increased Awareness of Microservices. The increase in awareness of microservices in recent years [7] has probably contributed to the increased use of microservices [2]. Two reasons for this increased awareness include the use of microservices by high-profile companies such as Amazon and Netflix and the endorsement of microservices by prominent industry figures. Netflix solved many of their problems using microservices and since then many other high-profile companies such as Uber have also started using the microservices architecture [4]. The significant increase in interest in microservices after 2014 [7] coincided with the publishing of Martin Fowler’s article on microservices [3] and the book Sam Newman’s *Building Microservices* by Sam Newman in the same year [5]. Since then, many other prominent industry writers such as Chris Richardson have popularised microservices by writing books, articles and presentations on the subject [11].

Supporting Technologies. Containers and deployment automation have increased the feasibility of microservices. In the past deploying a monolith and all its dependencies could take hours. Deployment time also increases linearly without deployment automation. Therefore, without deployment automation or containers, deploying a microservice application could take an entire week which is not practical [12]. To make the deployment of microservice applications practical, several technologies are necessary such as continuous integration, containers, monitoring and logging [6]. Since many of these supporting technologies have only been developed in the past several years, microservices have only recently become practical, which explains the recent increase in popularity.

Cultural Changes. In the past several years, methodologies such as extreme programming and companies such as Amazon with its ‘two-pizza’ teams have advocated that small autonomous teams are more productive than larger teams [4]. Since each microservice can be owned by a single team, the microservices architecture is ideal for this new organisational structure [3]. Other cultural trends such as DevOps with its emphasis on developer ownership, monitoring and infrastructure are well-suited to microservices [13].

3.2 RQ2: What are the Advantages of Migrating from a Monolithic to a Microservice Architecture?

In this section, we describe the benefits of the microservice architecture over the monolithic architecture that have been described in the white and grey literature to understand why a business might want to migrate from a monolithic to a microservices architecture.

Scalability. Services can be scaled vertically by running the services on more powerful hardware or horizontally by duplicating an application on multiple servers [14]. There

are limits to vertical scaling because in practice the hardware the service is running on has finite computational resources [15]. Therefore, as a business scales its services, it eventually needs to scale horizontally. Monoliths can be scaled horizontally by running several instances of the monolith on multiple machines but since the monolith must be scaled as one unit, it is not possible to independently scale individual modules within the monolith. Microservices address this problem by having separate independently deployable and scalable microservices for each business capability [3]. Microservices can be scaled horizontally more effectively and efficiently because individual microservices can be scaled independently. Thus, microservices have a scaling advantage over monoliths, especially for services that process many requests.

Resilience. A problem in a monolithic application could cause the entire application to fail as the modules in a monolith are all running in the same process [2]. In contrast, as each microservice runs independently in a different process, the boundaries between microservices act as bulkheads and problems in a microservice can more easily be confined to that microservice causing degraded performance instead of a full application failure [5]. In 2008, when Netflix was using a monolithic architecture, a single mistake caused several days of downtime [4]. By breaking its monolithic application into microservices, Netflix was able to achieve much better availability and resilience.

Organisation Alignment. As software teams grow larger, the rate of development tends to slow down as the communication overhead is often higher in bigger teams. To address this problem, Amazon has a “two-pizza” team rule to ensure that teams are no larger than about ten people [4]. The traditional monolithic architecture does not align well with small, autonomous teams because it’s often not clear how to divide up the work of working on the monolith between teams. One common solution to the problem is to assign a team to each layer of the monolith. For example, there could be a front-end team and a back-end team. The problem with this organisational structure is that it is necessary for a team to collaborate with another team to make changes to a layer other than the one owned by the team. This makes it significantly more difficult to make changes outside of the team’s own layer. As a path of least resistance, each team will tend to implement changes in their own layer creating a siloed architecture [3].

In contrast, the microservice architecture offers much better organisational alignment for small teams as there is often a simple one-to-one mapping between teams and microservices. Each team can own a single microservice and easily make changes to it without needing to consult other teams resulting in higher agility and velocity [16].

Technological Heterogeneity. Monolithic applications are usually built with a single technology or programming language. However, a business using a monolithic architecture may have to commit to using a single technology for a long time because the cost of porting the entire system to a new technology is high [10] which could promote technological conservatism and discourage experimentation [2]. Instead, microservices offer technological heterogeneity where each service can be implemented in using a different technology that is best suited to the business capability [5]. If a technology becomes obsolete, an individual microservice can be easily updated or replaced because of its small size and low coupling with the rest of the system.

Higher Velocity. Another major benefit to microservices over monoliths is increased development and deployment velocity. As the codebase of a monolith increases in size and complexity, deployment velocity tends to decrease because the entire monolith needs to be tested, built and redeployed for every change [17]. As deployment slows down, teams may decide to deploy multiple changes at a time to maintain velocity. However, this strategy increases the risk of regressive changes being introduced to the system [5].

As a monolithic codebase increases in size and complexity, development velocity also tends to fall. The codebase becomes increasingly difficult to understand as it grows which makes it more difficult to make changes and add new features [6]. Large monoliths also slow down the onboarding of new hires because it takes longer for new hires to understand a large codebase than a smaller one [18].

Many of these problems can be eliminated by using a microservice architecture. Microservices do not grow beyond a certain size because new business capabilities are implemented in new microservices [3]. Consequently, the problems that arise from large codebases are less likely to arise when small microservices are used.

3.3 RQ3: What are Some Techniques for Migrating Applications from a Monolithic to a Microservice Architecture?

A business with a monolithic architecture (MA) might decide to migrate to a microservice architecture (MSA) once the problems associated with their growing monolith become greater than the cost associated with migrating it to a microservice architecture [6, 9, 20]. The following section explores several migration techniques which have been described in white and grey literature and answers the fourth research question, “What are some techniques for migrating applications from a monolithic to a microservices architecture?”.

Rebuild the Application from Scratch Using Microservices. A business with a monolithic application that is old, tightly coupled or highly complex might decide to replace it with a new application built from scratch using microservices if doing so is more cost effective than migrating the old application [6]. In addition to saving time and money, the new application could be built with modern technologies and would be less likely to inherit the undesirable complexity of the monolith. In most cases, however, a business’s monolithic application is unlikely to be so poorly implemented that replacing it completely would be easier than breaking it up into microservices.

Monolith to Microservices Migration Steps. Migrating a monolith to microservices is a complex and important challenge for businesses that can be challenging to execute successfully [2]. The challenge is to select the appropriate ‘candidate microservices’ in the monolith: modules or groups of modules in the monolith which are intended to be extracted and implemented as microservices later [20]. We now describe the high-level steps we believe are required to migrate a monolith to a microservice architecture. We propose that migrating from a monolith to a microservice architecture involves the following steps:

1. Identify candidate microservices in the monolithic application.

2. Choose a method or strategy for extracting the microservices candidates and execute it until the monolith has been partially or fully replaced by microservices.

Techniques for Candidate Microservice Identification. A variety of techniques for identifying candidate microservices in monoliths have been described in the white and grey literature. These techniques can be grouped into three high-level categories: model-driven, static-analysis and dynamic analysis approaches; though the most common approach described in the academic literature is the model-driven approach [17]. Model-driven approaches involve creating a visual model of the business using domain diagrams, UML diagrams or data flow diagrams and using the model to identify boundaries between candidate microservices [21]. Static analysis approaches decompose monoliths by identifying boundaries in the source code structure of the monolith application and dynamic analysis approaches identify boundaries by analysing the execution traces of the running monolith [17, 21]. The categories and approaches in each category we will describe are as follows:

- Model-driven approaches:
 - Decompose by business capability
 - A dataflow-driven approach
 - A graph-based approach
- Static analysis approaches:
 - Identify seams
- Dynamic analysis approaches:
 - Functionality-oriented microservice extraction

Decompose by Business Capability. Since each microservice should handle a single business capability [5], one logical decomposition approach is to decompose by business capability. Business capabilities can be identified using domain-driven design (DDD) [22, 23]. Another way to identify business capabilities is via communication with stakeholders [24]. Then a microservice can be created for each business capability.

Dataflow-Driven Approach. A monolith application can be modelled as a dataflow diagram and this diagram can be used to identify microservices. Chen R. et al. [20] describes a three-step algorithm that uses a dataflow diagram (DFD) to identify microservices. In the dataflow diagram, data storage components and operations are represented as boxes and ovals respectively. Then microservices are identified as pairs consisting of operations and their output data (Fig. 3).

Graph-Based Approaches. Mazlami G. et al. [25] describe a graph-based candidate microservice identification algorithm involving two steps: construction and clustering. In the construction phase, a graph representation of the monolith is generated. In the

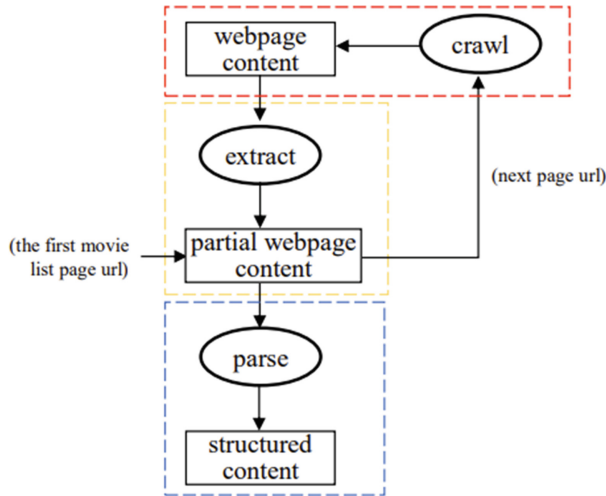


Fig. 3. Identifying microservice candidates using a dataflow diagram (adapted from [20]).

graph, nodes correspond to classes in the monolith and weighted edges between nodes indicate the level of coupling between classes. In the clustering phase, the graph is converted into a minimum spanning tree (MST) and edges are removed to partition the MST into several trees which are each clusters of highly coupled classes. These clusters are the candidate microservices.

Identify Seams. In the book *Building Microservices*, author Sam Newman states that monoliths can be decomposed into microservices by first identifying seams [5] which are sections of the codebase which can be modified or removed without affecting other components and are therefore good microservice candidates. To identify seams, Newman recommends using namespace constructs in the source code as a guide such as packages in the Java programming language (Fig. 4).

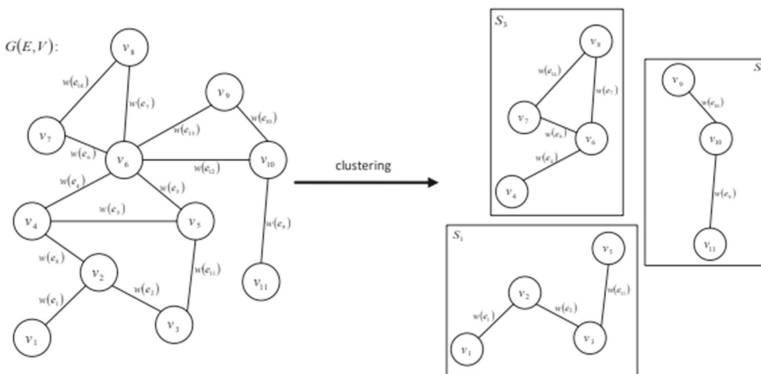


Fig. 4. Identifying microservice candidates using a graph of the monolith’s modules (adapted from [25]).

Functionality-Oriented Microservice Extraction. Jin, W. et al. have proposed a dynamic analysis approach named functionality-oriented microservice extraction which monitors the dynamic program behaviour of a monolith service, stores program behaviour in the form of logs and analyzes the logs to identify candidate microservices [26].

Combining Migration Techniques. Note that it may be effective to use several candidate microservice identification techniques. One strategy is to start with more abstract model-driven techniques and combine or verify the results of these methods using static or dynamic analysis approaches [27].

Microservice Extraction Strategies. Once candidate microservices have been identified in the monolith, the next step is to choose a strategy for extracting the candidate microservices to form actual microservices so that the monolith can eventually be replaced with microservices. When extracting microservices, Chris Richardson recommends using the following principles [28]:

1. *Migrate incrementally.* Services in the monolith should be converted to microservices incrementally rather than simultaneously to reduce risk and complexity.
2. *Migrate the services with the highest return on investment (ROI) first.* Richardson defines the services with the highest ROI as those that have the highest ease of extraction and the highest benefit of extraction. Richardson says that modules with more inbound dependencies are more difficult to decouple and thus have a lower ease of extraction. Modules with the highest benefit of extraction are those that would benefit most from the velocity benefits such as modules that are deployed frequently and the scaling benefits such as microservices that are under heavy load.

Richardson also describes a useful pattern named the strangler pattern for safely and gradually replacing a monolith with microservices in his book *Microservice Patterns* [29].

Strangler Application. To create a strangler application, microservices are extracted from the monolith and added to the strangler application. New functionality is implemented as new microservices in the strangler application to prevent the monolith from growing. Over time, as more services are extracted from the monolith, the strangler application grows while the monolith shrinks. Richardson outlines the following process for building a strangler application by extracting microservices from a monolith application [30]:

1. Split a module within the monolith to form two modules.
2. Split the database so that each module has its own database.
3. Create a new microservice for the new module.
4. Redirect traffic from the new module to the new microservice.
5. Delete the new module because it has been replaced by the microservice (Fig. 5).

The image above shows the result of applying the steps above. By repeating this process, the monolith is gradually replaced by microservices.

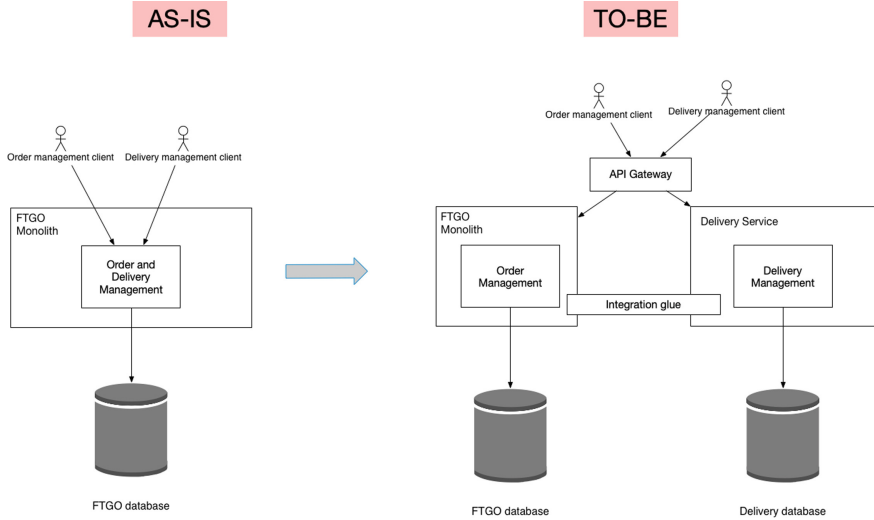


Fig. 5. Monolith to microservices migration using a strangler application (adapted from [30]).

3.4 RQ4: What are the Risks of Migrating from a Monolithic to a Microservices Architecture?

In our multivocal literature review, we have identified several risks associated with monolith to microservice migrations.

Unnecessary Migration. Although the benefits of microservices have been widely communicated, the microservice architecture is not the best architecture for all organisations. To make a monolith to microservices migration worthwhile, the costs of the migration such as extra code for inter-service communication and error handling should be exceeded by the benefits [18]. Microservices are generally more suitable for large organisations because the disadvantages of monoliths tend to increase as they become larger [20] and the benefits of microservices such as better scaling are only significant for heavily used services which are more likely to be found at large, established companies [18]. Therefore, monolith to microservices migrations should be avoided by small companies because the net benefit is more likely to be negative. Andy Singleton recommends that companies with fewer than 60 developers should not use microservices [18]. If a business has a good reason to migrate their application to a microservices architecture, there are still several pitfalls the business is vulnerable to.

Thinking Microservices are a Silver Bullet. Adopting microservices is not a substitute for other essential software engineering practices such as clean code, good design and automated testing and will not somehow cause all problems to disappear [31, 32]. Also, microservices will only benefit the company substantially if they are carefully implemented [32].

Adopting Microservices Without Changing Business Practices. A business that previously divided teams by layers (e.g. front end, back end) should change their team

structure after the adoption of microservices so that each team owns a single microservice [32]. Similarly, the team should adopt DevOps processes such as continuous delivery to manage their microservices effectively [31].

Difficulty Decoupling the Monolith. Extracting microservices from a monolith can be challenging [32] if there is a high degree of coupling in the monolith because a change in one part of the system would affect many components. Databases in monoliths tend to have particularly high levels of coupling and reputation for being difficult to decouple [5].

Unwillingness to Change. Developers who invested significant amounts of time into the development of the monolith may be reluctant to accept the significant change that is migrating to microservices. Traditional companies or older developers may be unwilling to accept the new microservices architecture because of its significant difference to more traditional architectures [32].

Increased Security Risk. In monolithic applications, modules communicate with each other within the same process via internal communication. In contrast, microservices use network calls to communicate with each other over a network. The problem is that the APIs microservices use to communicate with each other are exposed to the network resulting in a greater attack surface area [10]. If measures to compensate for this increased attack surface area are not introduced, the new microservices application could be less secure than the original monolith.

Lower Resilience. Microservices are a distributed system that relies on network calls instead of internal communication in the case of a monolith. These network calls can fail [5] and one problem could cascade through the system possibly resulting in lower resilience than the original monolith if methods for handling the network failures such as circuit breakers [5] are not put in place.

4 Limitations of Research

Although the researchers behind this paper have made every effort to create a complete and unbiased multivocal review it must be acknowledged that there are limitations and imperfections in our research. Perhaps the greatest limitation was that the initial literature review effort was conducted by four final year undergraduate students who had limited research knowledge and experience. This limitation has necessarily diminished the strictly academic quality of the work, but it is nevertheless felt that the findings are of interest to the community and that this can serve as a minor contribution to an important topic. The total time available to the researchers was also a constraint as the paper was researched and written as part of an assignment over several weeks, with the result that there are only limited references included, and even those included may include the effects of filter bubbles and recommendations of search engines. However, the research was guided by experienced software engineering academics at various stages, ultimately leading to the completion of this research paper. The fact that the steps involved in a multivocal review are well-defined also served to limit the possibility of errors being introduced.

Another significant limitation relates to the excessive volume of white and grey literature available through searches. For example, the search term ‘monolith to microservices’ returns over two thousand search results in Google Scholar which exceeds the limitation of most individual researchers. It is therefore necessary to sample only a small subset of those previous contributions. However, it is likely that a minority of these documents are highly relevant to our research topic and sorting results by relevance ensures that only the most relevant papers have been selected for this review.

A potential source of bias concerns the process of selecting relevant literature from the initial literature pool. To reduce the effect of subjectivity in this process, we used objective metrics such as the number of citations, publish date and search engine relevance where possible.

5 Directions for Future Research

We have observed that the distribution of research literature relevant to our research questions was uneven. We found a relative abundance of literature for the reasons behind the recent increase in the popularity of microservices, the advantages and risks of monolith to microservice migrations and methods for identifying microservice candidates in monoliths. However, we found that there was a lack of academic literature related to monolith to microservices migration steps after the identification of candidate microservices and practical code-focused migration strategies. To address this shortcoming, we found that it was often necessary to turn to grey literature to find relevant content. Therefore, we suggest that candidate microservice extraction strategies such as the strangler pattern and practical code-focused migration techniques could be areas with many opportunities for further academic research.

Although there are many advantages to the microservice architecture over the monolithic architecture, it may be that published material may tend to underemphasize the advantages of a monolithic architecture. Some companies such as Intercom [33], believe that the benefits of microservices have been overstated and that a monolithic architecture is often better for small companies. To reduce the risk of a bias in favour of the microservice architecture over the monolithic architecture, we believe there is an opportunity for future research that consolidates and articulates the advantages of the monolithic architecture.

Finally, since we had limited time when writing this paper, we believe there is an opportunity for a more thorough and complete multivocal review of monolith to microservices migrations. Also, as the software engineering discipline evolves, new ideas and practices related to the topic are likely to create new opportunities for research.

6 Conclusions

By applying a multivocal literature review, we have identified some of the reasons behind the increase in popularity of the microservice architecture, and the advantages and pitfalls associated with monolith to microservices migrations. Microservices promote lower coupling, increased organisational alignment, better scalability, velocity and resilience, and technological heterogeneity. We identified several monolith to microservices migration

techniques from three classes of migration techniques: model-driven, static analysis and dynamic analysis techniques and described how they could be used to identify boundaries in monoliths between candidate microservices. We then described high-level monolith to migration strategies and strategies for effectively and safely migrating monoliths to microservices such as the strangler application. Finally, we outlined some of the risks that can arise when migrating a monolithic application to microservices such as migrating for the wrong reasons, seeing microservices as a silver bullet and security risks.

Recent technologies related to deployment automation and service monitoring have increased the viability of microservices as an alternative to the monolithic architecture, resulting in increased popularity. It is therefore the case that the apparent rising adoption of microservices is critically dependent on a cocktail of other technology advancements, one further example of which is serverless computing, a paradigm in which software providers do not concern themselves with hardware. This might be particularly the case for highly distributed architectures (those with many microservices) in a serverless cloud computing environment, as software providers potentially only need to pay for services when they are executing. One example of this can be found in Function-as-a-Service (FaaS) [34]. Much is changing in this space, it is not just a technology fad, fundamental economic considerations are also present.

Discussions regarding the rise of microservices are potentially misleading, because it is not just microservices as a standalone concept that is on the rise, it is the convergence of several emerging concepts that when orchestrated together, present with a harmony that is appealing to software firms. It might be beneficial to academics and practitioners alike if this convergence was given a dedicated name: the primary constituent elements are serverless computing, microservices architecture, automated build and deployment pipelines, and service monitoring. In previous work we termed this *Continuous Software Engineering* [35], but even that concept has now been stretched. Perhaps this should be replaced with the term *Serverless Software Engineering*.

Acknowledgements. This research is supported in part by the Department of Enterprise, Trade and Employment, Ireland (<https://enterprise.gov.ie/en/>) under the Disruptive Technologies Innovation Fund grant number DTIF DT20180116, and also supported in part, by SFI, Science Foundation Ireland (<https://www.sfi.ie/>) grant No SFI 13/RC/2094 P2 to Lero – the Science Foundation Ireland Research Centre for Software.

References

1. Loukides, M., Swoyer, S.: Microservices Adoption in 2020 – O’Reilly. <https://www.oreilly.com/radar/microservices-adoption-in-2020/> (n.d.). Accessed 1 February 2022
2. Taibi, D., Lenarduzzi, V., Pahl, C.: Processes, motivations, and issues for migrating to microservices architectures: an empirical investigation. *IEEE Cloud Comput.* **4**(5), 22–32 (2017). <https://doi.org/10.1109/MCC.2017.4250931>
3. Fowler, M.: Microservices. <https://martinfowler.com/articles/microservices.html#footnote-etymology> (n.d.). Accessed 1 February 2022
4. Rud, A.: Why and how Netflix, Amazon, and Uber migrated to microservices: learn from their experience – HYS enterprise. <https://www.hys-enterprise.com/blog/why-and-how-netflix-amazon-and-uber-migrated-to-microservices-learn-from-their-experience/> (n.d.). Accessed 15 February 2022

5. Newman, S.: Building Microservices – Sam Newman – Google Books. https://books.google.ie/books?hl=en&lr=&id=ZvM5EAAAQBAJ&oi=fnd&pg=PT8&dq=building+microservices&ots=uh8heDdFXl&sig=U_FvCd-VitpQmi249fxelnEXjQc&redir_esc=y#v=onepage&q=building%20microservices&f=false (n.d.). Accessed 2 February 2022
6. Kazanavicius, J., Mazeika, D.: Migrating legacy software to microservices architecture. In: 2019 Open Conference of Electrical, Electronic and Information Sciences, EStream 2019 – Proceedings (2019). <https://doi.org/10.1109/ESTREAM.2019.8732170>
7. Data source: Google Trends. <https://www.google.com/trends>
8. Miller, R.: How AWS came to be – TechCrunch. <http://tcn.ch/29cG0Gh> (2016). Accessed 11 February 2022
9. de Lauretis, L.: From monolithic architecture to microservices architecture. In: Proceedings – 2019 IEEE 30th International Symposium on Software Reliability Engineering Workshops, ISSREW 2019, pp. 93–96 (2019). <https://doi.org/10.1109/ISSREW.2019.00050>
10. Dragoni, N., Giallorenzo, S., Lafuente, A. L., et al.: Microservices: yesterday, today, and tomorrow. *Present Ulterior Softw. Eng.*, 195–216 (2017). https://doi.org/10.1007/978-3-319-67425-4_12
11. Richardson, C.: What are microservices? <https://microservices.io/> (n.d.). Accessed 1 February 2022
12. Gouigoux, J.P., Tamzalit, D.: From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture. In: Proceedings – 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings, pp. 62–65 (2017). <https://doi.org/10.1109/ICSAW.2017.35>
13. Amazon: What is DevOps? – Amazon Web Services (AWS). <https://aws.amazon.com/devops/what-is-devops/> (n.d.). Accessed 17 February 2022
14. Guitart, J., Beltran, V., Carrera, D., Torres, J., Ayguadé, E.: Characterizing secure dynamic web applications scalability. In: Proceedings – 19th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2005 (2005). <https://doi.org/10.1109/IPDPS.2005.137>
15. Qu, C., Calheiros, R.N., Buyya, R.: Auto-scaling web applications in clouds. *ACM Comput. Surveys (CSUR)* **51**(4), 33 (2018). <https://doi.org/10.1145/3148149>
16. Prasandy, T., Murad, D.F., Darwis, T.: Migrating application from monolith to microservices. In: Proceedings of 2020 International Conference on Information Management and Technology, ICIMTech 2020, pp. 726–731 (2020). <https://doi.org/10.1109/ICIMTECH50083.2020.9211252>
17. Ponce, F., Marquez, G., Astudillo, H.: Migrating from monolithic architecture to microservices: a rapid review. In: Proceedings – International Conference of the Chilean Computer Science Society, SCCC, 2019-November. <https://doi.org/10.1109/SCCC49216.2019.8966423>
18. Singleton, A.: The economics of microservices. *IEEE Cloud Comput.* **3**(5), 16–20 (2016). <https://doi.org/10.1109/MCC.2016.109>
19. Richardson, C.: Introduction to microservices | NGINX. <https://www.nginx.com/blog/introduction-to-microservices/> (n.d.). Accessed 15 February 2022
20. Chen, R., Li, S., Li, Z.: From Monolith to microservices: a dataflow-driven approach. In: Proceedings – Asia-Pacific Software Engineering Conference, APSEC, 2017-December, pp. 466–475 (2018). <https://doi.org/10.1109/APSEC.2017.53>
21. Fritzsche, J., Bogner, J., Zimmermann, A., Wagner, S.: From monolith to microservices: a classification of refactoring approaches. In: Bruel, J.-M., Mazzara, M., Meyer, B. (eds.) *DEVOPS 2018*. LNCS, vol. 11350, pp. 128–141. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-06019-0_10

22. Khononov, V.: Learning domain-driven design – Vlad Khononov - Google Books. <https://books.google.ie/books?id=qAtHEAAAQBAJ&printsec=frontcover&dq=bounded+context&hl=en&sa=X&ved=2ahUKEwj35MTuyfL1AhVhmVwKHT7iAv8Q6AF6BAGEEAI#v=onepage&q=bounded%20context&f=false> (n.d.). Accessed 9 February 2022
23. Richardson, C.: Decompose by subdomain. <https://microservices.io/patterns/decomposition/decompose-by-subdomain.html> (n.d.). Accessed 9 February 2022
24. Bucchiarone, A., Dragoni, N., Dustdar, S., Larsen, S.T., Mazzara, M.: From monolithic to microservices: an experience report from the banking domain. *IEEE Softw.* **35**(3), 50–55 (2018). <https://doi.org/10.1109/MS.2018.2141026>
25. Mazlami, G., Cito, J., Leitner, P.: Extraction of microservices from monolithic software architectures. In: *Proceedings – 2017 IEEE 24th International Conference on Web Services, ICWS 2017*, pp. 524–531 (2017). <https://doi.org/10.1109/ICWS.2017.61>
26. Jin, W., Liu, T., Zheng, Q., Cui, D., Cai, Y.: Functionality-oriented microservice extraction based on execution trace clustering. In: *Proceedings – 2018 IEEE International Conference on Web Services, ICWS 2018 – Part of the 2018 IEEE World Congress on Services*, pp. 211–218 (2018). <https://doi.org/10.1109/ICWS.2018.00034>
27. Fan, C. Y., Ma, S. P.: Migrating monolithic mobile application to microservice architecture: an experiment report. In: *Proceedings – 2017 IEEE 6th International Conference on AI and Mobile Services, AIMS 2017*, pp. 109–112 (2017). <https://doi.org/10.1109/AIMS.2017.23>
28. Richardson, C.: Decompose your monolith – Six principles for refactoring a monolith to microservices. <https://chrisrichardson.net/post/refactoring/2020/07/28/six-principles-for-refactoring-to-microservices.html> (n.d.). Accessed 9 February 2022
29. Richardson, C.: Microservices patterns. <https://microservices.io/book> (n.d.). Accessed 11 February 2022
30. Richardson, C.: Refactoring a monolith to microservices. <https://microservices.io/refactoring/> (n.d.). Accessed 8 February 2022
31. Richardson, C.: Microservices adoption antipatterns – the series. <https://microservices.io/microservices/antipatterns/-/the/series/2019/06/18/microservices-adoption-antipatterns.htm> (n.d.). Accessed 16 February 2022
32. Carrasco, A., van Bladel, B., Demeyer, S.: Migrating towards microservices: migration and architecture smells. In: *IWoR 2018 – Proceedings of the 2nd International Workshop on Refactoring, Co-Located with ASE 2018*, pp. 1–6 (2018). <https://doi.org/10.1145/3242163.3242164>
33. Scanlan, B.: 10 technical strategies to avoid when scaling your startup (and 5 to embrace) – Inside intercom. <https://www.intercom.com/blog/ten-technical-strategies-to-avoid-when-scaling-your-startup-and-five-to-embrace> (n.d.). Accessed: 13 February 2022
34. Grogan, J., et al.: A multivocal literature review of Function-as-a-Service (FaaS) infrastructures and implications for software developers. In: Yilmaz, M., Niemann, J., Clarke, P., Messnarz, R. (eds.) *EuroSPI 2020. CCIS*, vol. 1251, pp. 58–75. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56441-4_5
35. O’Connor, R.V., Elger, P., Clarke, P.: Continuous software engineering – A microservices architecture perspective. *J. Softw. Evol. Process* **29**(11), 1–12 (2017)
36. Garousi, V., Felderer, M., Mäntylä, M.V.: Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *J. Inf. Softw. Technol.* **106**, 101–121 (2019)