

# Microservices Boundary Determination Migration in DevOps: a Case Study

Ali TaeiZadeh<sup>1\*</sup>, Zahra Lotfi<sup>2</sup> and Ali J. Ramadhan<sup>1,3</sup>

<sup>1</sup>University of Qom, Qom, Iran

<sup>2</sup>TIS Research Center, Tehran, Iran

<sup>3</sup>University of Alkafeel, Najaf, Iraq

**Abstract.** The microservice architecture (MSA) is a widely used and researched approach in industry and academia. However, designing the migration to MSA is a complex and challenging task, and there is a lack of clear guidelines on how to address both business and technical issues during the process. This paper presents a step-by-step method for determining the boundaries of microservices, which is a critical activity in MSA migration for both practitioners and academics. We conducted a case study of SHAMIM, a nationwide project serving over 120,000 students, to demonstrate the proposed method. The microservice boundaries were determined by utilizing both business and technical input types, including domain-driven design (DDD), business processes, service call numbers, and data access patterns in databases. Our findings suggest that MSA migration can be conducted more reliably by using change documents maintained by the DevOps team. The proposed method leads to clear improvements in the determination of each microservice boundary, and it can be useful for practitioners and academics involved in MSA migration. INDEX TERMS Microservice Architecture, Microservices Migration, Monolith, microservice boundaries boundaries, determination, DevOps.

## 1 Introduction

Recently, there has been a hype in modernizing information technology processes and systems through cloud computing. It is argued that monolith applications cannot achieve desired cloud capabilities, such as optimized containerization and clustering of critical parts of a system. [1]. Additionally, maintaining a monolith application reliably and in control can be challenging in the long run [2]. Supporting a monolith requires large teams with a plethora of work, which can result in slow development and deployment of new versions. To address this issue, both industry and academics have turned their attention to Microservice Architecture (MSA) due to its ability to provide independence and scalability, which is similar to the service-oriented architecture (SOA) [3]. Unlike monolith or even SOA-based monolith, MSA promises agile deployment that requires a small team and loosely coupled microservices. Therefore, the microservice migration from monolith applications has gained considerable attention in both academic and industry areas [4].

\* Corresponding author: [alitaee@gmail.com](mailto:alitaee@gmail.com)

Moreover, MSA has been applied in a majority of cloud projects, with over 80% of cloud-native applications using it [5]. MSA facilitates cloud-native benefits, as cloud-based technologies like Kubernetes and Docker cannot be realized without independent and scalable microservices [3],[6].

However, the expected outcomes of SOA, such as service independence and scalability, have not been achieved due to unresolved issues in the design phase [7]. Similarly, MSA relies on the quality of the design phase to address the issues in SOA. Determining microservice boundaries is the first and most crucial step in the design process, and many studies have been dedicated to this research area due to its challenges [7],[8]. In terms of design phase issues, independence and granularity remain unresolved challenges, and SOA's pitfall is still unaddressed [10].

The determination of microservice boundaries indicates the starting point of incremental migration, which clarifies the priority of each part in the process. Additionally, the orientation of MSA migration, whether business or software-oriented, is crucial since it determines the type of inputs or techniques that can be applied [11]. Several studies emphasize identifying microservices based on business capabilities [12]; however, there is a lack of consensus on this approach in SOA [13]. Moreover, due to the descriptive nature of high-level documents in the business domain, such as DDD, making decisions on granularity and boundaries can be ambiguous.

Furthermore, the majority of MS boundaries determination methods rely on manual tasks (e.g., preparation of inputs) which are costly and time-consuming [14], [15]. There is a lack of expertise and tools that are required for MS's boundaries determination [7]. Essential expertise resulted from the collaboration between various teams and stakeholders was emphasized by several studies [9]. This is where DevOps can play an important role [3]. DevOps is a set of techniques that promote the collaboration between network admins, programmers, and managers. Several studies identified the DevOps as a facilitator of MSA migration [3], [6], [16], [17]. The reason is that DevOps success depends on upgrading the team culture which include changing the existing activities or changing the hierarchy structure of the team and creating a collaborative atmosphere between the team members [1].

The shared entities such as, ESB for microservice communication, shared libraries, and shared relational databases could hinder MSA to assure independent microservices [18]. Dedicating a database per microservice requires a prerequisite in software development and infrastructure domain [19]. The problem of the microservice' dependencies on other entities issue is still considered as an unsolved issue related to optimized boundary determination.

Decomposition is one of the most critical steps in MS boundaries determination [20]. Decomposition has emerged as a tool or method of determining boundaries according to measures like cohesion and coupling of business-oriented entities or based on business functionalities; for example, Mono2Micro is a tool that decomposes based on business use cases [1]. In addition, manual-based decomposition method relies on increasing expert collaborations for analyzing business capabilities for boundaries determination [9]. However, abstraction level of input documents of MS identification process is important, selecting high-level documentation make the measures descriptive and leads to ambiguity that several studies have pointed out as a core challenge [11], [21].

In particular, input types selection for determining MS boundaries is crucial in satisfying the business or technical aspects, where the existing migration methods mainly rely on only high level input types in business aspect. This research area has attracted the attention of several researchers; however, there is a lack of practical guidelines to ease the boundaries determination in the migration process [16]. Moreover, review of literature indicates that few studies have provided guidelines, while several studies lack guidelines.

The purpose of this paper is to facilitate the MSA migration by solving the core issue in MS boundaries determination to identify microservices. This paper reviews the literature,

carries out a case study, and conducts interviews with experts to address the following specific objectives:

Objective 1: To identify the main challenges in the migration process to MSA faced by practitioners and academics.

Objective 2: To achieve clear steps for boundaries determination that satisfies both business and technical aspects.

Objective 3: To propose suitable guidelines to ease the boundaries identification in a DevOps firm.

In our case study, we have studied a large monolith application, called ‘SHAMIM’, which provides an e-learning service. It embeds different learning-related services, various technologies in the frontend, backend, mobile app, virtualization, clustering database, etc. constituting a large technical team and complex system that resists changes. Proposing a method for migration to MSA without impacting the current system was a necessity.

The remainder of this paper is organized as follows: section 2 discusses the literature review; section 3 presents the methodology, sections 4 and 5 describe the case study and results, and section 6 provides discussion, conclusion, and implications for future work.

## 2 Literature Review

### *Microservice and SOA*

The concept of service orientation and the enterprise service bus (ESB) were introduced to address problems with monolithic applications. Service-oriented architecture (SOA) has since evolved, and it is generally agreed upon that web services play a critical role as a communication model, leading to the addition of API management to the software ecosystem. However, the important goal of loosely coupled services has been overlooked, and SOA's grand vision has been limited to implementing the API layer on top of monolithic systems rather than creating loosely coupled, reusable, and granular services [4], [8]. Therefore, the issue of adhering to SOA principles presents challenges to renewing software architecture and fulfilling its delayed promises [15]. Additionally, after implementing SOA practices, many researchers have recognized that the focus needs to shift to design-time issues to ensure that the architecture delivers on its promises.

### *Microservice Migration Issues*

Microservices Architecture (MSA) is regarded as a cloud-native architectural style emerging after SOA. MSA aims at arranging an application as a set of loosely-coupled services. MSA's promises in terms of independency and granularity remain as unsolved challenges as is the case with SOA. MSA drivers vary from scalability and maintainability to agility from early adopters to traditional companies. On the other hand, assessing adoption barriers are also important because the challenges play a critical role in the adoption rate in which some are related to the realization of the MSA promises, while many are associated with the transition process to MSA. As much as the transition process becomes clear and cost-effective, the adoption rate will be increased even for SMEs enterprises. Taibi et al. (2017) indicate ESB for microservice communication, shared libraries and shared relational databases as three factors that hinder MSA's independent microservices [18].

The Microservices Architecture (MSA) is considered a cloud-native architectural style that emerged after SOA [3]. MSA aims to organize an application into a set of loosely-coupled services, but its promises in terms of independence and granularity remain unsolved challenges, as with SOA [10]. MSA drivers vary from scalability and maintainability to agility, from early adopters to traditional companies. However, assessing adoption barriers is also important because the challenges play a critical role in the adoption rate. Some of these challenges are related to realizing MSA promises, while many are associated with the

transition process to MSA. If the transition process becomes clear and cost-effective, the adoption rate will increase, even for SMEs [22]. Taibi et al. (2017) identified ESB for microservice communication, shared libraries, and shared relational databases as three factors that hinder MSA's independent microservices [18].

### *ESB (Enterprise service bus)*

The necessity to comply with information technology trends such as cloud and Internet of Things (IoT) indicates that even the previous service orientation rationale is changing; for example, central governance by ESB was ideal in SOA, while complete decentralization in MSA is recommending dumb, and lightweight ESB [8][18]. Therefore, the emphasis is on decreasing central units like ESB and putting its duties inside microservices.

### *Shared Databases*

The dependency of microservices on a shared relational database is a challenging issue because dedicating a database per microservice requires a prerequisite in the software development and infrastructure domain [19]. Hence, the problem of the microservice' dependencies on other entities is still considered as an unsolved issue. It is associated with several number of embedded duties in each microservice to support communication, database, and quality issues that are interpreted in service boundaries determination.

### *Decomposition*

The primary activity in microservice migration is determining microservice boundaries, which is crucial in addressing all the associated issues [3]. An essential step in the migration process is the selection and analysis of input types. This information is then used in the decomposition process to determine the optimal boundaries for each microservice.

The process of decomposition is generally regarded as a manual and imprecise undertaking [23]. Breaking down a monolithic application into microservices can be a difficult task in the migration to MSA, as it depends on the expertise, domain knowledge, and experience of the individuals carrying it out [24]. The success of this process, therefore, is largely dependent on the proficiency of software architects. In practice, decomposition is frequently employed during critical deployment in migration operations [14].

### *Incremental decomposition*

Newman (2015) prefers incremental decomposition migration as it informs developers about microservices and mitigates large faults [25]. Knoche & Hasselbring (2019a) apply a decomposition strategy with the aim of incremental migration to support the maintainability and decrease deployment issues.

Further, in many cases, the gradual decomposition of a monolith is inevitable; thus prioritizing a microservice will be a must. The first step in decomposition is strategy selection. Major decomposition methods are based on business capabilities or domain driven design (DDD) [2].

### *Decomposition Tools*

Kalia et al. (2021) developed and examined a decomposition tool known as Mono2Micro for decomposing monolithic Java application. It leverages well-defined business use cases and runtime call relations to generate functionally cohesive partitioning of application classes. They found that Mono2Micro provides valuable assistance to practitioners in producing functionally explainable and cohesive microservice decompositions [1].

Gysel, et al. (2016) proposed a structured, repeatable approach to decompose services based on 16 novel coupling criteria (i.e., compatibility and constraints dimensions) related to

coupling. The compatibility set includes structure and content vitality, and constraints entails consistency and security constraints. Their proposed method relied on incremental and iterative way [26]. In a relevant study, Balalaei et al. (2016) have used DevOps practices to ease the incremental migration process and provided migration patterns with explanations in two sections: before and after the migration; however, they have not provided clear guidelines.

### *Inputs of Microservice Boundary Determination Methods*

The decomposition process results in boundaries for microservices, which can be seen as synonymous with the microservice identification process. Therefore, a successful decomposition process is a prerequisite for microservice identification and, ultimately, a successful migration.

Furthermore, existing research is divided in its perspective on MSA, between business-oriented or technical-oriented approaches, which can affect the migration toward microservices. This division impacts all architectural decisions, such as the selection of input types for microservice identification, decomposition strategies, granularity level, and dependencies of each microservice.

The current study first collected the inputs and criteria used in previous studies (see Table 1) and then discussed them with experts in the domain. The results showed that business-oriented inputs have been prioritized, specifically Domain-Driven Design (DDD), which has been applied by many recent studies for boundary determination because of its richness in domain knowledge [27]–[29].

**Table 1.** Microservice Boundary Identification Methods

Inputs	Orientation	Method, Criteria	Guide lines	Reference
Domain knowledge, UML	Business	Business functionality	◐	[25]
user-provided scenarios, service dependencies	Business	Automatic generation of a Service Dependency Graph for service-scenario and service-service	●	[30]
from DDD	Business	Focus on measurements of cohesion and coupling Coupling	◉	[28]
monolith modules	Technical	Best coupling for containerizing MSc	◉	[31]
modules dependency	Technical	Tracing data and function calls in run-time	●	[32]
Work load, Feature model	Technical	cluster features to find boundaries of best performance MSs	◉	[21]
Model-driven engineering and DDD	Business	Descriptive analysis	◐	[29]

Inputs	Orientation	Method, Criteria	Guide lines	Reference
Splitting services based on technical and organizational metrics	Business and Technical	Finding decision points in a migration process through 19 case studies	●	[33]
Source code	Business	Visual directed graph and AI partitioned MS recommender	○	[1]
DDD, Domain model	Business	Weighted Graph to cluster services by 16 coupling criteria	○	[26]

Furthermore, a review of the literature indicates that while some studies have provided guidelines [30], [32], [33], others have only partially presented guidelines [25], [29], and several studies lack guidelines altogether [1], [26], [28], [31].

### MSA and DevOps

Moreover, MSA is an emerging architecture that can accelerate performance in cloud systems [34]. The adoption of MSA is a newly emerging subject that has attracted both software engineering and cloud enthusiasts [35]. Microservices are the building blocks of new computing architecture that are mostly compatible with cloud elements. Many cloud-based applications have adopted microservices to achieve the benefits of service orientation at both the app and network levels. Recently, cloud-native applications have become a hype, leading to the emergence of new methods and platforms, as well as new needs such as fast software testing and deployment.

DevOps is a trend in the cloud that is defined as a set of practices combining software development and information technology operations. DevOps has been applied in many MSA practices and it was predicted that almost 80% of cloud software systems would be implemented according to MSA by the end of 2021 [36]. In addition, another report stated that the reason for MSA adoption in 47% of enterprises was that DevOps decreases the time between changing and publishing on the real system [37].

DevOps decreases the time between the creation of a new version of software and the deployment time in the target environment, which is considered as one of the cloud-native enablers. DevOps comprises a set of practices (such as continuous planning, deployment, integration, monitoring, and testing) to develop, test, and deploy software changes swiftly and reliably through creating strong collaboration between the software developers, testers and operators [3] [38].

Furthermore, one of MSA principles is API-based interaction between service-to-service or service-to-database instead of direct calls through sharing data structure, schema, and other details which force all developers to be aware of source code or database not owned by them; hence, API-based interaction makes independency of programmers becomes a reality, and all these MSA characteristics support DevOps “ideology” [39]. DevOps structure emphasizes on increasing communication and engagement of developers and network admins to realize its promises as well as stresses on the independency of each team [36], [40]–[42]. In the same way, MSA aims to achieve agility in the development and deployment process [9]. Therefore, this matching of goals indicates that there are similarities between MSA and DevOps.

Balalaei et al. (2018) emphasize on pairing the MSA and DevOps as a success factor for both of them [6].

However, applying a new architecture in software engineering in a DevOps context changes the structure and the roles of the team [43], system architecture [44], and infrastructure [45] where MSA migration imposes changes on both of them [3]. These changes will in turn affect DevOps agility; however, it should be resolved within MSA migration and after it. Fortunately, DevOps has shown its adaptability with new changes [46]. MSA and DevOps collaboration can address their common challenges like optimized containerization for automatic scaling and agility in the deployment of microservices, which could be achieved by DevOps CI (continuous integration) and CD (continuous deployment) [3], [47]. However, MSA migration practices in the DevOps context lacks guidelines and tools that could pose challenges [3]. Therefore, designing the right MS boundaries accelerates the deployment and leads to independency of MS that are common principles of DevOps and MSA.

### **3 Methodology**

This study aims to address challenges associated with migrating from a monolithic system architecture (MSA) to microservices architecture (MSA) by proposing a set of required phases and clarifying the entire migration process. The determination of the MS's boundaries is a critical activity in the migration process, and it should be supported by guidelines to ensure successful migration. To this end, the study conducted a case study and interviewed experts in the field. The case study was conducted between February 2020 and February 2021, and it was selected due to the possibility of a deep investigation and accessibility to team and system architecture details within the enterprise. Additional criteria for selection included the maturity of the DevOps team and the possibility of monitoring the migration activities. Furthermore, the study selected experts with experience and responsibility in the field of microservices and the DevOps team. The research relied on case study observations and interview analyses as evaluation tools. The study pursued the following research questions:

RQ1: What are the main challenges faced by practitioners and academics in the boundary determination process?

RQ2: How can we propose steps to optimize the boundary determination process that satisfies both business and technical aspects?

RQ3: What are the suitable guidelines for reducing ambiguity in identifying boundaries?

Interview

The rationale for conducting the interviews was to explore collaboration between the technical and business sides, identify challenges, gather input on accelerating microservices (MS) migration from both business and technical aspects, address concerns of business and DevOps team managers, evaluate the results, seek expert opinions on guidelines, and reach consensus on decisions related to MS migration [20], [28]. The interview data was gathered from experts in operational and academic domains. The interviews were conducted with six experts with different roles, including developer, product owner, infrastructure administrator, project and IT department manager, and DevOps team leader [26]. Each interview was carried out in two rounds of sessions, each lasting almost 60 to 90 minutes.

### **4 Case Study**

The study analyzed a learning management system (LMS) called SHAMIM. It is a comprehensive LMS that serves almost 120,000 students across the country, including all education processes from registration to presenting courses. Additionally, it offers functional services for assessment (quiz) and quality services such as security authorization and message management. During the Covid-19 era, its workload increased drastically, requiring agility

in terms of the development process and optimization of infrastructure operations. The case study was used as an evaluation tool, and the researchers conducted observations at the case study firm in a step-by-step process after executing each step of the boundaries identification process. The findings of each step were shared through interviews with experts, and the outcomes were analyzed. Subsequently, the results were analyzed by discussing them with experts in a focus group interview to collect complimentary comments.

### SHAMIM Description

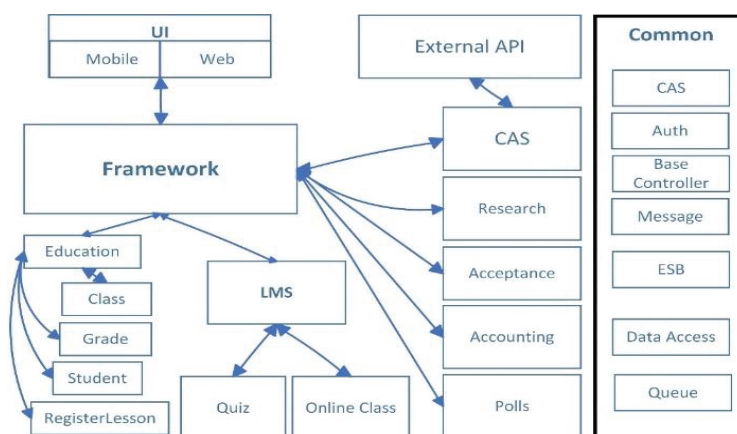
SHAMIM is an e-learning system that consists of tightly coupled apps, including a registration system, learning management system (LMS), accounting, research system, and a number of apps outside the university. Over time, the increasing demands for new features and interconnections with other apps have made the system larger and more complex.

During the Covid-19 pandemic, the SHAMIM system has become vital for education, and the quality factors such as reliability and stability have become crucial to avoid service failure. Additionally, the scalability of the system has been critical due to an increase in the number of learners, and new features have reinforced the monolith volume.

Various technologies in the frontend, backend, mobile app, virtualization, clustering database, etc., involve a large technical team in both programming and infrastructure that resist changes. The core of SHAMIM has been built based on the Yii framework, and the main app in the system is LMS, which manages and presents the learning files.

Furthermore, SHAMIM exchanges information with the ministry and other universities, stakeholders, and enterprises through the API layer. The UI layer supports the web and mobile apps, while the CAS provides central user authorization services with external APIs. Education, acceptance, research, polls, quiz, and virtual class comprise complementary SHAMIM subsystems.

Accounting apps provide functional requirements of the system in different domains. The common apps on the right side of Figure 1 provide the infrastructure and non-functional requirements of the SHAMIM. For example, ESB (enterprise service bus), as a logical communication layer, manages the data exchange between services or apps. Queue supports the data received from other enterprises through the API layer, and Authentication provides data access between SHAMIM subsystems.



**Fig. 1.** SHAMIM Monolith Architecture includes educational services and communication or security service named common (right)

## *Monolith Challenges*

Our initial observation of the existing monolith indicates that SHAMIM has faced several challenges over time, as highlighted by domain experts. These challenges include:

1) A high number of inevitable changes, including changes related to management opinions (50%), bug fixing (15%), and changes in software infrastructure due to a significant increase in the number of users (35%). The number of changes has been increasing, almost doubling from 2016 to 2021 compared to the same time period before 2016.

2) Rapid changes in software engineering methods, leading to changes in structure, workflow, and documentation, both in particular and in software management in general. This has led to different points of views among programmers and managers regarding structure, workflow, and documentation.

3) A greater emphasis on infrastructure performance during the Covid-19 pandemic, as online learning became the only way of learning. Consequently, experts recommended applying auto-scaling technologies, such as Docker, Kubernetes, message queue, etc., resulting in infrastructure changes. Addressing these challenges in MSA migration led to guidelines from researchers and domain experts based on practical experience.

## *Migration to MSA*

Migration to MSA is a challenging process that includes software and managerial issues [48]. Previous studies have emphasized the importance of adopting an incremental deployment approach and identifying microservices [25]. Our migration process for SHAMIM takes a business-oriented view while also leveraging technical assets, which has been shown to increase engagement among all stakeholders [33] and provides migration guidelines considering existing SOA architecture and reinforcing DevOps structure [15], [25]. Based on our analysis of previous studies, interview findings, and consideration of technical and business dimensions, we have identified the main steps in our migration process, starting with microservice identification. The first decision point in MSA migration is how to decompose the monolith to identify candidate microservices. According to literature review findings and expert comments [18],[44], incremental decomposition has been adopted to provide sufficient time for monitoring and analyzing the results of each phase. Determining the starting point in an incremental migration process is crucially important because the enterprise should derive the most benefit from the output of the first set of identified microservices. Some studies stress end-user-related services as the first emerging microservices so that the benefits of MSA could be realized [50].

In addition, to reach a consensus, some project-related employees were asked to rank SHAMIM's services according to their effectiveness. The results from the management and technical teams prioritized LMS, Quiz, CAS, and Authentication as they have been used more than other services.

## *Optimal Boundaries*

The issue of microservice identification is tightly coupled with the granularity level and the rationale behind boundary determination. Metrics for determining optimal and specific boundaries remain ambiguous in previous studies, which have proposed general criteria such as the number of lines of code, ability to rewrite the service in 2-6 weeks, small team size, etc. [21].

In the SHAMIM project, to achieve a convincing decision, we first selected Domain-Driven Design (DDD) elements, namely bounded context, domain events, and aggregation, to identify microservices that reinforce business-IT alignment. Based on information gathered from the monolith, UML class diagrams, and previous service-created

documentation, we identified the first candidates for microservice boundaries by measuring the coupling relations between monolith components from high-level documents. There was agreement among management and the technical team regarding the prioritization of services based on higher frequency calls. According to Vural et al. (2021), the interconnections were extracted from the UML diagram and DDD and then calculated as shown in Equation (1):

Let ‘C’ be the cohesion, ‘R’ be the number of relationships, and ‘N’ be the components inside a microservice candidate.

$$C = \frac{R + 1}{N} \quad (1)$$

In addition, the average cohesion is calculated as shown in equation (2) where n is the number of microservices or monolith’ components that are related to each other in an incremental manner.

$$Avg(C) = \frac{\sum_{i=1}^n C_i}{n} \quad (2)$$

Therefore, the cohesion of each candidate set has been calculated based on the extracted relations from UML class diagrams and the built documentation.

The first draft of microservice boundaries emerged after analyzing the DDD documents. However, there was no consensus among experts regarding microservice boundary determination, and consequently, it remains an open issue.

Guidelines for optimized boundaries:

In order to solve the ambiguity issue, during interviews with experts, two reasons were found behind this problem: the descriptive nature of DDD documents and lack of technical data on boundary determination. Thus, there was an emphasis on using technical data as complementary input to quantify the boundary determination problem. Hence, the findings of literature reviews and case study analyses, including findings from 12 interview sessions with experts in the domain, resulted in proposing the following guidelines as a step-by-step process:

Incremental migration has been followed, and we should adopt the business processes that will be involved in the first phase. There is a consensus on the 80-20 rule; hence, we have identified the most executed business processes based on counting calls to each business process and mapping them to corresponding services. Therefore, the list indicates the most used services that have priority in the business-oriented aspect. Then, project-related employee interviews have concluded that there are five business processes as the most used actions, namely: ‘Online Class’ (managing students on entering and using the virtual class app), ‘Register Lesson’ (managing lessons taken by students), ‘Quiz’ (providing an online exam environment), ‘Insert Grades’ (managing actions on grades), and ‘Roll Call’ (managing the presence process).

Although design-time documents such as UML, BPMN, etc. are more understandable, eliciting service dependencies from execution layer tools provides more details. A comprehensive study of the execution layer reveals two types of dependencies: dependencies between services and dependencies between table data in the database(s) that lead to extracting the coupling degree from both sources. In the SHAMIM monolith, the critical existing services under ‘Education app’ are namely, ‘Student’, ‘RegisterLesson’, ‘Class’, and ‘Grade’ (Refer to Figure 1). As the first source, we should find a lightweight solution for calculating the number of dependencies between services. To achieve the coupling degree between services, we have defined a parent constructor that logs each call of its child classes. In each child class, we called the parent constructor; therefore, we collected the number of calls and the name of the caller in the log table, which determines the intensity of dependencies between services. The call log has been gathered in an isolated environment that only permits a specific business process and its services to be executed.

```

public function __construct($config = [])
{
    /***** Log request *****/
    Log( get_called_class() );
    /*****/
}
    
```

**Fig. 2.** Parent constructor code to count the calls for each class

Hence, the afferent and efferent coupling for each existing service on execution time has been calculated from the log table which can reveal the service’s dependencies per execution of each business process. As a sample, Figure 3 demonstrates the classes that have been called when the “Register Lesson” business process is executed.

log_time	class_name
11:16:24	RegisterLesson
11:16:24	Student
11:16:25	RegisterLesson
11:16:25	Student
11:16:25	Quiz
11:16:25	Student
11:16:25	Class
11:16:25	ClassOnline
11:16:25	Student
11:16:25	Student
11:16:25	RegisterLesson
11:16:25	RegisterLesson

**Fig. 3.** List of services can be called when “Register Lesson” business process has executed

Subsequently, we named the most used business processes in SHAMIM as follows: A. Registered Lesson, B. Online Class, C. Quiz, D. Insert Grade, and E. RollCall. By repeating the above process for each target business process, in a collective view, a table of dependency is achieved (Table 2).

**Table 2.** Collective view of dependencies between services regarding execution of target business processes (A to E)

Monolith’s App name	Existing Service	Registered	Online Class	Quiz	Insert Grade	Roll Call
Education	Student	*	*	*	*	*
	RegisterLesson	*			*	*
	Class	*				
	Grade				*	
LMS	Quiz	*		*		*
	OnlineClass	*			*	*

Table 2 illustrates the existing services under the monolithic application that were called during the execution of a business process (in this case, processes A, B, C, D, and E). Based on the dependencies between services, it can be seen from Table 2 that ‘Student’, ‘RegisterLesson’, ‘Quiz’, and ‘OnlineClass’ could potentially be grouped together to form a

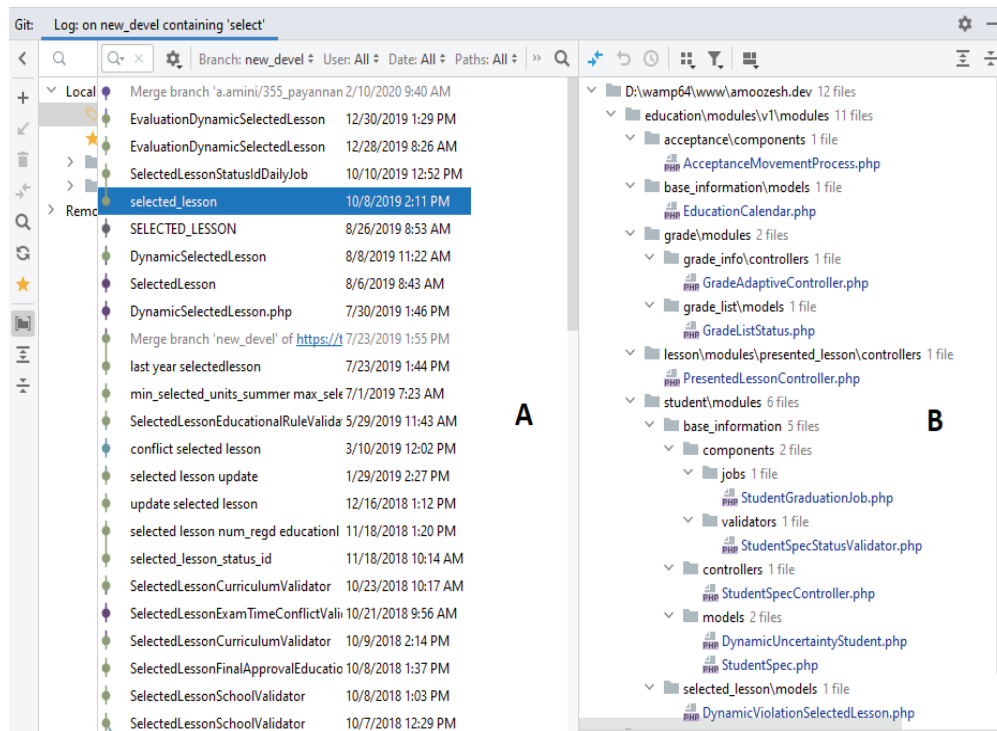
candidate microservice, while ‘Class’ and ‘Grade’ could form a separate microservice due to their fewer associations with other services during execution.

The database is another source of implicit knowledge about data aspects that can be helpful in recognizing similarities in data behavior within execution time. Each service consists of code and database tables. The idea is to monitor the data behavior for each business process (BP) execution and analyze the coupling degree between services from the database aspect. We activated the log, which is supported by most DBMSs in Mysql, to save all transactions. The data collection was done in the isolated environment which only permits a specific business process and its services to be executed. Table 3 shows the result of monitoring data access for business processes A through E during execution time. The results indicate that ‘tbl-student’, ‘tbl-RegisterLesson’, and ‘tbl-Class-List’ have similar data access patterns, which suggests high coupling between those tables. No meaningful similarity was observed for other database actions in the SHAMIM monolith.

**Table 3.** Data access within execution time of each business processes (A to E)

Monolith’s App name	Existing Service	Registered Lesson	Online Class	Quiz	Insert Grade	RollCall
Education	Student	*	*	*	*	*
	RegisterLesson	*			*	*
	Class	*				
	Grade				*	
LMS	Quiz	*		*		*
	OnlineClass	*			*	*

Changes in apps provide implicit knowledge about the reasons behind each change in services and their effects on other services. This can reveal a map of hidden cause and effect between services for each change. Structured and reliable information about changes can be obtained from source management tools such as GitHub. Thanks to the DevOps team, Git has been adopted as a DevOps tool for source control. Git indicates the dependent services in each change, making it more reliable than design documents like UML because it is the coordinator of any code change. For each change, or 'commit' in Git language, a list of affected services can be identified, indicating the affected files per change as shown in Figure 4. This information is valuable for finding tightly coupled services that have changed together per specific commit, which is a crucial factor in grouping the existing services in a candidate MS when each business process is modified. Therefore, knowing the list of changed services when each business process is modified and finding the tightly coupled services that have changed together per specific commit are essential in grouping the existing services in a candidate MS.



**Fig. 4.** View of Git app (A) shows the commits (changes) list and (B) shows affected files per commit

Therefore, in collective view, table of dependencies measured in percentage is achieved (Table 4). It indicates coupling of ‘student’ and ‘RegisterLesson’ together and there was no strong reason to bundle other services based on changelog tracking.

**Table 4.** Tracking BP's changes effects percentage on each existing services

Monolith's App name	Existing Service	A	B	C	D	E
Education	Student	80	60	60	50	50
	RegisterLesson	90	70	30	30	80
	Class	40	25	25	0	0
	Grade	0	0	0	100	0
LMS	Quiz	20	0	90	15	20
	OnlineClass	20	80	0	0	10

Based on the results of the four-step process described above, the microservice boundary determination roadmap became clear and reasonable. The identified microservice candidates are 'Student' and 'RegisterLesson' as a consolidated candidate, and 'Class', 'Grade', 'Quiz', and 'Online class' as separate candidates.

In software development, the standard approach is to use a framework to develop and deploy applications while ensuring quality measures such as reusability, security, and agility. However, in the context of MSA, an embedded framework is necessary to maintain the benefits of a framework and ensure the independence of each service. Because heavyweight frameworks can be cumbersome, there has been a movement towards lightweight frameworks that are more adaptable to the microservice atmosphere. For example, Lumen is a micro framework introduced for Laravel, and Yii2 has been proposed as a micro version of the Yii framework used in the SHAMIM monolith. Therefore, framework-based monoliths should migrate to a microframework to allow the use of a micro framework in addition to any microservice.

In the monolithic architecture, the ESB plays a central role in controlling and governing service messaging as an active and intelligent system that decreases service independence by

enforcing its rules. In the microservice architecture, the ESB should be replaced by a lightweight and simple messaging layer [11]. We have chosen to use Apache Kafka as a lightweight messaging system that does not enforce any specific format.

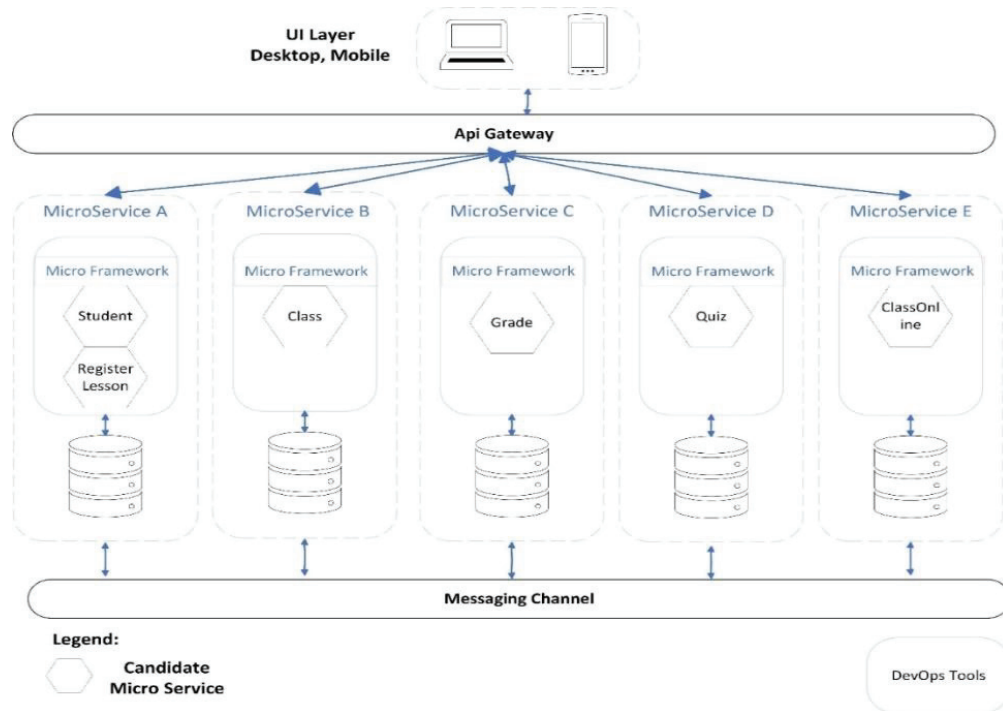


Fig. 5. New microservice-based architecture of SHAMIM

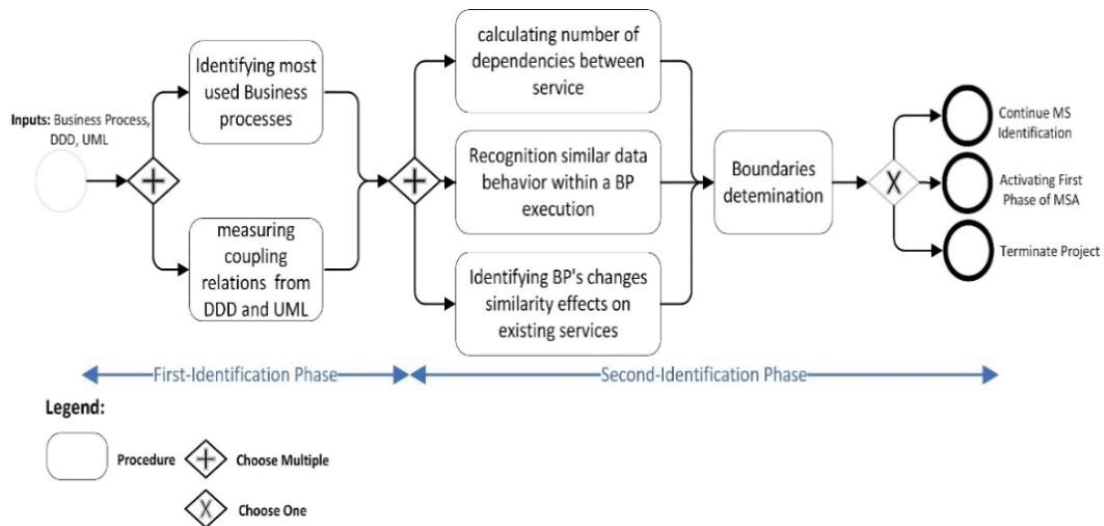
Following the guidelines outlined above, Figure 5 illustrates the new microservice-based architecture of SHAMIM. The architecture identifies microservices in the first phase from the most commonly used business processes and can be incrementally expanded.

## 5 RESULTS

### *Migration strategies*

Successful migration to MSA requires expertise and consultancy to ensure a solid foundation for the new architecture. However, the migration process is closely tied to the internal logic and knowledge of the systems, as determining the appropriate boundaries for microservices is critical [2]. The MSA migration strategy depends on the business or technical orientation views, which can affect input types, methods, and results. Selection between input types and methods should be made based on the project goals and firm circumstances. This paper focuses on the business orientation and provides migration guidelines in a DevOps context.

Our study findings demonstrate that migration to MSA is a critical and challenging process that involves a variety of roles compared to SOA. Successful MSA migration depends on clear guidelines that address ambiguity and help in decision points during the migration process. Drawing upon practitioner and academic perspectives, we have focused on the determination of microservice boundaries as the first major challenge. We have proposed systematic guidelines for identifying microservices based on a case study and recommendations from technical and business experts. Figure 6 illustrates the proposed process for identifying candidate microservices.



**Fig. 6.** Boundaries determination in proposed MSA migration steps including first phase (for business related inputs) and second phase (for technical related inputs)

### To answer the first RQ1

After analyzing the data, we identified three challenges. The first challenge is to determine the input types that provide suitable background information to support clear decisions and satisfy both the business and technical teams involved in the migration process. For the first identification phase, we selected Domain-Driven Design (DDD) documents and Unified Modeling Language (UML) use case diagrams, while for the second identification phase, we used call logs between services, table data dependencies per execution, and changelogs. These inputs include both high-level and low-level abstractions, addressing the concerns of both business and technical teams. Even low-level abstraction input types such as call logs between services are rooted in the business process, making it a valuable business entity. The combination of these input types reduces the ambiguity problem and provides more precise results for decision-making, such as determining boundaries.

The second challenge is to modify the structures to achieve successful practice in MSA migration. One of the principles of MSA is to reinforce the independence of microservices, which conflicts with components such as ESB and software framework. ESB is widely used in SOA-based architecture, while software frameworks are commonly used in software development. To address these issues, we have proposed alternative solutions. Firstly, we replace ESB by assuring the quality inside each microservice and by applying a light messaging system. Secondly, to solve the framework reference problem, we have utilized the micro-framework concept in the proposed architecture alongside each microservice.

### Answer to the second RQ2

The third challenge is related to the role of DevOps structure in MSA migration. DevOps structure causes close and systematic relations between developers and infrastructure team that supports the migration process and agile interaction between teams in deploying MSA. In addition, DevOps focuses on the documentation of technical events which is productive. In our case study before DevOps implementation, the documentations mainly belonged to high-level or design-time documentation; however, after DevOps deployment, the execution-time documentation systematically increased and was supported by DevOps tools. We have utilized a change log from Git to extract services' dependencies per change that are reliable and update since it is a must in DevOps culture. We have proposed a method to monitor and cluster changes according to the used source (Git) to be a pillar in MS candidate identification.

### ***Answer to the Third RQ3***

Based on the findings of the study, we proposed a set of guidelines to decrease ambiguity and provide a roadmap for practitioners. The first guideline describes step by step how to utilize the inputs in MSA migration: first, achieving the relation coupling from high-level input types namely, DDD and UML use case diagram, second, prioritizing business processes to be involved in phased, incremental process, and third, preparing and applying low-level abstraction input types including service dependencies and data dependencies to determine the optimized boundaries. Furthermore, there were other guidelines to support the new architecture in general: a) how to tightly bind the business process concept as long as the migration project progresses, and how to track the BP's services or BP's data resulting in microservices which are clearly associated with BPs domain. Another guideline involves the culture that should be focused on in a successful MSA such as, IT-business members and developers-infrastructure's admins with defined structured relations. This is because MSA issues are multi-dimensional; for example, optimized containerization as an MS infrastructure subject requires that changes be made by developers in their commits.

## **6 Conclusion**

The implementation of MSA in combination with DevOps in a business-oriented setting can help overcome the disadvantages of monolithic architecture by providing independent microservices that can autonomously develop and deploy. However, there are numerous challenges in implementing such an architecture. Clear guidelines that consider both business and technical assets and leverage the strengths of DevOps can facilitate the migration to MSA and address the obstacles that arise.

The objectives of this study were threefold: to identify the main challenges faced by practitioners and academics in migrating to MSA, to identify the structures of DevOps that support MSA migration, and to propose suitable guidelines to increase the performance of MSA in a DevOps firm. To achieve these objectives, we conducted a comprehensive literature review, interviewed six experts in the field, and carried out a case study involving an LMS called SHAMIM in a DevOps context.

We found that the most challenging issue in migrating to MSA is the identification of microservice boundaries when extracting them from a monolith [11], [15]. Moreover, while the business orientation of MSA has received considerable attention, business-oriented methods have faced more ambiguity in boundary identification [28]. Therefore, there is a need for a method that considers both business and technical concerns. To address this issue, we have presented a step-by-step guideline that includes both business- and technical-oriented input types, with a focus on the business orientation of the identified microservices.

Our review of the literature and the case study results showed that DevOps structures, tools, and documents positively affect MSA migration. Many studies report that DevOps success implementation depends on changing the team culture, such as changing methodologies, existing activities, or the hierarchy structure of the team, or fostering collaboration between team members to enhance DevOps success [3]. Similarly, MSA migration benefits from systematic collaboration not only among developers but also between development, infrastructure, and business teams. Thanks to the DevOps team, the full usage of Git as a DevOps tool for source control has been established, supporting the presented method by providing a reliable and updated list of changes. Therefore, migration to MSA is facilitated in a DevOps context because the culture of collaboration between teams has evolved. DevOps can thus be considered a prerequisite for a successful migration to MSA.

In microservices, the team should be cross-functional and multi-task. Even programmers should have a light knowledge of infrastructure operation to improve collaboration and agility in the DevOps context. The team members reported that they have saved more time because

bug fixing time and challenges with the operation team have been reduced, allowing them to spend more time on developing new features or addressing software quality aspects.

Our contribution to the field is threefold: (1) we have described an e-learning/LMS system as an example of MSA migration in a frequently used monolith, (2) we have identified and explained the advantages and obstacles faced by practitioners of a DevOps team in a microservice migration, and (3) we have proposed a set of guidelines that consider both business and technical domains, utilizing input types and methods attributed to both contexts and satisfying experts in both domains.

### **Implication**

We selected Moodle1 as our LMS in the SHAMIM case study, representing one of the most promising open-source LMS worldwide [51]. Since Covid-19 has contributed to higher usage of LMSs, their principal role makes scalability a necessity which can be achieved by increasing Moodle adaptability with promising architectures such MSA on DevOps to increase its performance, scalability, and agility. Our guidelines and case studies have some implications for practitioners and academic domains.

- Education organizations can practically benefit from a case study finding showing how monolith can be migrated to MSA that is based on Moodle as one of the most used LMSs.
- The selected incremental migration steps facilitate the enterprises to apply migration without undergoing serious downtime which is an important factor in LMS systems.
- The obtained guidelines identified useful assets and activities in the DevOps team regarding MSA migration.
- This study increases engagement of technical team presenting the context for justification of boundaries determination in a business-oriented method by involving technical assets in the MS identification process.

### **Limitation**

The study has come up with a set of guidelines to smoothly migrate from monolith to MSA in the DevOps context which is an ever-growing demand in a cloud-based environment. The study finding is promising; however, work remains to be done on both conceptual/research and engineering/implementation levels. For example, the CI/CD DevOps automation needs more in addressing MSA principles in independent containerization to automate from microservice to the infrastructure. One of the major problems in the decomposition of the monolith into several microservices in DevOps infrastructure is the interconnection of servers and preserving independency of microservices which could be an attractive subject for further study.

### **References**

1. A. K. Kalia et al., “Mono2Micro: An AI-Based Toolchain for Evolving Monolithic Enterprise Applications to a Microservice Architecture,” in Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020, pp. 1606–1610, doi: 10.1145/3368089.3417933.
2. C. Richardson, *Microservices Patterns: With examples in Java*. Manning Publications, 2018.
3. M. Waseem, P. Liang, and M. Shahin, “A Systematic Mapping Study on Microservices Architecture in DevOps,” *J. Syst. Softw.*, vol. **170**, p. 110798, Dec. 2020, doi: 10.1016/j.jss.2020.110798.
4. S. Newman, *Monolith to Microservices [Book]*. O’Reilly Media, Inc., 2019.

5. X. Larrucea, I. Santamaria, R. Colomo-Palacios, and C. Ebert, “Microservices,” *IEEE Softw.*, vol. **35**, no. 3, pp. 96–100, 2018, doi: 10.1109/MS.2018.2141030.
6. A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture,” *IEEE Softw.*, vol. **33**, no. 3, pp. 42–52, 2016, doi: 10.1109/MS.2016.64.
7. M. S. Hamzehloui, S. Sahibuddin, and A. Ashabi, “A study on the most prominent areas of research in microservices,” *Int. J. Mach. Learn. Comput.*, vol. **9**, no. 2, 2019.
8. K. Indrasiri and P. Siriwardena, *Microservices for the Enterprise: Designing, Developing, and Deploying*. 2018.
9. G. Marquez, C. Taramasco, H. Astudillo, V. Zalc, and D. Istrate, “Involving Stakeholders in the Implementation of Microservice-Based Systems: A Case Study in an Ambient-Assisted Living System,” *IEEE Access*, vol. **9**, pp. 9411–9428, 2021, doi: 10.1109/ACCESS.2021.3049444.
10. R. Janeček, “Service Oriented Architecture Pitfalls,” in *International Conference on Current Trends in Theory and Practice of Computer Science*, 2009, pp. 37–45.
11. P. S. Kasun Indrasiri, *Microservices for the Enterprise: Designing, Developing, and Deploying* [Book]. Apress, 2018.
12. S. Taherizadeh and M. Grobelnik, “Key influencing factors of the Kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications,” *Adv. Eng. Softw.*, vol. **140**, p. 102734, 2020, doi: <https://doi.org/10.1016/j.advengsoft.2019.102734>.
13. T. Kohlborn, A. Korthaus, T. Chan, and M. Rosemann, “Identification and Analysis of Business and Software Services—A Consolidated Approach,” *IEEE Trans. Serv. Comput.*, vol. **2**, 2009, doi: 10.1109/TSC.2009.6.
14. M. Mazzara, N. Dragoni, A. Bucchiarone, A. Giaretta, S. T. Larsen, and S. Dustdar, “Microservices: Migration of a Mission Critical System,” *IEEE Trans. Serv. Comput.*, vol. **1374**, no. c, pp. 1–14, 2018, doi: 10.1109/TSC.2018.2889087.
15. J. Fritzs, J. Bogner, A. Zimmermann, and S. Wagner, “From Monolith to Microservices: A Classification of Refactoring Approaches,” in *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, 2019, pp. 128–141.
16. J. Bogner, J. Fritzs, S. Wagner, and A. Zimmermann, “Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality,” *Proc. - 2019 IEEE Int. Conf. Softw. Archit. - Companion, ICSA-C 2019*, pp. 187–195, 2019, doi: 10.1109/ICSA-C.2019.00041.
17. C. Bühler, “Microservices in a DevOps Context,” 2021.
18. D. Taibi and V. Lenarduzzi, “On the Definition of Microservice Bad Smells,” *IEEE Softw.*, vol. **35**, no. 3, pp. 56–62, May 2018, doi: 10.1109/MS.2018.2141031.
19. J. Bogard, “Avoiding Microservice Megadisasters - Jimmy Bogard - YouTube,” 2021.
20. D. Taibi, V. Lenarduzzi, and C. Pahl, “Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation,” *IEEE Cloud Comput.*, vol. **4**, no. 5, pp. 22–32, Sep. 2017, doi: 10.1109/MCC.2017.4250931.
21. S. Klock, J. M. E. M. van der Werf, J. P. Guelen, and S. Jansen, “Workload-Based Clustering of Coherent Feature Sets in Microservice Architectures,” in *2017 IEEE International Conference on Software Architecture (ICSA)*, 2017, pp. 11–20, doi: 10.1109/ICSA.2017.38.
22. F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi, “From monolithic systems to Microservices: An assessment framework,” *Inf. Softw. Technol.*, vol. **137**, p. 106600, Sep. 2021, doi: <https://doi.org/10.1016/j.infsof.2021.106600>.
23. G. Kecskemeti, A. C. Marosi, and A. Kertesz, “The ENTICE approach to decompose monolithic services into microservices,” in *2016 International Conference on High*

- Performance Computing Simulation (HPCS), 2016, pp. 591–596, doi: 10.1109/HPCSim.2016.7568389.
24. D. Taibi and K. Systä, “From Monolithic Systems to Microservices: A Decomposition Framework based on Process Mining,” 2019, doi: 10.5220/0007755901530164.
  25. S. Newman, *Building Microservices*. O’Reilly Media, 2015.
  26. [M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann, “Service Cutter: A Systematic Approach to Service Decomposition,” in *Service-Oriented and Cloud Computing*, 2016, pp. 185–200.
  27. F. Rademacher, J. Sorgalla, and S. Sachweh, “Challenges of Domain-Driven Microservice Design: A Model-Driven Perspective,” *IEEE Softw.*, vol. **35**, no. 3, pp. 36–43, 2018, doi: 10.1109/MS.2018.2141028.
  28. H. Vural and M. Koyuncu, “Does Domain-Driven Design Lead to Finding the Optimal Modularity of a Microservice?,” *IEEE Access*, vol. **9**, pp. 32721–32733, 2021, doi: 10.1109/ACCESS.2021.3060895.
  29. R. A. Schmidt and M. Thiry, “Microservices identification strategies : A review focused on Model-Driven Engineering and Domain Driven Design approaches,” in *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*, 2020, pp. 1–6, doi: 10.23919/CISTI49556.2020.9141150.
  30. S.-P. Ma, C.-Y. Fan, Y. Chuang, I.-H. Liu, and C.-W. Lan, “Graph-based and scenario-driven microservice analysis, retrieval, and testing,” *Futur. Gener. Comput. Syst.*, vol. **100**, pp. 724–735, 2019, doi: <https://doi.org/10.1016/j.future.2019.05.048>.
  31. S. Sarkar, G. Vashi, and P. P. Abdulla, “Towards Transforming an Industrial Automation System from Monolithic to Microservices,” in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2018, vol. **1**, pp. 1256–1259, doi: 10.1109/ETFA.2018.8502567.
  32. Z. Ren et al., “Migrating Web Applications from Monolithic Structure to Microservices Architecture,” 2018, doi: 10.1145/3275219.3275230.
  33. R. H. Hamdy, M. Ayas, Philipp Leitner, “Facing the Giant: a Grounded Theory Study of Decision-Making in Microservices Migrationse,” 2021, [Online]. Available: <https://arxiv.org/abs/2104.00390v2>.
  34. A. Furda, C. Fidge, O. Zimmermann, W. Kelly, and A. Barros, “Migrating Enterprise Legacy Source Code to Microservices: On Multitenancy, Statefulness, and Data Consistency,” *IEEE Softw.*, vol. **35**, no. 3, pp. 63–72, 2018, doi: 10.1109/MS.2017.440134612.
  35. G. Liu, B. Huang, Z. Liang, M. Qin, H. Zhou, and Z. Li, “Microservices: Architecture, container, and challenges,” *Proc. - Companion 2020 IEEE 20th Int. Conf. Softw. Qual. Reliab. Secur. QRS-C 2020*, pp. 629–635, 2020, doi: 10.1109/QRS-C51114.2020.00107.
  36. C. Pahl and P. Jamshidi, “Microservices: A systematic mapping study,” in *CLOSER 2016 - Proceedings of the 6th International Conference on Cloud Computing and Services Science*, 2016, vol. **1**, pp. 137–146, doi: 10.5220/0005785501370146.
  37. “Global Microservices Trends Report,” 2018. Accessed: Jul. 01, 2021. [Online]. Available: <https://go.lightstep.com/global-microservices-trends-report-2018>.
  38. B. T. Klein, G. Giese, J. Lane, J. G. Miner, J. J. Jones, and O. Venezuela, “An Approach to DevOps and Microservices.,” 2020. doi: 10.2172/1635752.
  39. D. Trihinas, A. Tryfonos, M. D. Dikaiakos, and G. Pallis, “DevOps as a Service: Pushing the Boundaries of Microservice Adoption,” *IEEE Internet Comput.*, vol. **22**, no. 3, pp. 65–71, 2018, doi: 10.1109/MIC.2018.032501519.
  40. N. C. Mendonca, P. Jamshidi, D. Garlan, and C. Pahl, “Developing Self-Adaptive Microservice Systems: Challenges and Directions,” *IEEE Softw.*, pp. 1–7, 2019, doi: 10.1109/MS.2019.2955937.

41. I. Ozkaya, “Are DevOps and Automation Our Next Silver Bullet?,” *IEEE Softw.*, vol. **36**, no. 4, pp. 3–95, 2019, doi: 10.1109/MS.2019.2910943.
42. L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect’s Perspective*, 1st ed. Addison-Wesley Professional, 2015.
43. K. Nybom, J. Smeds, and I. Porres, “On the Impact of Mixing Responsibilities Between Devs and Ops,” in *Agile Processes, in Software Engineering, and Extreme Programming*, 2016, pp. 131–143.
44. L. E. Lwakatare et al., “DevOps in practice: A multiple case study of five companies,” *Inf. Softw. Technol.*, vol. **114**, pp. 217–230, 2019, doi: <https://doi.org/10.1016/j.infsof.2019.06.010>.
45. B. Familiar and J. Barnes, “DevOps Using PowerShell, ARM, and VSTS,” in *Business in Real-Time Using Azure IoT and Cortana Intelligence Suite: Driving Your Digital Transformation*, Berkeley, CA: Apress, 2017, pp. 21–93.
46. T. Laukkarinen, K. Kuusinen, and T. Mikkonen, “Regulated software meets DevOps,” *Inf. Softw. Technol.*, vol. **97**, pp. 176–178, 2018, doi: <https://doi.org/10.1016/j.infsof.2018.01.011>.
47. L. Baresi, M. Garriga, and A. De Renzis, “Microservices Identification Through Interface Analysis,” in *Service-Oriented and Cloud Computing*, 2017, pp. 19–33.
48. F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi, “From monolithic systems to Microservices: An assessment framework,” *Inf. Softw. Technol.*, vol. **137**, p. 106600, 2021, doi: <https://doi.org/10.1016/j.infsof.2021.106600>.
49. M. Lin, J. Xi, W. Bai, and J. Wu, “Ant Colony Algorithm for Multi-Objective Optimization of Container-Based Microservice Scheduling in Cloud,” *IEEE Access*, vol. **7**, pp. 83088–83100, 2019, doi: 10.1109/ACCESS.2019.2924414.
50. H. Knoche and W. Hasselbring, “Drivers and Barriers for Microservice Adoption-A Survey among Professionals in Germany 1 Drivers and Barriers for Microservice Adoption-A Survey among Professionals in Germany,” *Enterp. Model. Inf. Syst. Archit. - Int. J. Concept. Model.*, vol. **14**, no. 1, pp. 1–35, 2019, doi: 10.18417/emisa.14.1.
51. A. I. Abdula, H. A. Baluta, N. P. Kozachenko, and D. A. Kassim, “Peculiarities of using of the Moodle test tools in philosophy teaching,” 2020.