IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 24, 2022, accepted February 7, 2022, date of publication March 10, 2022, date of current version April 12, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3158648

Accumulation and Prioritization of Architectural Debt in Three Companies Migrating to Microservices

SAULO SOARES DE TOLEDO[1], ANTONIO MARTINI[1], (Member, IEEE), PHU H. NGUYEN[2], AND DAG I. K. SJØBERG[1], (Member, IEEE)

[1]Department for Informatics, University of Oslo, 0316 Oslo, Norway
[2]SINTEF, 0373 Oslo, Norway

Corresponding author: Saulo Soares de Toledo (saulos@ifi.uio.no)

ABSTRACT Many companies migrate to microservices because they help deliver value to customers quickly and continuously. However, like any architectural style, microservices are prone to architectural technical debt (ATD), which can be costly if the debts are not timely identified, avoided, or removed. During the early stages of migration, microservice-specific ATDs (MS-ATDs) may accumulate. For example, practitioners may decide to continue using poorly defined APIs in microservices while attempting to maintain compatibility with old functionalities. The riskiest MS-ATDs must be prioritized. Nevertheless, there is limited research regarding the prioritization of MS-ATDs in companies migrating to microservices. This study aims to identify, during migration, which MS-ATDs occur, are the most severe, and are the most challenging to solve. In addition, we propose a way to prioritize these debts. We conducted a multiple exploratory case study of three large companies that were early in the migration process to microservices. We interviewed 47 practitioners with several roles to identify the debts in their contexts. We report the MS-ATDs detected during migration, the MS-ATDs that practitioners estimate to occur in the future, and the MS-ATDs that practitioners report as difficult to solve. We discuss the results in the context of the companies involved in this study. In addition, we used a risk assessment approach to propose a way for prioritizing MS-ATDs. Practitioners from other organizations and researchers may use this approach to provide rankings to help identify and prioritize which MS-ATDs should be avoided or solved in their contexts.

INDEX TERMS Architectural technical debt, microservices, software maintainability, cross-company study, qualitative analysis.

I. INTRODUCTION

When companies migrate their software towards a microservice architecture, the software is split into a small set of independent services. Many of the practical difficulties encountered in previous architectures can be mitigated by using microservices. They support small and frequent releases, improve scalability, and promote independence among teams. However, microservices also bring new management and technical demands, such as the need to be business-domain driven and understand distributed systems [1].

Like any architectural style, the microservice architecture is prone to architectural technical debt (ATD), which may incur high costs [2]. ATD is a type of technical debt (TD) consisting of sub-optimal architectural solutions, which deliver benefits in the short-term but increase overall costs in the long run [3]. Some ATDs are specific to microservices [2] and might not be considered as problems in other architectures. For example, microservices should communicate through a ''dumb pipe'' (i.e., there should not exist any transformation logic between the services), while in previous Service Oriented Architectures (SOA), a common approach is to have some logic between services to transform data. In this paper, we consider ATDs only in the context of microservices applications and thus name them MS-ATDs.

One of the reasons for companies to migrate to microservices is repaying known ATDs from their previous architectures while, at the same time, obtain the benefits of this new architectural style. Figure 1 exemplifies such a migration:

The associate editor coordinating the review of this manuscript and approving it for publication was Giuseppe Destefanis.

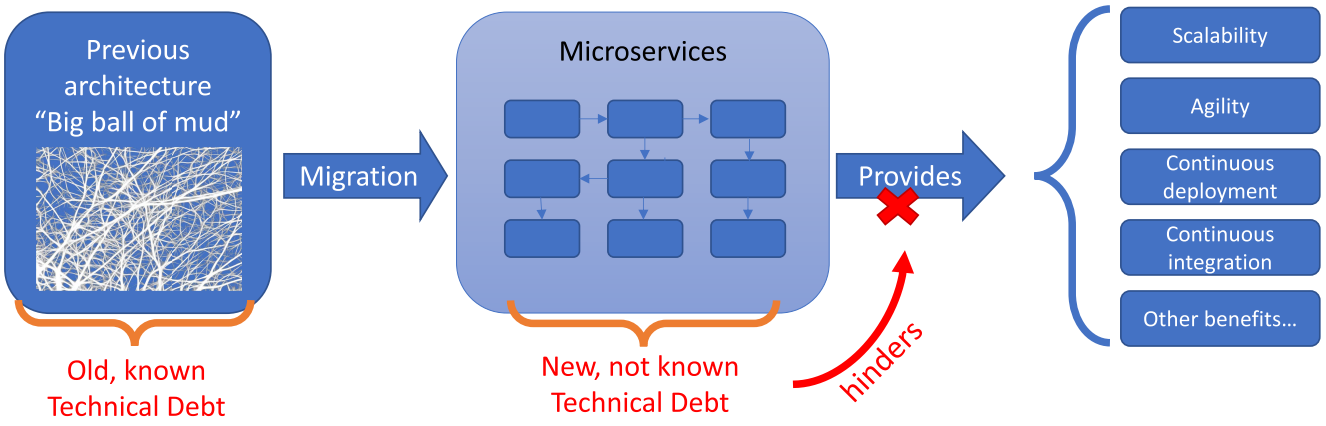


FIGURE 1. New ATD found after the migration to microservices may hinder the benefits of the new architecture.

a company has a "Big Ball of Mud" architecture [4] and repays ATDs in a migration to microservices. As the company believes that most ATDs from before were paid, the new microservice architecture can be prone to new, unknown MS-ATDs. The new microservice architecture is expected to have better scalability and agility, allow enhanced continuous integration and delivery pipelines, and provide many other benefits, such as allowing independent teams to work in parallel, having more testable code, and better control of costs in the cloud [1], [5]. However, the new MS-ATDs reduce these benefits and can be more costly than previous debts.

Migration to microservices has been investigated in previous studies from different perspectives. Several authors have proposed tools and approaches for assisting migration to microservices; see the systematic mapping by Bushong *et al.* [6]. A few other studies have investigated TD and related concepts in this new architectural style [2], [7], [8]. However, none of these studies have covered how ATD has accumulated during migration from a prior architecture to microservices. A lightweight survey of the current literature looking for the terms "microservices," "micro-services," "prioritization," and "technical debt" in some of the major research databases (ACM Digital Library,[1] IEEE Xplore,[2] Scopus[3]) highlighted that there are no relevant papers on how to prioritize MS-ATDs. Existing secondary studies on microservices [9]–[11] do not address prioritizing MS-ATDs. Companies must address the costs of such debts at a later stage of migration. Furthermore, after observing ATDs from the old architecture being repaid during migration, practitioners might have a false impression that the project is going well without noticing new MS-ATDs.

During migration to a microservice architecture, practitioners have the opportunity to identify MS-ATDs in a timely manner before they become widespread in the entire architecture. Knowing the risky and costly MS-ATDs facilitates practitioners in deciding which MS-ATDs to avoid, remove, and prioritize repayment. This study investigates the following research questions (RQs) in companies that have started their migration to microservices:

- **RQ1:** Which MS-ATDs do companies encounter during early migration to microservices?
- **RQ2:** Which MS-ATDs do companies foresee in the future of the migration?
- **RQ3:** Which MS-ATDs do the practitioners find difficult to solve?
- **RQ4:** How important do practitioners perceive MS-ATDs?
- **RQ5:** How can companies prioritize which MS-ATDs to avoid or repay?

To answer these questions, we conducted a multiple case study of three companies in the early stages of migration to microservices. As shown in Figure 2, we investigated the present and future stages of migration to microservices in these companies. RQ1 aims to identify the debts that occur in the early stages of migration (*Present* in Figure 2) to microservices: TD is often introduced early and persists throughout the software life cycle [12]. RQ2 investigates the debts estimated to occur in the future, helping practitioners to avoid or mitigate them (*Future* in Figure 2). Answering RQ3 highlights MS-ATDs that are difficult to remove and thus may either require more effort to be repaid or remain in the system for a long time. Answering RQ4 highlights which MS-ATDs practitioners consider risky (important to them) and, thus, should be prioritized for removal or mitigation. The difficulty and the risk of the MS-ATDs affect how practitioners prioritize them. Finally, RQ5 investigates how companies can prioritize the removal or mitigation of MS-ATDs identified and discussed in the previous questions (see Figure 2). Prioritizing MS-ATDs is an important management activity [13].

The remainder of this paper is organized as follows. Section II provides the background for this study. Section III describes our research design. Section IV presents our results and discussion as well as implications for research and

[1] https://dl.acm.org/
[2] https://ieeexplore.ieee.org/
[3] https://www.scopus.com/

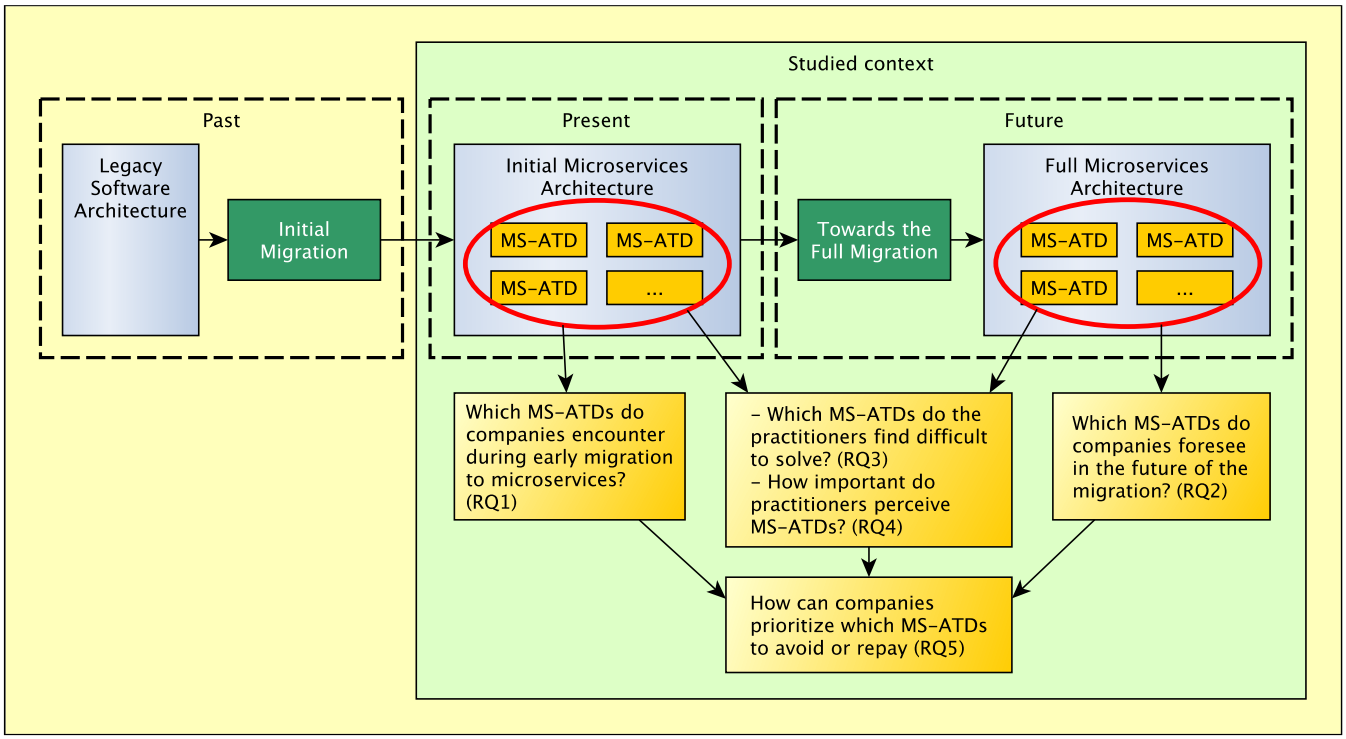


FIGURE 2. Relationship between the research questions, the ongoing architecture (with the migration in progress), and the final microservice architecture (as envisioned by the practitioners).

practice. Section V discusses the limitations of this study. Section VI presents the related work. Section VII concludes the paper and highlights future work.

II. BACKGROUND

A. MIGRATION TO MICROSERVICES ARCHITECTURE

Lewis and Fowler [5] defined the microservice architecture as "an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms." In a microservice architecture, each microservice is autonomous, allowing developers to select the most appropriate set of tools and programming languages to be used. Small services tend to reduce code complexity and increase code maintainability. Moreover, because microservices are deployed independently, each microservice has its own delivery pipeline, can be tested independently, and can be scaled individually [5].

The microservice architecture is an alternative to monolithic applications, which are developed as a single unit [14]. Compared with monolithic applications, microservices are easier to scale, have shorter cycles for testing, building and release, and are frequently less affected by downtime [1]. However, the microservice architecture also has drawbacks and challenges. Having each service deployed separately introduces latency in communication, requires the management of network failures, increases operational complexity, and demands the management of eventual consistency [1]. During migration to microservices, practitioners reported extended time to release features, high coupling, and deficiencies in communication and knowledge sharing, among others [15].

Microservices may be considered as a way of implementing Service-Oriented Architecture (SOA), although there are different opinions about whether microservices are an instance of SOA [16]. There is a clear overlap between the characteristics of SOA and the microservice architecture. Many concepts and techniques in microservices have been borrowed from SOA, such as service discovery, service registries, API gateways, and circuit breakers [17]. Even so, SOA describes applications that cannot be considered microservices. For example, many SOA applications are still implemented using an Enterprise Service Bus (ESB), a centralized software component providing infrastructure to the services composing the application and mediating communication. An ESB may intercept and modify the data, among other functions [18]. On the other hand, microservices require a *dump pipe* for communication (i.e., a communication layer used simply to transfer data) without any modifications or transforming capabilities. Other characteristics apply for SOA but not for microservices, such as: there is no such guidance about the service granularity in SOA, while for microservices, each microservice should represent only one capability, and SOA may support transport protocol transformations, while microservices usually rely on REST over HTTP or a protocol supported by a message bus [19].

Well-known companies, such as Amazon and Netflix, have been using microservices to overcome difficulties with their previous monolithic architectures [5]. The success of

microservices in these companies made other companies embrace this architectural style and migrate to it from their previous monolithic architectures. There are many reports on such migrations in the industry and academia [10].

There are several approaches for migrating monolithic architectures to microservices, ranging from decomposition strategies to data-driven approaches [8]. Many patterns have also been discussed in academia and industry to guide migration [14], [20]. Frequently, the migration is advised to be incremental in that microservices gradually replace the functionality in the monolith or new microservices are created to implement new features [20]. An incremental migration results in both the original and new architectures coexisting and working together.

B. PRIORITIZATION OF ARCHITECTURAL DEBT (ATD)

Technical debt (TD) denotes a suboptimal solution that delivers short-term benefits at the expense of increased overall costs in the long run [21]. An ATD is a type of TD related to a product's architecture [3]. Findings from previous surveys identify ATD as one of the most challenging types of TD to unveil and manage [12], [22], [23].

Microservice architecture is prone to ATD. De Toledo *et al.* [2] listed 16 ATDs specific to microservices (i.e., MS-ATDs), which were organized into 12 more general ATDs. Despite the possibility of finding these ATDs in other architectural styles (e.g., APIs might be inadequately used in any architectural style), their causes and consequences are different from those in microservices.

Table 1 shows seven MS-ATDs, an example for each, and a brief explanation of what is specific to microservices in each debt. These MS-ATDs are a subset of those initially reported by de Toledo *et al.* [2], and were selected according to the criteria described in Section III-A. Table 1 also lists the number of each MS-ATD in de Toledo *et al.* [2] for correspondence.

MS-ATDs must be prioritized before repayment [13]. One way of prioritizing is through risk assessment [24], [25]. Risk represents the possibility of loss (suffering the impact of the debt). The MS-ATDs that pose the highest risk should be addressed first. Through risk assessment, we developed a systematic approach to identify, analyze, and evaluate the risk of each MS-ATD. We define the risk of a debt as the probability of the debt to occur multiplied by its impact, as shown in Equation 1. We use this definition in Section III-D to address the risk of the debts.

$$risk(debt) = probability(debt) \times impact(debt) \quad (1)$$

C. MS-ATD DURING MIGRATION TO MICROSERVICES

After deciding to migrate to microservices, a company must consider completely rewriting the software in the new architectural style or proceeding with an incremental migration. A company must make this decision by considering its context. However, a complete rewrite is often very costly. In that case, there are approaches that companies may use to proceed with an incremental migration, such as those proposed by Yoder and Merson [20] and Newman [14]. The companies in our study executed an incremental migration.

MS-ATDs can occur during migration to microservices. For example, unplanned data sharing may arise when practitioners share databases among several microservices and the previous architecture, or microservice APIs may be malformed because practitioners maintain compatibility with the previous architecture. Knowing which MS-ATDs occur at different times during migration might help practitioners overcome their difficulties before they become too harmful, reduce costs, and speed up migration.

III. RESEARCH DESIGN

This section describes the process of our exploratory multiple-case study [26], which is summarized in Figure 3, and a detailed protocol is available online.[4]

Our study was conducted in three companies during the early stages of migration to microservices. We scheduled three 45-minute presentations, one for each participating company, with practitioners involved in microservice projects in the respective companies. The goals of the presentations were to raise practitioners' awareness of MS-ATDs, ensure they had the same understanding as the researchers regarding the debts, and collect initial data.

During the scheduled presentations, we introduced the practitioners to a list of MS-ATDs based on a previous study (Step 2 of Figure 3). The identification and selection (Step 1) of those MS-ATDs are described in Section III-A. Next, we asked interviewees to answer a set of predefined questions (Step 3). The results from the interviews were analyzed, summarized, and presented to a subset of the original participants in another round of three 45-minute presentations, one for each company (Step 4). During the second interaction with the companies, we collected additional information through semi-structured interviews (Step 5). We recorded all interactions with the participants for posterior analysis.

A. IDENTIFICATION AND SELECTION OF THE MS-ATDs USED IN THIS STUDY

Practitioners from the participating companies granted us a limited amount of time enough to prioritize seven MS-ATDs. We selected those MS-ATDs from a list found in de Toledo *et al.* [2], one of the most comprehensive studies covering MS-ATDs in large companies running mature microservice projects. The list from de Toledo *et al.* [2] includes 12 debts, and we selected seven debts according to the following criteria:

(i) We selected the debts reported by at least three companies in the previous study, resulting in six out of the original 12 debts. We considered that frequent MS-ATDs found by companies running microservices for several years are likely to be found in other companies. The original study divided the debt "reusing third-party implementations" into two distinct sub-debts.

[4] https://bit.ly/3qVwFJq

TABLE 1. The MS-ATDs selected for this study.

#	Name and description	Example	What is new in microservices?
1	*Insufficient metadata* (#1 in [2]): Many microservices communicate through messaging. However, these messages could have additional metadata to identify their producers, consumers, targets, and others in some cases. Insufficient metadata may make it challenging to track dependencies among services and find the producers for debugging purposes, as well as other issues.	A system comprises many microservices processing data available through a message bus. In this example, a message can only be consumed by one service at a time, but there is no guaranteed order. If the message gets malformed, the reason might be related to a specific combination of modifications made by previous services. If there is no metadata for tracking the changes, it might be challenging to identify the causes of the issue.	Different SOA approaches might use messages in their communication, but microservices are more fine-grained than those, which increases the number of services and, consequently, the impact of the debt. Monoliths are self-contained and usually do not need messaging approaches to establish communication among their modules. On the other hand, Microservices might critically depend on messages to communicate with other services because they form a distributed system.
2	*Microservice coupling* (#2 in [2]): Microservice coupling is about how changes in one service require changes in another service. The coupling might be done intentionally to save development time, and it frequently increases team dependency. There are different types of coupling [14], but we focus on the services' implementation, including their contracts and interfaces (e.g., API endpoints and message formats).	A *files* service provides access to a set of files for authorized users. The authorization is currently performed at the *users* service. For every request received by the *files* service, another request is made to the *users* service to verify access to the files. Changes to the *users* service's API might affect the *files* service, creating a dependency among the development teams. The files access authorization should possibly be moved to the *files* service instead.	Coupling among microservices frequently causes dependency among teams, reducing teams' velocity and agility. While companies have a false impression that they have decoupled teams and well-defined agile practices, they might be silently blocked by coupling among the microservices.
3	*Inadequate use of APIs* (#4 in [2]): Many services communicate with the other services through APIs. When these APIs are not well defined or misused, they lead to issues.	An API exposed through HTTP that is not following required standards. For example, removing an item from a shopping cart is done through an HTTP method called GET, but it should use DELETE instead. This API's users have difficulties understanding it.	Each microservice exposing an API might be developed by a different team, increasing the probability of having many different API standards. The inadequate use of APIs impacts the functioning of other services and development teams.
4	*Excessive diversity* (#6 in [2]): Microservices allow mixing multiple programming languages, data-storage technologies, supporting tools, and others. However, having too many different technologies across the system may create difficulties with standardization and management.	Using containers is a common technique in environments running microservices. Developers can easily find start containers running almost any GNU/Linux-based system to use in their projects. However, there are several hundreds of containers setups, and each one uses its own distinct tools. Using too many distinct containers for solving the exact same problem requires team members to learn a different setup every time they change teams.	A monolith is usually developed using a limited set of programming languages and tools. On the other hand, Microservices can be developed with completely distinct languages and setups, increasing the likelihood of excessive diversity. Other SOA approaches might also suffer from this problem, but the number of services in a microservice architecture is usually higher, making the problem more costly.
5	*Unplanned data sharing/synchronization* (#8 in [2]): Microservices should have their own databases. When different microservices share the same database, unexpected issues such as cascading breakings may occur. On the other hand, when microservices have distinct databases, there might be problems with data synchronization.	Two microservices share a database and use a table of users. There is a field in the database storing the users' full names. The developers in the first service decide to split the *full name* column into *first name* and *last name*. The second service might stop working because it does not find the original field for the full name.	Microservices should have their own databases, which is not required for other architectural styles, including other SOA approaches. The way the databases are designed for microservices is also different: they should reflect the business domain, which leads to distinct domain-related issues. Sharing databases or synchronizing them might lead to blocking and other issues among teams.
6	*Misusing shared libraries* (#10.1 in [2]): Many companies encapsulate code into libraries and distribute them to be used by many services. Suppose such a distribution is not properly managed. In that case, many libraries may lead to difficulties, including breaking changes, dependencies among teams, delays, and additional costs to update all the services using the libraries on every new release.	A data encryption library is developed by a separate team in the company and used by dozens of services throughout the project. A high-security issue is found in the library, and the library developers release a new version with the fix. Every service should update the library to the last version. Due to other priorities and feature development, it is not possible to update the library in the entire organization, and many services will remain with the issue.	Libraries have different consequences in distinct architectural styles. Monoliths, for example, bundle them in a single deployment package, while microservices do the same for each deployment. A system with hundreds of services may have hundreds of deployments of the same library. Thus, issues found in a single library immediately affect dozens or hundreds of services and teams.
7	*Unnecessary settings* (#11 in [2]): Each microservice usually has a set of settings to be defined in the environment, such as the database address, memory limits, and others. However, if there are many unnecessary settings, the probability of misconfiguration is more significant, potentially leading to crashes or other issues.	One of the configuration settings for accessing databases is to inform a "port number." Databases run in a default port if not changed. For example, it is unnecessary to have a configuration setting in an application to inform the default port of a database management service. The application could have an optional configuration setting that, if not present, automatically falls back to the default value.	There are several times more settings in microservices than in monolithic architectures, increasing the likelihood of unnecessary settings. The impact of those settings in a distributed setup is higher than in a single deployment setup because of the more significant amount of settings and the possibility of causing cascading failures.

For simplicity, we focused only on the sub-debt "misuse of internal shared libraries" reported by multiple companies.

(ii) We previously knew that the companies involved in this study extensively used asynchronous communication among services and were interested in prioritizing any related MS-ATDs. Thus, we included an additional debt in our list: "Insufficient metadata," the only of the remaining debts related to asynchronous communication among services, resulting in a total of seven debts, as detailed in Table 1.

Other debts we did not consider in this study might be relevant to the companies, including (but not limited to) the debts identified by de Toledo *et al.* [2]. However, additional debts might be considered in future prioritization by practitioners and future research studies.

B. STUDIED COMPANIES

We studied three large software companies that had just started modernizing their (large) legacy monolithic systems using a microservice architecture. Figure 2 presents the relationship between the research questions, the ongoing architecture (with the migration in progress), and the final microservice architecture as presently visualized by the practitioners.

Company A provides business software and IT-related development and consultancy, employs nearly a dozen thousand employees, and has hundreds of thousands of customers, mainly in northern Europe. The participants in our study are from the core teams that have been developing a dynamic ERP system for large companies, which is one of the company's flagship products. It is a large monolithic system in which customers can buy licenses and install them through Company A or its certified partners. Software teams have broken down their monolithic ERP product towards using microservice architecture and providing their product as a service on the cloud. One of their main goals is to avoid pitfalls and (remove) debts while migrating their product.

Company B provides IT and product engineering services, with approximately twice as many employees as Company A. Company B serves thousands of customers in more than 90 countries. The branch with which we conducted our study focuses on financial services, such as banking solutions, and is located in a Nordic country. They sell and maintain complex banking solutions for many banks and have continuously developed and modernized their products, focusing more recently on microservice architecture and modern software development. MS-ATD is a considerable concern that software teams want to control better.

Company C is one of the largest financial services groups in the Nordic region (mainly banking) with 9000+ full-time employees serving several millions of customers. The software teams we interacted with were at the core of their in-house software department, specialized in architecture and technology. This software department has a good tradition of handling TDs and has a reputation for allowing engineers to take software engineering courses.

TABLE 2. Attendees for the first presentation.

Roles	Number of attendees			Total
	Company A	Company B	Company C	
Developer	1	14	7	**22**
Architect	6	1	11	**18**
Manager	2	2	1	**5**
Other	0	2	0	**2**
Total	**9**	**19**	**19**	**47**

C. DATA COLLECTION

The data collection occurred as illustrated in Steps 3 and 5 in Figure 3. A total of 47 participants, distributed as shown in Table 2, participated in the first data collection (Step 3). The participants had different backgrounds and experiences with microservices. For each MS-ATD presented, we asked the participants the following three questions:

(i) Have you encountered this MS-ATD in your current project? (RQ1)
(ii) Do you foresee this MS-ATD in the future of the project? (RQ2)
(iii) Do you know how to avoid or mitigate this MS-ATD? (RQ3)

The first two questions were answered with *yes*, *not sure*, or *no*. The third question was answered with *yes*, *partially*, *no*, or *not applicable (n/a)*. *Not applicable* was used by the practitioners that considered the MS-ATD as irrelevant or out of context in their projects. For example, insufficient metadata in messages is not applicable to contexts in which messages are not used.

We also asked the participants to report whether they understood the explanation of the MS-ATD. Only three participants from Company B said that they did not understand the explanation, which concerned MS-ATD 1, MS-ATD 2, and MS-ATD 5, as described in Table 1. In general, our explanation of the MS-ATDs was well accepted and understood by the participants.

The practitioners perceived some MS-ATDs as riskier than others and, as such, more important to them. At the end of the presentation, we asked the participants to rank the three most important MS-ATDs according to their point of view on the project (RQ4).

After data analysis (Section III-D), we invited the most experienced interviewees from the previous session to a new group interview to discuss the results of the previous interviews. These experts constituted approximately 30% of the original participants. We presented the results, asked for clarifications of context, and discussed the prioritization of MS-ATDs in future development (RQ5). More specifically, we asked the following questions:

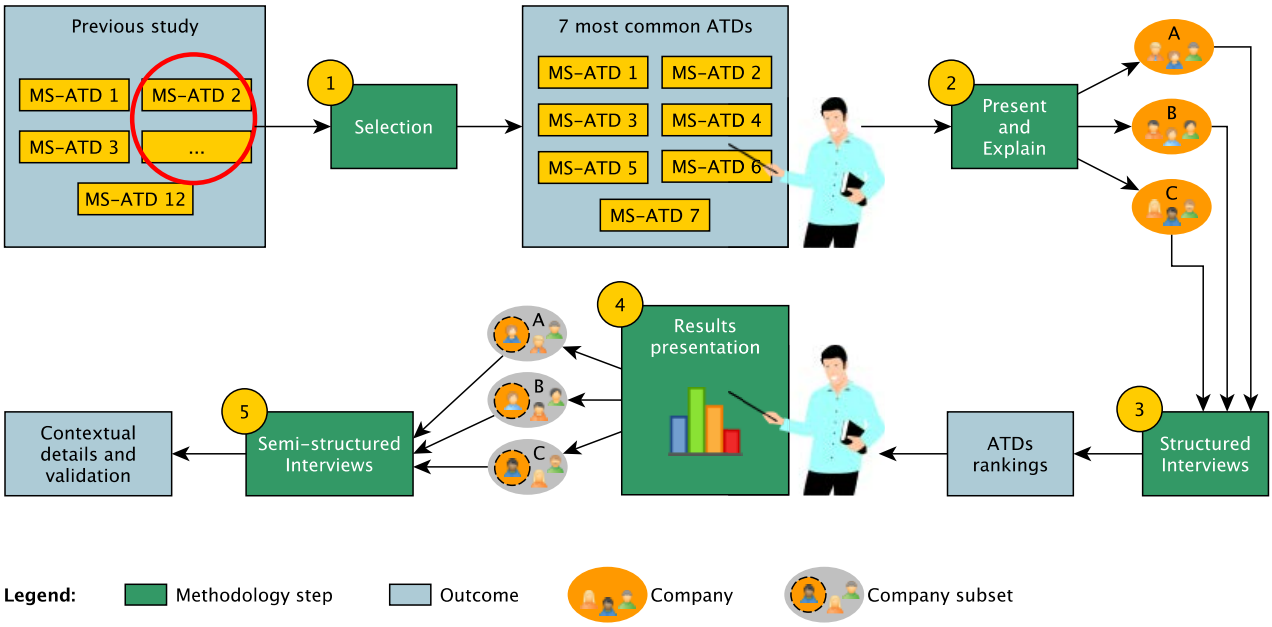


FIGURE 3. An overview of the research process.

(i) What are your considerations regarding these results?
(ii) What are the causes of these results?
(iii) Do you agree with these results? Why?

These experienced interviewees had a good overview and understanding of their projects and thus provided additional helpful information for our analysis.

D. DATA ANALYSIS

Descriptive statistics were used to compare the results for the different companies. We created rankings for each question, for example, from the most found to the last found MS-ATD in RQ1. To create the rankings, we transformed the categorical answers into numeric values, as follows:

- Every MS-ATD reported as found, foreseen, or difficult to solve was counted as 1 for each answer.
- MS-ATDs reported as not found, unforeseen, not difficult to solve, or not applicable to their context were counted as 0.
- Partial answers, i.e., "not sure" or "partially," were counted as 0.5. This value represents a 50% probability of a debt being found. It is reasonable to assume that some practitioners are more confident than others when answering these questions. Thus, 0.5 is an approximation to balance all answers. Future studies may find better measures of practitioner confidence in answering these questions.

The remainder of this section explains the analysis procedure in detail. The set of MS-ATDs in Table 1 is denoted by $D = \{d_1, d_2, \ldots, d_7\}$.

1) THE RANKING OF THE MOST ENCOUNTERED MS-ATDs

The question about the MS-ATDs encountered so far had three possible answers: *yes*, *not sure*, and *no*. We counted the number of votes for each answer: e_{yes}, e_{no}, and $e_{not\ sure}$. We weighted each answer: 1 for **yes**, 0 for *no* (because they represent MS-ATDs that were not encountered and, thus, are not relevant for our ranking), and 0.5 for *partially*.

Based on the e_{yes}, e_{no}, and $e_{not\ sure}$, and on the respective weights, we computed the weighted sum of votes v_{e_i} for each MS-ATD encountered $d_i \in D$, as defined in Equation 2.

$$v_{e_i} = (1 \times e_{yes}) + (0.5 \times e_{not\ sure}) + (0 \times e_{no}) \quad (2)$$

The set with all values v_{e_i} was used to compute the ranking $R_{encountered}$ of the MS-ATDs encountered by practitioners. The ranking is defined in Equation 3 and represents the ordered set of debts $d \in D$ according to the respective values previously calculated: d_i is higher than d_j if v_i is greater than v_j, which means that d_i is encountered more than d_j. For cases in which the weighted sum of votes is numerically the same for more than one MS-ATD, we consider as most encountered the debt with less uncertainty, i.e., with less "not sure" answers.

$$R_{encountered} = order(D, \{v_{e_1}, v_{e_2}, \ldots, v_{e_7}\}) \quad (3)$$

2) THE RANKING OF THE MOST FORESEEN MS-ATDs

The question regarding the most foreseen MS-ATDs in the project's future had the same possible answers (*yes*, *not sure*, and *no*) as for the question about encountered MS-ATDs. We used the same reasoning as before to calculate Equations 4 and 5.

$$v_{f_i} = (1 \times f_{yes}) + (0.5 \times f_{not\ sure}) + (0 \times f_{no}) \quad (4)$$

$$R_{foreseen} = order(D, \{v_{f_1}, v_{f_2}, \ldots, v_{f_7}\}) \quad (5)$$

3) THE RANKING OF THE MS-ATDs THAT THE PARTICIPANTS DO NOT KNOW HOW TO SOLVE

The third question, which asked whether the participants knew how to solve each MS-ATD, had four possible answers: *no*, *partially*, *yes*, and *not applicable (n/a)*. We aimed to have a final ranking in which the first MS-ATD was the one in which the practitioners did not have the complete solution (since they could also answer *partially*) or did not know how to avoid or mitigate. We counted the number of votes for each answer, k_{no}, $k_{partially}$, k_{yes}, and $k_{n/a}$, and applied weights for them: 1 for *no*, 0 for *yes* and *n/a* (because they represent MS-ATDs for which the companies already have a solution or that do not apply to their cases), and 0.5 for *partially* (because they represent a solution that does not entirely repay the MS-ATD, but that at least reduces its risk).

Based on k_{no}, $k_{partially}$, k_{yes}, and $k_{n/a}$, and on the respective weights, we computed the values v_{k_i} for each $d_i \in D$, as defined in Equation 6, which were used to compute the ranking R_{known} of the MS-ATDs, as described in Equation 7.

$$v_{k_i} = (1 \times k_{no}) + (0.5 \times k_{partially}) + (0 \times k_{yes}) + (0 \times k_{n/a}) \quad (6)$$

$$R_{known} = order(D, \{v_{k_1}, v_{k_2}, \ldots, v_{k_7}\}) \quad (7)$$

4) THE RANKING OF THE IMPORTANCE OF THE MS-ATDs FOR THE PARTICIPANTS

The ranking of the importance of the MS-ATDs given by the participants is formed by the ordered set of MS-ATDs $d_i \in D$. d_i is higher than d_j if v_{i_i} is greater than v_{i_j}; v_{i_i} is the number of votes for MS-ATD i. In other words, we have an ordered list from the most important to the least important MS-ATD (see Equation 8).

$$R_{importance} = order(D, \{v_{i_1}, v_{i_2}, \ldots, v_{i_7}\}) \quad (8)$$

5) THE FINAL RANKING OF IMPORTANCE FOR THE MS-ATDs

TD can be threatened as a software risk because of the uncertainty of interest payments [24], [25]. Therefore, a risk analysis is appropriate for prioritizing our MS-ATDs; the first MS-ATD to be paid is the one that poses a higher risk to the company and the project. As presented in Equation 1, the risk of an MS-ATD can be calculated as the probability of the debt to occur multiplied by the impact of that debt. Therefore, we computed the priority score p_i for each MS-ATD i in Table 1 based on this risk definition. Such a priority score is calculated using Equation 9: (i) the probability of having the debt is calculated as the product of the number of practitioners who believe the debt will happen in the future and the number of people who do not know how to solve the debt; (ii) the impact is represented by the importance given by the practitioners for the debt (we add 1 to the number of votes on the importance to prevent multiplication by zero if no practitioner has voted for the debt as important).

Our approach, represented by Equation 10, uses the priority scores to compute the priority ranking. However, other authors may consider different methods.

$$p_i = (v_{f_i} \times v_{k_i}) \times (1 + v_{i_i}) \quad (9)$$

$$R_{priority} = order(D, \{p_1, p_2, \ldots, p_7\}) \quad (10)$$

6) VISUALIZING THE PRIORITY RANKING

We used the score defined in Equation 9 for the prioritization ranking. However, for visualization and readability purposes, we applied the transformation [27] defined in Equation 11 to the score.

The maximum value of the priority score depends on the number of participants and votes, and there is no upper limit. Therefore, we normalized the score between 0 and 1 by dividing it by the maximum score. We used a logarithmic transformation to reduce the differences between the values. Using two as the logarithm base is reasonable when the data range is less than two powers of 10 [27]. We add 1 to the normalized value to ensure that we do not have negative numbers after our transformation (if the priority is zero, we have $log_2(1) = 0$ as the minimal value possible). Finally, we transformed the score into a scale between 1 and 10 by multiplying the results by 10.

$$s_i = \log_2 \left(\frac{p_i}{max(\{p_1, p_2, \ldots, p_7\})} + 1 \right) \times 10 \quad (11)$$

7) THE QUALITATIVE ANALYSIS

We used the recorded interviews to identify contextual information that could explain the practitioners' decisions. Owing to the limited interview time, the practitioners focused on the MS-ATDs they considered the most important while discussing each RQ. In a few cases, the practitioners also discussed debts that were less important to them.

For each MS-ATD in our ranking, we looked for a mention of the debt during the interviews. We found explanations and quotations that helped us interpret the results. For example, when asked about the reasons for having shared databases, one interviewee from Company A said, "*We decided to share the database at the beginning of the migration to speed up the process*." Thus, the debt exists at the top of their rankings because they explicitly decided to have it, and they are paying the respective costs.

IV. RESULTS AND DISCUSSION

Table 3 shows our raw data with the number of participants who voted for each answer in the first three RQs (i.e., for the MS-ATDs encountered so far, foreseen in the future, and that the practitioners do not know how to avoid), and the number of votes for importance, for each company. The raw data may be used by the reader to replicate our study or to use them in different ways as those proposed in this work.

Figures 4, 6, and 8 show the percentage of practitioners who voted for each answer in each MS-ATD regarding the respective RQs.

Tables 4, 5, 6, and 7 present the rankings calculated from the data in Table 3 using the formulas described in Section III-D. These tables contain colors to facilitate the

TABLE 3. The raw answers for the most encountered MS-ATDs, the most foreseen MS-ATDs, the MS-ATDs practitioners do not know how to avoid, and the importance for each MS-ATD according to practitioners.

ATD	Company	MS-ATDs encountered			MS-ATDs foreseen			MS-ATDs practitioners do not know how to avoid				Votes for importance
		Found	Not sure	Not found	Foreseen	Not sure	Not foreseen	No	Partially	Yes	N/A	
Insufficient metadata	A	2	2	5	4	2	3	4	1	1	3	1
	B	1	6	9	3	7	5	2	5	0	9	4
	C	6	5	7	9	7	1	1	6	3	5	5
Microservice coupling	A	4	1	4	7	1	1	4	3	1	1	9
	B	8	4	4	12	4	0	6	5	1	5	8
	C	10	1	3	10	1	2	5	6	2	1	13
Inadequate use of APIs	A	6	0	2	5	1	3	2	4	2	1	6
	B	6	3	5	7	4	2	0	6	5	3	6
	C	9	1	3	11	1	1	0	12	1	0	4
Excessive diversity	A	2	0	6	3	0	5	3	3	1	1	0
	B	1	1	12	6	0	7	0	8	2	2	1
	C	7	0	6	8	2	3	4	6	3	1	6
Unplanned data sharing/sync.	A	6	1	2	5	3	1	1	7	0	1	7
	B	7	4	3	10	3	0	4	6	1	2	7
	C	4	4	6	6	4	3	3	7	2	2	6
Misusing shared libraries	A	6	1	1	6	2	1	4	3	1	1	2
	B	10	0	3	11	1	1	5	6	2	0	10
	C	9	0	3	11	3	0	6	6	1	1	3
Unnecessary settings	A	6	0	1	6	1	1	1	4	3	1	2
	B	9	2	4	8	1	5	5	4	3	1	3
	C	9	2	2	10	1	2	2	9	1	1	2

identification of the MS-ATD for the distinct companies, i.e., the same MS-ATD has the same color for all the rankings, facilitating the comparison among them. We focused on the top-3 debts of the rankings because they were the debts the companies mainly discussed in our follow-up interviews and were thus most relevant to the companies.

A. WHICH MS-ATDs DO COMPANIES ENCOUNTER DURING EARLY MIGRATION TO MICROSERVICES? (RQ1)

Figure 4 shows the percentage of practitioners who voted for each answer. Table 4 shows a ranking calculated as defined by Equation 3, ordered from the most found MS-ATD to the less found MS-ATD. Figure 5 shows the values used to calculate the rankings and is used to support our discussion.

1) MISUSING SHARED LIBRARIES

"Misusing shared libraries" is among the most encountered MS-ATDs for Companies A and B, and fourth for Company C. This debt has the least uncertainty among the practitioners (see Figure 4). Only 13% of the participants from Company A, the company with the fewest participants, answered "Not sure." Among the participants from all three companies, 75–77% answered "Found."

A practitioner from Company A said, "*We have created a lot of smaller projects ourselves that we use in our solutions as packages [as shared libraries] in the monolith*." Another practitioner complemented with an example to justify why they use shared libraries: "*During the migration, we are still sharing the database with the monolith. We have some encrypted data that must be accessed by the same decryption algorithms available as a library. So, we share the library among the microservices*." Thus, in this case, the need for such shared libraries is caused by the dependency on the monolith.

Company B has internal restrictions on how many times a service should be called, as explained by one interviewee: "*If you try to run this external validation several times an hour, suddenly you get a call from those running that service saying that you cannot do this*." They work around these issues using shared libraries instead of relying on external services, which makes them use more shared libraries and might indicate an infrastructure that is not yet prepared for distributed systems.

For Company C, the weighted sum of votes for this debt is similar to that for top-3 debts (see Figure 5c). Figure 4 shows that as many as 75% of the practitioners encountered it in their projects. As a financial services company, they have several legacy systems. These systems potentially share code with microservices through libraries, increasing the likelihood of having this debt.

2) UNNECESSARY SETTINGS

"Unnecessary settings" is the only MS-ATD in the top-3 for all companies. The number of practitioners unsure about the presence of this debt is relatively small compared with other

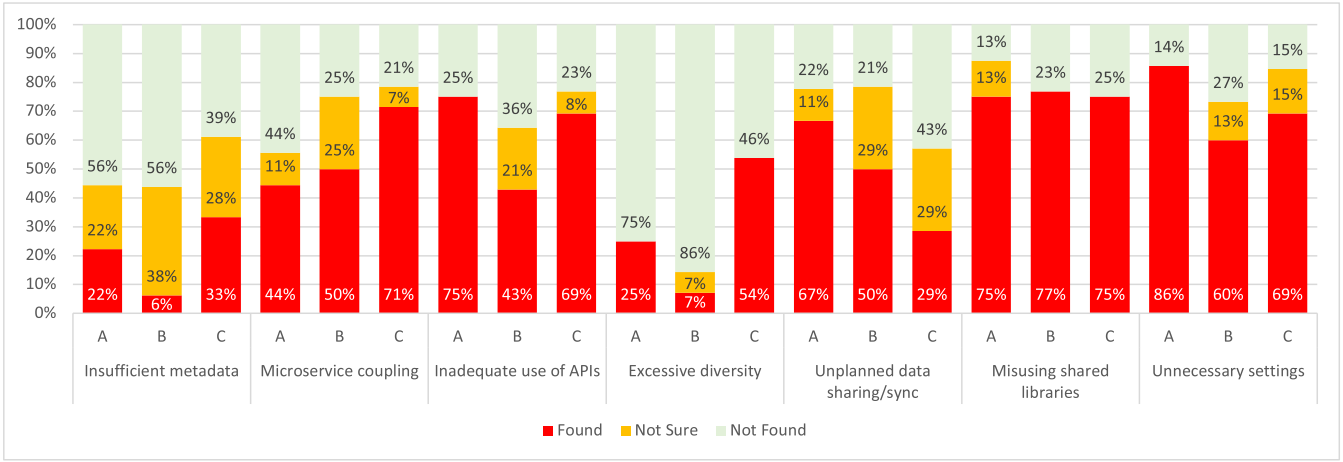


FIGURE 4. Percentage of practitioners who voted for each answer regarding the debts found on each company. Not all practitioners voted for all questions for each company.

TABLE 4. Ranking of the most encountered MS-ATDs calculated through Equation 3. Each MS-ATD is associated with the same color to facilitate identifying it across the rankings from the distinct companies.

	Company A	Company B	Company C
1	Misusing shared libraries	Unnecessary settings	Microservice coupling
2	Unplanned data sharing/synchronization	Misusing shared libraries	Unnecessary settings
3	Unnecessary settings	Microservice coupling	Inadequate use of APIs
4	Inadequate use of APIs	Unplanned data sharing/synchronization	Misusing shared libraries
5	Microservice coupling	Inadequate use of APIs	Insufficient metadata
6	Insufficient metadata	Insufficient metadata	Excessive diversity
7	Excessive diversity	Excessive diversity	Unplanned data sharing/synchronization

debts, and 60–86% of the practitioners reported it as found (see Figure 4). Thus, this debt is relatively common across companies and easy to recognize.

One interviewee from Company A said, "*The situation regarding configuration settings today is chaotic*" while trying to explain that there was no approach to control the addition of unnecessary settings to the services, and, thus, the debt was common to be found. Companies B and C reported that this debt is so common that it must be accepted when using microservices. One interviewee from Company B, for example, said, "*This is an expected consequence of having many small services, each with its own settings*." Only practitioners from Company A reported that they would like to mitigate this debt in the future.

3) MICROSERVICE COUPLING

"Microservice coupling" is among the three most encountered MS-ATDs for Companies B and C. A considerable number of practitioners are unsure of the existence of this debt (see Figure 4). Possible reasons are that the practitioners did not perceive the costs of this debt in the current stage of their projects, that they did not have a tool to make those dependencies visible, or that they might not be sure about the design of their services.

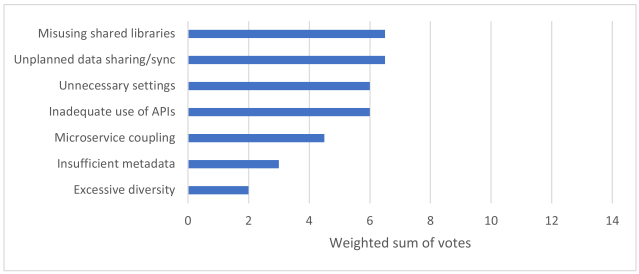
Company A explained that the new microservices are one way to reduce coupling from the previous system. However, they do not seem concerned with coupling among the microservices themselves in the current stage because they are in an early stage of migration, with only a small part of a monolithic architecture migrated to microservices. Answering this question requires observing Company A again in the future to understand the evolution of microservice coupling.

Companies B and C seem to have a higher number of microservices and teams involved with the services than Company A. Therefore, it was easier for them to visualize microservice coupling.

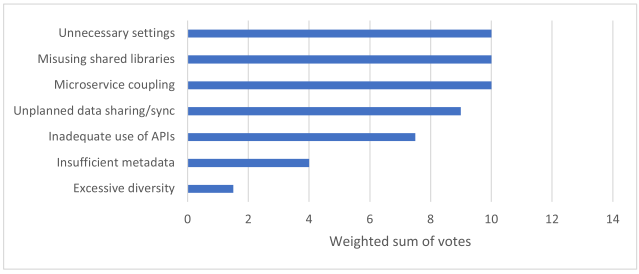
Our impression is that the practitioners only start to think about microservice coupling at a later stage when more couplings have been created.

4) UNPLANNED DATA SHARING/SYNCHRONIZATION

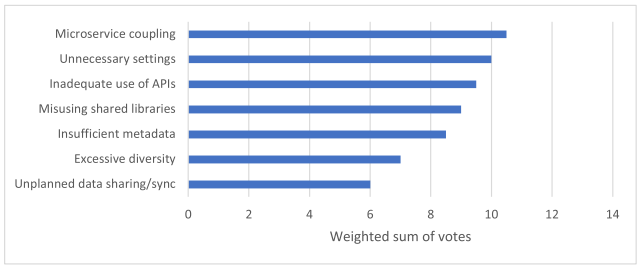
Microservices recommend decentralized data management; however, some practitioners do not entirely agree with this recommendation. When centralizing data management, some additional services may have to share data with other services in a way that was not previously planned. On the other hand, practitioners might not properly plan the database synchronization properly when decentralizing data man-



(a) Company A



(b) Company B



(c) Company C

FIGURE 5. Values for the calculation of the ranking of MS-ATDs found for each company.

agement. These situations lead to ''unplanned data sharing/synchronization,'' which only appears on the top-3 list for Company A. This debt represents the difficulties of splitting a large database that is running for a long time or synchronizing distinct databases. Many practitioners were uncertain about the existence of this debt (see Figure 4), indicating that identifying it is more challenging than identifying others.

Several practitioners have stated that they wondered whether sharing databases across microservices is an incorrect decision. One reason for that is the trade-off represented by this debt: splitting the database increases issues with synchronization among services.

Company A decided to split the database at a later stage of its migration to microservices. Thus, they deliberately acquired this debt to accelerate the initial steps of their migration and plan to repay it later.

For Company B, despite this debt being ranked fourth, at least 50% of the practitioners reported it and 29% were unsure (see Figure 4). This debt is almost as important as the other debts.

It is not clear from our data why this debt is the last on the list for Company C. It might be that the practitioners from Company C who participated in our interviews were primarily involved in well-designed services that had their own databases and did not have to synchronize with other services. This can only be confirmed by a more in-depth study.

5) INADEQUATE USE OF APIs

''Inadequate use of APIs'' only appears on the top-3 list for Company C.

This debt was ranked fourth in Company A, but with as many as 75% of developers reporting it, it was close to the top three debts. In Company B, this debt has been reported less.

Company C could not explain why this debt was among the most found when asked during the follow-up interviews. However, we might have identified a disagreement between technical leaders and other practitioners. The company will discuss it internally.

6) INSUFFICIENT METADATA

''Insufficient metadata'' is the debt with most uncertainty among all the debts. In practice, many practitioners could not connect the debt with their examples. It is unclear whether additional metadata can resolve the current issues. To be repaid, this debt may require a global overview of the architecture. However, many practitioners focus on their own services, and only a few have such a global overview of the architecture.

7) EXCESSIVE DIVERSITY

''Excessive diversity'' is the debt in which the majority of the participants had a strong opinion about its existence: only 7% of the participants from Company B reported not being sure about it, while all the other practitioners answered ''found'' or ''not found'' (see Figure 4). However, in our interviews, we noticed a lack of consensus on the extent to which this is a debt. Some practitioners believe that such technology diversity is acceptable, while others believe it incurs high costs.

Only 25% of the practitioners from Company A reported this debt as found. Company A has a well-defined set of technologies and platforms for the development of its microservices. Moreover, they have only migrated a small part of their monolith to microservices, and used the same .NET Core technology stack. Thus, Company A expected to have fewer complaints about this debt than the other companies.

Company B was satisfied with its current policy on the diversity of technologies while using microservices. On the other hand, Company C is divided on their opinions; about half of the practitioners believe there is a problem with their policy on diversity, but in spite of that, they did not want to limit the technologies used by other teams.

Main findings

F1. The use of shared libraries starts at the early migration stages and is usually related to the convenience of reusing

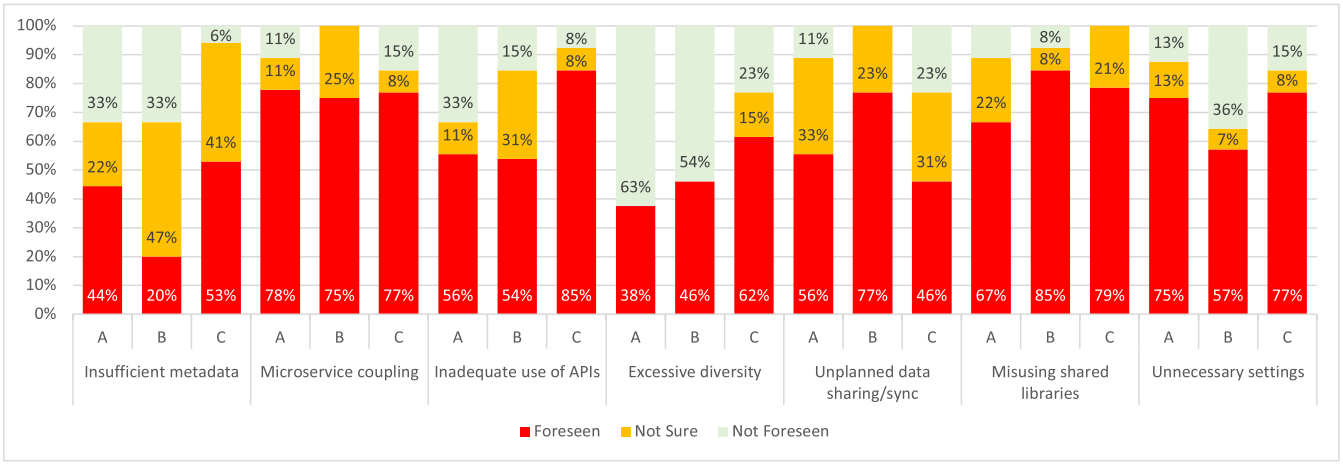


FIGURE 6. Percentage of practitioners who voted for each answer regarding the debts foreseen on each company. Not all practitioners voted for all questions for each company.

code from the original architecture. However, companies may misuse the shared libraries.

F1. Unnecessary settings are common during the early stages of migration. Some practitioners try to find ways to mitigate this debt (sooner or later), while others find it more convenient to maintain the debt and its extra cost.

F2. The costs of microservice coupling are not recognizable in the initial stages of migration, and the debt is down-prioritized. Practitioners may not prioritize this debt and may tend to postpone repayment.

F3. Compared to the other debts, insufficient metadata and unplanned data sharing/synchronization are the debts in which more practitioners are unsure about their existence, which might indicate that identifying these debts is more challenging than identifying others.

B. WHICH MS-ATDs DO COMPANIES FORESEE IN THE FUTURE OF THE MIGRATION? (RQ2)

Figure 6 presents the percentage of practitioners who voted for each answer regarding MS-ATDs foreseen in the next steps of migration. More practitioners are not sure about the debts in the future than when compared to the debts in the present, as detailed in Section IV-A. Such an increase in the number of "not sure" answers is expected because practitioners are reasoning about a possible future.

The remainder of this section discusses the most foreseen MS-ATDs extracted from the results for each company according to our data and ranking calculation defined in Equation 5. The rankings are presented in Table 5, ordered from the most foreseen to the least foreseen debt. The values used to calculate the rankings are shown in Figure 7.

1) MICROSERVICE COUPLING

"Microservice coupling" is the top item in the ranking for Companies A and B, and the fifth debt in the ranking for Company C. Compared to the currently found MS-ATDs in Section IV-A, 75–77% of the practitioners from all three companies estimate that this debt will increase.

In the follow-up interview with Company A, senior practitioners did not expect such a result because they planned to reduce microservice coupling in the future. This result

FIGURE 7. Values for the calculation of the ranking of MS-ATDs foreseen for each company.

TABLE 5. Ranking of MS-ATDs foreseen in each company calculated through Equation 5. Each MS-ATD is associated with the same color to facilitate identifying it across the rankings from the distinct companies.

	Company A	Company B	Company C
1	Microservice coupling	Microservice coupling	Misusing shared libraries
2	Misusing shared libraries	Misusing shared libraries	Insufficient metadata
3	Unnecessary settings	Unplanned data sharing/synchronization	Inadequate use of APIs
4	Unplanned data sharing/synchronization	Inadequate use of APIs	Unnecessary settings
5	Inadequate use of APIs	Unnecessary settings	Microservice coupling
6	Insufficient metadata	Insufficient metadata	Excessive diversity
7	Excessive diversity	Excessive diversity	Unplanned data sharing/synchronization

highlights that other participants may not share the same point of view. Thus, they may internally discuss the reasons for such concern, as explained by one of the practitioners: "*These numbers inform us that we have an important job in informing everyone about what we want to do*".

One practitioner from Company B explained that this is expected: "*We are going to see more on coupling if nothing is done today to change [the process]*".

For Company C, "Microservice coupling" is among the less foreseen debts (see Table 4). However, all debts had substantial votes by practitioners from Company C: 77% of the practitioners reported this debt as foreseen, and only 8% were unsure (see Figure 6). However, our ranking provides a starting point for prioritizing the debts.

2) MISUSING SHARED LIBRARIES

"Misusing shared libraries" is in the top-3 for all companies in the ranking. We present the reasons for this for each company below.

One interviewee from Company A said, "We have a lot of dependencies on other parts of the system. Instead of implementing something new, we use these dependencies to focus on the main goal. Most of these dependencies will be addressed in the end, but you cannot address them all during the migration process." Therefore, they will still use libraries they believe are necessary and will postpone their removal.

Company B reported no plan to reduce the usage of shared libraries today; they foresee this debt coming again in the future.

Company C started a discussion on whether the shared libraries were an issue. One practitioner said, "*We first need to discuss whether shared libraries are necessarily bad, are they?*" They do not seem to have plans to change how they use these libraries. Thus, they might prevent the costs of misusing shared libraries by closely following library usage. Our discussion on this topic may have increased practitioners' awareness of the debt.

3) UNNECESSARY SETTINGS

"Unnecessary settings" is in the top-3 for Company A only.

Company A visualizes the need to reduce unnecessary settings in the future but believes that the problem will first increase before they have a better approach to handle it. They wanted a solution to the problem, but that was not a priority.

Companies B and C expect this debt, but they accept it and do not have plans to mitigate it. They are not concerned with the costs of this debt. It is possible that it is better to pay interest in this type of debt than to repay it.

4) UNPLANNED DATA SHARING/SYNCHRONIZATION

"Unplanned data sharing/synchronization" is in the top-3 for Company B only. This debt is still among those with more participants who are unsure about the debt. The reasons may be the same as those explained in Section IV-A: practitioners are still questioning whether sharing databases is always a bad practice because they foresee cases in which this seems to be a good solution for the problem.

Company A decided to postpone the migration of the database to the microservices. Thus, they currently have the costs of using a centralized database and do not foresee the complete migration of the database. However, Figure 7a shows that there is no difference in the value used to calculate the rankings between this debt and the "Unnecessary settings," one of the top-3 debts in the ranking for Company A.

Company B saw this debt as a challenge that will increase if nothing else is done to reduce it. Thus, some practitioners have already observed that the company needs to work on a solution for the debt.

It is unclear from our data why this debt goes to the bottom of the list for Company C, a result similar to that described in Section IV-A.

5) INADEQUATE USE OF APIs

"Inadequate use of APIs" is in the top-3 for Company C only.

Company C explained the same as reported in Section IV-A: they could not explain why this debt was among the most found when asked during the follow-up interviews, so they went back to investigate this further with developers. Companies A and B did not provide any further comments.

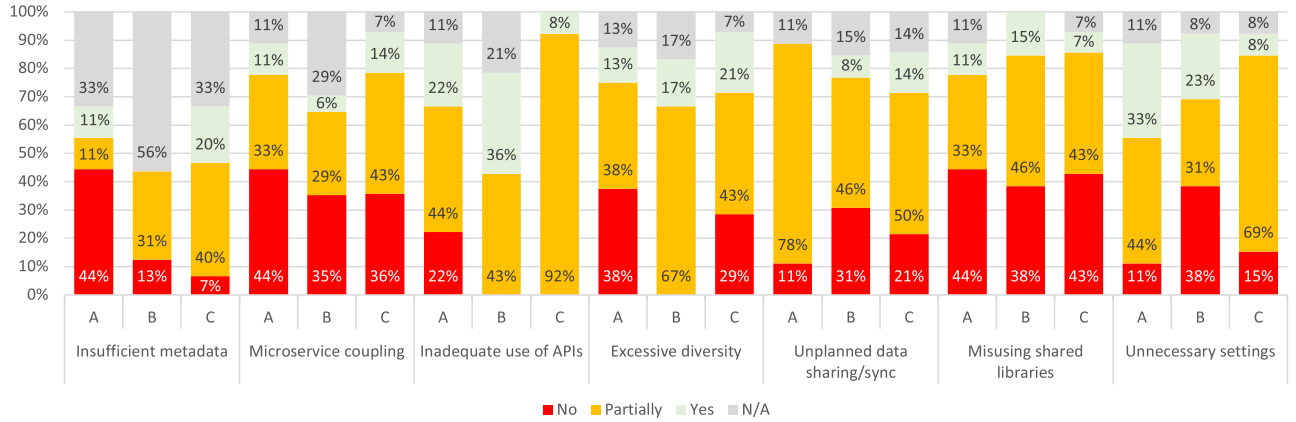


FIGURE 8. Percentage of practitioners who voted for each answer regarding the debts practitioners know how to avoid on each company. Not all practitioners voted for all questions for each company.

6) EXCESSIVE DIVERSITY

"Excessive diversity" is the debt in which the majority of the participants had a strong opinion about its existence: only 15% of the participants from Company C reported not being sure about it, while all the other practitioners answered "found" or "not found" (see Figure 6). Compared with the described in Section IV-A, the participants from all companies believe that this debt will increase in the future.

Company A has the fewest participants among the companies in this study. Such a result is expected because they seem to have reasonable control of technology diversity in their current stage of development.

Company B reported that they do not have proper control of such diversity, which might lead to an increase in this debt in the future, indicating that they should be aware of the issue and control it in advance.

Company C already saw the costs of this debt, and they believed that the problem would increase because there was no plan to limit such diversity.

7) INSUFFICIENT METADATA

"Insufficient metadata" keeps being the debt with the higher number of practitioners not sure about it. The possible reasons are explained in Section IV-A: practitioners have difficulties seeing this debt in their contexts and are not sure whether additional metadata is the right solution for the cases they observed. The number of practitioners who answered "not sure" increased more than for other debts. Again, this debt seems difficult to identify and estimate in the future and might concern architects more than developers, who are only involved with the development of microservices.

Companies A and C foresee cases in which they need additional metadata and consequently increase the probability of having this debt. On the other hand, Company B is mostly uncertain about this debt.

Main findings

F1. The practitioners seem to foresee an increase in microservice coupling. Microservice coupling might increase unnoticed in the early stages of migration, and suddenly become visible with many microservices.

F1. The practitioners foresee the use of shared libraries because they plan to use libraries to accelerate migration. Therefore, they may have to deal with the misuse of such libraries later.

F2. The practitioners accepted the extra costs of "Unnecessary settings." Therefore, they foresee the presence of such debt.

F3. The practitioners are most uncertain about to what extent "unplanned data sharing/synchronization" is a debt. They foresee the debt because they are unsure how to repay it.

C. WHICH MS-ATDs DO THE PRACTITIONERS FIND DIFFICULT TO SOLVE? (RQ3)

Figure 8 presents the percentage of practitioners who voted for each answer regarding MS-ATDs they did not know how to avoid. Most practitioners are not confident about solving the problem, which means that they do not know whether what they are using is a solution to prevent such debts in most cases.

The remainder of this section discusses the top-3 most found MS-ATDs extracted from the results for each company, according to our data and ranking calculation defined in Equation 7. The rankings are listed in Table 6. The MS-ATD on the top of the list is the debt that most practitioners do not know how to prevent. The values used to calculate the rankings are shown in Figure 9.

1) MISUSING SHARED LIBRARIES

"Misusing shared libraries" is in the top-3 for all companies in the ranking. None of the companies seemed to have considered good alternatives for shared libraries in the early stages of migration.

Company A reported that it is difficult to avoid these libraries because of the dependencies they have on the original architecture. These libraries ensure that all services

TABLE 6. Ranking of MS-ATDs that companies do not know how to avoid calculated through Equation 7. Each MS-ATD is associated with the same color to facilitate identifying it across the rankings from the distinct companies.

	Company A	Company B	Company C
1	Misusing shared libraries	Microservice coupling	Misusing shared libraries
2	Microservice coupling	Misusing shared libraries	Microservice coupling
3	Unplanned data sharing/synchronization	Unnecessary settings	Excessive diversity
4	Excessive diversity	Unplanned data sharing/synchronization	Unnecessary settings
5	Insufficient metadata	Insufficient metadata	Unplanned data sharing/synchronization
6	Inadequate use of APIs	Excessive diversity	Inadequate use of APIs
7	Unnecessary settings	Inadequate use of APIs	Insufficient metadata

FIGURE 9. Values for the calculation of the ranking of MS-ATDs the practitioners know how to avoid for each company.

behave in the same way when dealing with a centralized database and old pieces of software. Thus, they believe that this is difficult to deal with now, and their removal will be postponed.

Company B reported that the debt would increase because there is no plan to reduce the usage of shared libraries today. One participant from this company said, "*There is really no gold solution; everything is a trade-off.*" It is difficult for them to avoid using shared libraries.

Company C was not convinced about reducing the usage of shared libraries. When discussing the implementation of functionality as a service, one practitioner said, "*This will add latency everywhere.*" For him, this is unacceptable, and a shared library solves the problem without additional latency. However, companies with more mature microservice architectures have reported increased maintenance costs owing to the growing use of shared libraries [2]. Company C may have a different context from the companies studied by de Toledo *et al.* [2], and how they use shared libraries does not lead to problems. However, it is also possible that they are just down-prioritizing the debt, and they might incur high costs later. A more in-depth study would be useful to help companies such as Company C achieve a good trade-off between performance and maintainability while using shared libraries.

2) MICROSERVICE COUPLING

"Microservice coupling" is in the top-3 for all companies in the ranking. This result highlights that practitioners do not know how to properly prevent this debt from occurring. Thus, it is important to invest in training and techniques to prevent or reduce coupling in the early stages and, therefore, to reduce the growth of the debt interest.

Company A reported that coupling is difficult for practitioners to solve, and that more code reviews should help to reduce it.

Companies B and C reported that they did not have a good approach to solving this debt.

3) UNPLANNED DATA SHARING/SYNCHRONIZATION

"Unplanned data sharing/synchronization" is among the top-3 for Company A only.

Company A had a complex database that was difficult to split. Practitioners postponed splitting the database because it is challenging and costly to do so immediately.

Company B has this debt as fourth in its ranking. However, as shown in Figure 9b, there is no difference in its value with the third element in the ranking despite more votes for partial solutions (see Figure 8). Nevertheless, the differences are minimal (a small difference in uncertainty compared to

the third debt in the list), and this debt is almost as important as the third one in the list for Company B.

Several practitioners from Company C also voted for this debt as difficult to solve despite the existence of other debts with more votes (see Figure 8).

4) UNNECESSARY SETTINGS

''Unnecessary settings'' is in the top-3 for Company B only.

Company A did not discuss how it planned to reduce unnecessary settings in their services.

Company B had difficulties finding solutions for the increasing number of unnecessary settings, and considered the debt difficult to handle. One of the practitioners said, ''*Approaches to control the growth of these settings, such as peer review, might introduce time-demanding formal procedures, such as waiting for external teams' reviews and additional coordination*.'' Thus, Company B does not see good approaches to dealing with this issue, placing this debt as one of the top-3 in the list.

Company C has this debt as the fourth in the ranking, but it is as difficult as the third in the ranking (see Figure 9c).

5) EXCESSIVE DIVERSITY

''Excessive diversity'' is in the top-3 for Company C only.

For Company A, this debt is fourth in the ranking, but it is as difficult as the third in the same ranking (see Figure 9a). They only migrated a small part of the monolithic architecture to microservices and used the same .NET Core technology stack, reducing the diversity of technologies. However, they also reported having outsourced teams, and the new microservice architecture would allow other teams to use other technology stacks. The debt may worsen in the future.

Company B seems to have this diversity controlled, but it is not clear how they performed this control. The diversity of technology stacks for Company B stems from their various projects for different customers, but not from the same project with diverse microservices using different stacks.

Company C, on the other hand, reported that it is difficult to limit the technologies and languages without receiving complaints from teams already using a distinct set of technologies throughout the company. They accepted the debt now, but might need to develop company-level guidelines to keep it under control in the future. This seems to be a social rather than technical problem that is difficult to solve.

6) INSUFFICIENT METADATA

Answers for ''insufficient metadata'' carries a lot of uncertainty. Since practitioners have difficulties visualizing the solution in practice, they are not sure whether solving it is difficult.

Company A was the company in which most practitioners who voted for the debt were sure about it being difficult to solve (see Figure 8). Since Company A is also the company with the fewest microservices in the migration, it has had a hard time seeing cases where this debt applies.

Companies B and C seem to have more microservices that use messages in which the debt could apply. Thus, they seem to have a better overview of how the metadata could be implemented in their cases than Company A. However, they were uncertain about whether what they thought as a solution would help them solve the problem. Therefore, there is a large amount of uncertainty.

7) INADEQUATE USE OF APIs

The ''inadequate use of APIs'' is a debt with a high degree of uncertainty, but it is also one of the debts with more participants answering they know how to solve. Practitioners from all companies reported that this was a matter of education and training for creating good APIs. During our follow-up interviews, senior practitioners and architects reported knowing the good practices for developing APIs. They informed us that they were already investing in spreading knowledge on the subject throughout the companies. One possible interpretation of these results is that the companies still have practitioners learning about topics such as how to control API versioning and deprecation. Such cases generated some uncertainty in the teams, but they knew how to repay the debt.

Main findings

F1. Practitioners consider ''microservice coupling'' difficult to solve. Therefore, it is important to invest in training and techniques to prevent or reduce this debt early on in a project.

F2. Companies share libraries to reuse code from the original architecture. However, they do not consider the costs of misusing them in the early stages of migration, which might incur high costs later.

F3. ''Excessive diversity'' is a debt that is difficult to mitigate after practitioners start using distinct technologies. Having some agreement regarding the technologies to use in the early stages of migration would facilitate dealing with this debt later.

D. HOW IMPORTANT DO PRACTITIONERS PERCEIVE MS-ATDs? (RQ4)

Figure 10 and Table 7 show the importance of the MS-ATDs as perceived by the practitioners and calculated using Equation 8 for each company. Figure 10 groups the answers by MS-ATD and shows the percentage of votes per company. Table 7 presents the MS-ATDs ordered by the company.

There are clear differences among the companies, indicating that the importance of MS-ATDs is context dependent. For example, ''excessive diversity,'' the last in the rankings for Companies A and B, is in the top-3 for Company C. Two debts are consistently among the most important for all three companies: ''microservice coupling'' and ''unplanned data sharing/synchronization.'' The remainder of this section discusses each MS-ATD and possible reasons for these results.

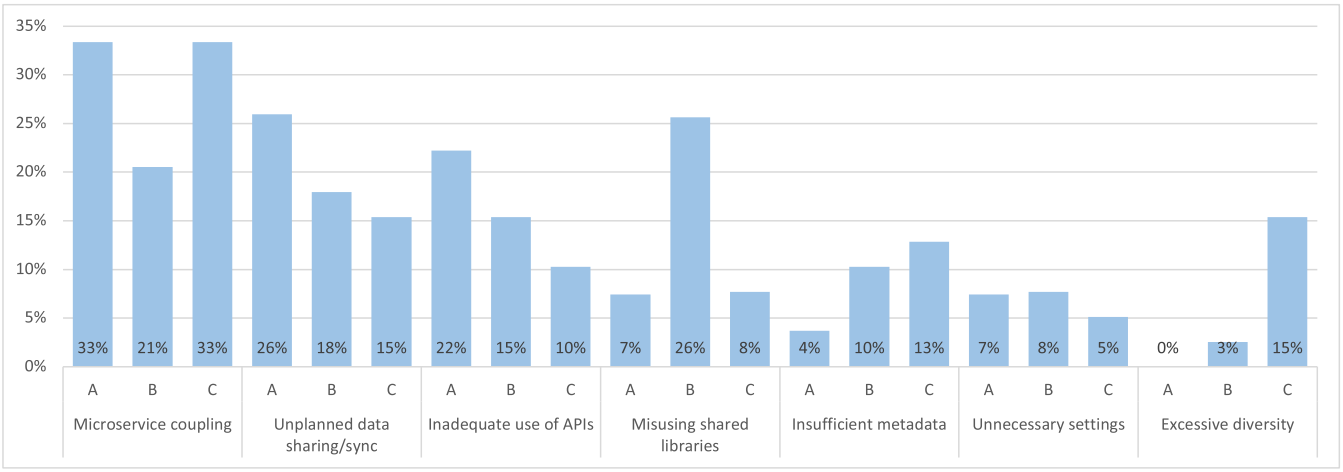


FIGURE 10. Importance of the MS-ATDs as perceived by the practitioners calculated by the Equation 8.

TABLE 7. Ranking of the most important MS-ATDs according to the practitioners calculated through Equation 8. Each MS-ATD is associated with the same color to facilitate identifying it across the rankings from the distinct companies.

	Company A	Company B	Company C
1	Microservice coupling	Misusing shared libraries	Microservice coupling
2	Unplanned data sharing/synchronization	Microservice coupling	Excessive diversity
3	Inadequate use of APIs	Unplanned data sharing/synchronization	Unplanned data sharing/synchronization
4	Unnecessary settings	Inadequate use of APIs	Insufficient metadata
5	Misusing shared libraries	Insufficient metadata	Inadequate use of APIs
6	Insufficient metadata	Unnecessary settings	Misusing shared libraries
7	Excessive diversity	Excessive diversity	Unnecessary settings

1) THE DEBTS THAT WERE IMPORTANT FOR ALL THREE COMPANIES

All three companies highly reported two MS-ATDs: "microservice coupling" and "unplanned data sharing/synchronization" (see Table 7). These debts are explicitly mentioned in the definition of microservices: low-coupled services with their own databases. One possible interpretation of these results is that practitioners may clearly identify debts against the definition of microservices.

As reported in Section IV-A, Company A reported that "microservice coupling" does not exist at present, and they do not see its costs. However, with the increasing complexity of the software, they foresee it coming (see Section IV-B) and recognize it as difficult to solve (see Section IV-C). On the other hand, Companies B and C have reported "microservice coupling" as found in the present, expected in the future, and difficult to solve. Therefore, this debt was expected to be considered important by practitioners.

Regarding "unplanned data sharing/synchronization," we found that all three companies reported it as important, but they have very distinct answers to the previous RQs. Company A postponed working on the database; thus, it is important to address this issue soon. Company B reported this debt for all previous rankings, but they considered other debts more important than this one. One possible interpretation of this result is that some of the practitioners in Company B believe that database sharing or synchronization approaches are adequate for their needs. In contrast, others are more skeptical of those approaches: they seem to discuss data sharing/synchronization approaches, but not all agree on doing so. Our method identified a disagreement among practitioners from the company. With this information, they could now discuss it internally. Company C, on the other hand, has this debt as the last for the first two rankings (found and foreseen) and as one of the last debts in the ranking of difficulty to solve. However, they consider this debt one of the three most important for them. One possible interpretation of the results from Company C is that they believe that this debt is important, but they have it under control. They shared databases and performed synchronizations among different services, but deliberately did so in situations they presumed they were right. Thus, they seemed to have planned these cases carefully.

2) THE DEBTS THE PRACTITIONERS MOSTLY DISAGREE

There are three debts for which the companies have distinct points of view: "misusing shared libraries," "inadequate use of APIs," and "excessive diversity." The causes of these differences may be the context of the projects or other unknown factors.

Company A has already reported the use of shared libraries to accelerate the development process (see Section IV-B). Company C reported that it was common to use such libraries. Companies A and C reported that it was challenging to remove misused shared libraries later. However, they do not consider this debt very important because they believe most of the shared libraries' use was correct. If this is the case, an in-depth study would help identify good practices while using shared libraries in microservices. If this is not the case, they may suffer from the costs of this debt in the future, as reported by companies with more mature microservice architectures [2]. On the other hand, Company B identifies ''misusing shared libraries'' as one of the most important debts to be mitigated.

''Inadequate use of APIs'' varies among companies. For Company A, this is one of the three most important debts; for Company B, it has medium importance; and for Company C, this debt is one of the three less important debts. In general, practitioners reported that it is important to have good APIs, but they did not make further comments regarding that specific debt, even though we asked.

Regarding the ''Excessive diversity,'' there are considerable differences among companies. No practitioner from Company A voted for this debt. Company A seems to have good control of diversity, which might explain the number of votes for that debt (see Figure 10). Company B reported that they have some diversity but that it is not important. They did not have any costly issues related to this debt. Finally, Company C reported a very distinct result compared to the other companies: this debt was one of the most important to them. Company C appear to be the company with the most diverse set of technologies. Some practitioners have reported concerns regarding their current policy of not limiting the technologies used by their microservices.

3) THE LESS IMPORTANT DEBTS

Other two debts were considered less important by practitioners: ''unnecessary settings'' and ''insufficient metadata.'' However, there are some differences among the companies. We discuss these differences below.

None of the three companies is concerned with the ''unnecessary settings'' because they report it as a problem that cannot be avoided in microservices. Company A was the only company to comment that might try to mitigate it in the future, without explaining how, since it is a future concern.

''Insufficient metadata,'' on the other hand, seems to be more context-dependent and had distinct votes from the different companies. The low number of votes might be related to the uncertainty in the previous responses (see Sections IV-A and IV-B).

E. HOW CAN COMPANIES PRIORITIZE WHICH MS-ATDs TO AVOID OR REPAY? (RQ5)

As mentioned in Equation 1, the risk of an MS-ATD is the probability of its occurrence multiplied by its negative impact. The more people foresee the debt, the more likely it is to happen, and the fewer people know how to solve the debt, the more likely it is to persist for a longer time. Therefore, we combine (multiply) the weighted sum of votes for the foreseen debts (defined in Equation 4) and the weighted sum of votes for the debts practitioners do not know how to solve (defined in Equation 6) as the probability of an MS-ATD to occur. Additionally, we used the importance given by the practitioners as the impact of the debt because it is more likely that practitioners consider important debts that have a more significant impact on their contexts. This interpretation is the basis for the calculation of Equation 5.

Figure 11 presents both the ranking calculated from Equation 10 and the normalized priority scores for each company and debt defined at Equation 11. Additionally, Figure 11 shows colors for the debts with high, medium, and low priorities. Table 8 presents another visualization of the ranking calculated from Equation 10 for each company.

Companies can use our ranking to prioritize their debts in their own contexts. Below, we present four examples to explain how our prioritization ranking can be helpful.

Based on the information presented by practitioners, our method found that microservice coupling is a major concern for all three companies. This debt received many votes from all companies as foreseen, difficult to solve, and important to avoid. Combining these votes into our ranking makes this debt stand out with the higher priority score among all the studied MS-ATDs. Other companies with mature microservice architectures have also reported this MS-ATD to be important [2]. Thus, the results of this study underscore that microservice coupling arises very early and should be addressed as soon as possible to prevent the debt from increasing. As the number of microservices increases, it becomes increasingly difficult to remove the coupling in microservices.

The next example shows how our method can capture contextual information regarding MS-ATDs. ''Excessive diversity'' is the last MS-ATD recommended for Companies A and B, but it is one of the three first debts recommended to be prioritized for Company C. Our qualitative data revealed that Company C had internal concerns regarding diversity, but the leaders did not want to enforce any limitations on the teams. The same concerns did not exist for Companies A and B. On the other hand, for Company C, our method captured internal concerns, allowing them to discuss how to address the issue internally.

In the next example, ''Insufficient metadata'' was identified as having a low priority. Practitioners were not convinced that this debt could be a problem in their projects. This debt involves understanding the big picture of the microservices, including their communication and the impact of the costs across many microservices and teams. Practitioners in the early stages of migration to microservices may not have enough information to be certain about this debt. They might not have a good overview of the architecture because they are focused on their own services, or they might not have enough microservices for this debt to be visible. Whatever the case,

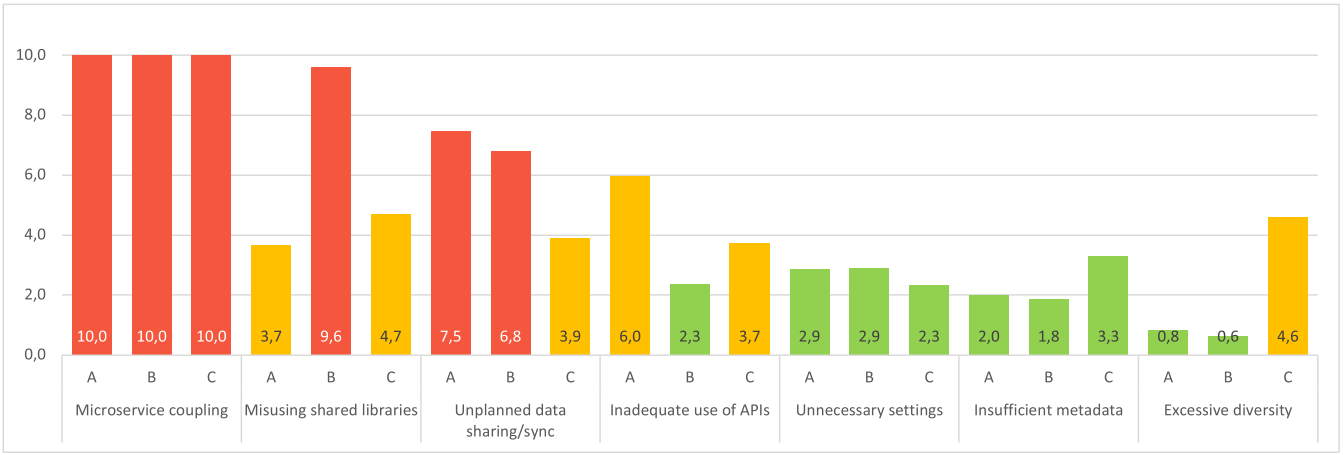


FIGURE 11. Priority ranking calculated through Equations 9 and 10, normalized between 1 and 10. Red bars indicate debts with high priority. Orange bars indicate debts with medium priority. Green bars indicate debts with low priority.

TABLE 8. Priority rankings of the proposed MS-ATDs calculated through Equation 10.

	Company A	Company B	Company C
1	Microservice coupling	Microservice coupling	Microservice coupling
2	Unplanned data sharing/synchronization	Misusing shared libraries	Misusing shared libraries
3	Inadequate use of APIs	Unplanned data sharing/synchronization	Excessive diversity
4	Misusing shared libraries	Unnecessary settings	Unplanned data sharing/synchronization
5	Unnecessary settings	Inadequate use of APIs	Inadequate use of APIs
6	Insufficient metadata	Insufficient metadata	Insufficient metadata
7	Excessive diversity	Excessive diversity	Unnecessary settings

there is much uncertainty, and this debt does not seem to be sufficiently significant at the current stage of development.

Finally, as the last example, "Misusing shared libraries" was recommended with high priority to Company C, despite only a small percentage of practitioners having voted for it as important 10. This is an interesting result because our method found reasons to believe this debt could be a problem and properly warned practitioners of such concerns. This debt is the most foreseen (see Table 5) and difficult to solve (see Table 6) for Company C. There is a high probability that this debt will arise and have high costs if not properly mitigated. The costs reported by mature companies regarding this debt are considerable [2], and the practitioners were properly warned and may discuss it internally.

F. OVERALL REMARKS

Practitioners tend to ignore "unnecessary settings." They believe that this debt cannot be removed while using microservices and they just accept it. A similar circumstance happens with the "excessive diversity," another debt practitioners do not give much importance. When the debt arrives, it is difficult to repay because practitioners might resist not using the technologies they have adopted until the present date.

Other debts are deliberately taken during the migration to microservices: "unplanned data sharing/synchronization" and "misusing shared libraries." Several practitioners have reported that using existing databases instead of creating new ones for the microservices may accelerate the migration process. Therefore, they neither plan to have separate databases nor plan the data sharing adequately. There has been less discussion regarding the synchronization of separated databases because they mostly share it to avoid synchronization issues. Regarding the debt "misusing shared libraries," practitioners hardly believe that the libraries they use are misused. Still, they use them to accelerate the development process and avoid dealing with latency across services. Practitioners rarely think about the number of libraries being used, nor track down the libraries that have constant updates and might block several teams on the project. Practitioners should consider alternatives and use shared libraries when none of the other options is feasible.

Debts such as "insufficient metadata" are difficult to identify. A possible reason is that this is a context-dependent issue or that a broader view of the project is required, usually shared by its architects, and frequently overlooked by microservice developers, who are more concerned with the single services they are developing. Such a situation is both an advantage and

a disadvantage of the microservice architecture: practitioners may focus on developing their own microservices without worrying about the work of other teams; however, they lose the overview of the big picture of the project.

''Inadequate use of APIs'' is a debt easy to remove and that is always in the middle of our rankings: it is never the most found or foreseen debt, but it is also never in the bottom of the rankings. Therefore, it is a common debt of medium importance. This debt can be mitigated through good practices in developing APIs.

Finally, ''microservice coupling'' is one of the most important issues that need to be avoided by teams. Our previous study [2] also encountered the problem of coupling in more mature microservice projects than those studied here. One possible cause of coupling might be insufficient experience by teams on how to delimit microservice boundaries, but other causes may also be worth investigating.

The companies found our results useful in helping them address MS-ATDs in their contexts. As future work, it would be interesting to draw a threshold in our priority ranking above which the debts should be fixed and below which it is safe to postpone debt repayment.

G. MODELING UNCERTAINTY

As presented in Table 3, some practitioners answered ''not sure'' for the MS-ATDs found (RQ1) and foreseen (RQ2). In our analysis (Section III-D), we considered those answers to have a 50% probability of the MS-ATD occurrence. Such a decision entails that two ''not sure'' count as one ''yes.'' However, practitioners may have meant more or less than 50%. Although we do not have more information to model this uncertainty, we could have chosen other more or less conservative weights. Here, we discuss the changes in our prioritization when ''not sure'' is assigned such other weights.

Other areas of research, such as health research (see, for example, the results for the Behavioral Risk Factor Surveillance System, BRFSS, a health-related survey from U.S. residents [28]), completely ignore the uncertainty in their analysis. On the other hand, in our case, the practitioners wanted to express a chance of the debt to happen, although they were not entirely sure. In other words, they expressed a chance, a probability that leads us to the next subject to discuss: the value of the probability. Nevertheless, we want to see what would change if we use this approach in our prioritization. Therefore, we compile the results where ''not sure'' is given a value of 0.

In addition to the previous consideration, we want to see what would change if we consider that the participants are almost sure about the existence of the debt. Therefore, we used a probability of 0.8. Similarly, we want to see the changes if we consider that participants are skeptical about the existence of the debt. For the last case, we used a probability of 0.2. Table 9 shows the changes in all our rankings, including the final prioritization proposal, for all these values compared to the value of 0.5, which we used in our previous analysis. The numbers indicate the changes in the positions in the ranking. The arrows indicate the direction of the change, up or down, in the ranking.

As shown in Table 9, most of the changes are plus or minus 1, and there are even fewer changes in the final prioritization rankings. Therefore, our approach is not highly affected by changes in the uncertainty probabilities. Since we only collected which participants were unsure, we believe that 0.5 would give us a better chance to balance out skeptical and pessimistic opinions from all possible values. The construction of the final prioritization ranking required us to select one of these values. Otherwise, showing different rankings with different probabilities would confuse participants.

As a future improvement, we suggest using a Likert scale to gather more precise data on practitioners' uncertainty. One could model values or categories from ''very improbable'' to ''very probable.'' Additionally, a ''do not know'' option would help to exclude invalid answers to RQ1 and RQ2, if any. However, these additions require caution because they demand more attention and time from practitioners on each debt. Therefore, they might increase the participants' response times and affect their participation ratings in the questionnaire.

H. IMPLICATIONS FOR RESEARCH AND PRACTICE

We received positive feedback from practitioners, indicating that our approach is helpful in decision-making regarding the prioritization of MS-ATDs. A few examples: ''We would like to try it again with a more mature project.'' (Company A); ''We are of course interested in going deep. (...) [We need] a serious internal discussion among the developers, architects, and management.'' (Company B). Company C is going to adjust its priorities based on the MS-ATDs we discussed and asked for additional information on the debts.

Our prioritization method is lightweight and can be applied periodically by companies to manage the risk of ATD during the development process, for example, within an agile architecture framework such as CAFFEA [29]. The risk of a debt changes according to project needs and should be re-evaluated. The requirements for this method are to have a list of MS-ATDs that can be discussed (for example, as in this study, one can use an existing catalog [2]) and one facilitator (researcher or practitioner) who is familiar with the debts and can present them to the remaining participants, collect the answers, and compute the rankings.

Researchers may apply the prioritization method in this study to investigate ATDs from other studies. We envision that the technique can be applied not only for MS-ATDs, but also for any ATD within an organization. However, more research is needed. Researchers also have the raw data available to facilitate comparisons with their own findings. Additionally, there are several suggestions for future and in-depth work that might be useful for research purposes, such as supporting the early identification of microservice coupling, shared libraries, and database sharing and synchronization, which are some of the most costly debts in our

TABLE 9. The changes in the rankings when using different probabilities (0, 0.2, and 0.8) for the uncertainty ("not sure") answers, in comparison with the results with probability 0.5, for each MS-ATD.

Debt	Company	MS-ATDs encountered (see Table 4)			MS-ATDs foreseen (see Table 5)			Final prioritization ranking (see Table 8)		
		0	0.2	0.8	0	0.2	0.8	0	0.2	0.8
Insufficient metadata	A									
	B				1 ↓	1 ↓				
	C				3 ↓	1 ↓	1 ↑			
Microservice coupling	A									
	B			2 ↑				1 ↓	1 ↓	
	C			2 ↑	2 ↑	1 ↑	1 ↑			
Inadequate use of APIs	A	1 ↑	1 ↑	1 ↑						
	B				1 ↓	1 ↓				
	C				1 ↑	1 ↑		1 ↑	1 ↑	
Excessive diversity	A									
	B				1 ↑	1 ↑				
	C									
Unplanned data sharing/synchronization	A	1 ↑	1 ↑	1 ↑			1 ↑			
	B			1 ↑			1 ↑			
	C			1 ↑				1 ↓	1 ↓	
Misusing shared libraries	A	1 ↓	1 ↓	1 ↓						
	B	1 ↑	1 ↑	2 ↓			1 ↓	1 ↑	1 ↑	
	C			2 ↓			1 ↓			
Unnecessary settings	A	1 ↓	1 ↓	1 ↓			1 ↓			
	B	1 ↓	1 ↓	1 ↓	1 ↑	1 ↑				
	C			1 ↓		1 ↓	1 ↓			

list, or investigating the benefits of using our prioritization approach continuously in agile development processes.

V. LIMITATIONS

Given the availability constraints of companies and practitioners, we used a subset of MS-ATDs identified in a previous study. Therefore, the final prioritization results may be partial. Still, we selected the most common MS-ATDs found in seven companies experienced with microservices from de Toledo *et al.* [2]: we considered them also to be the most likely to be relevant to the companies participating in this new study. In addition, the practitioners in this study acknowledged the importance of the selected MS-ATDs during the interviews. The companies involved can also follow a similar procedure to prioritize additional debts and enrich the existing prioritization list. In this case, there is no need to collect the answers for RQ1 again, RQ2, and RQ3 for the MS-ATDs investigated previously.

The prioritization procedure presented here is based on our interpretation of the research context. The value 0.5, used for the answers "not sure" and "partially" impacts our results, especially for responses that are mainly composed of those options. Our interpretation of this result is that, since it is not possible to obtain a better approximation of the uncertainty or the partiality of a solution, the probability of the value being close to 0.5 is high because some practitioners might be very uncertain. By contrast, others might be very sure of their answers. Therefore, on average, the values tend to be 0.5. Further studies could attempt to obtain a more precise estimation of this probability.

The practitioners might not know the debts or might have a different understanding of a specific debt (e.g., shared databases might be understood as coupling by some practitioners and as a very distinct debt by others), which might affect their answers regarding the existence of the debt, or whether they know how to solve it. To reduce this threat, we presented and described each MS-ATD before asking questions. The practitioners were asked to provide their answers immediately after the explanation. We also collected information on practitioners' satisfaction with our descriptions and interpretations. Only three interviewees reported difficulties in understanding our explanations for one question each.

It was also challenging to interpret the situations in which the participants answered: "not sure" or "partially." We considered those as a 50% probability of the answer being

"yes." However, this interpretation may not be accurate. Practitioners and researchers should interpret these rankings cautiously. For example, the second element in the priority ranking might be the best to start with for a specific company. However, such an interpretation is a reasonable approximation of all the answers because some practitioners have more knowledge than others and some solutions are less partial than others. Therefore, we considered that, on average, the answers converged to 50%. Our rankings are suggestions for discussion by practitioners and researchers. The raw data are provided in Table 3 for further interpretation.

Finally, it is possible that other companies not included in our study have successfully implemented a different prioritization or refactoring strategy and applied it to different MS-ATDs than those discussed in this work. However, none of the companies in this or previous studies reported a similar prioritization approach. We performed a lightweight literature review at the beginning of this study, but we did not find any other strategies for prioritizing MS-ATDs. Our work is exploratory and can be considered a starting point for future research. More studies are necessary to investigate additional MS-ATDs and to further validate and potentially improve our approach.

VI. RELATED WORK

Lenarduzzi *et al.* [8] monitored the evolution of code TD in a small to medium-sized company that migrated a monolithic system to microservices. They concluded that the migration reduced the overall code TD. The authors did not study architectural TD, which was the focus of this study.

Taibi *et al.* [7] defined a taxonomy of 20 microservice anti-patterns, based on 27 interviews with experienced practitioners. Several of these anti-patterns are related to migration: "no DevOps tools," "too many technologies" (the same as excessive diversity in our paper), "I was taught to share" (which we defined as misusing shared libraries), "static contract pitfall" (i.e., APIs that are not properly versioned), "mega-service," "shared persistence" (part of our unplanned data sharing or synchronization), "sloth" (too many coupling among the microservices, resulting in a distributed monolith), and "trying to fly before you can walk" (i.e., migrating to microservices while the practitioners lack the necessary skills). In our study, we considered the overlapping anti-patterns to be MS-ATDs. (For discussing the relationship between architectural anti-patterns and ATDs, see de Toledo *et al.* [2]). Additionally, we created rankings for the identified MS-ATDs and investigated how to prioritize them.

Martini *et al.* [30] identified and prioritized ATDs through architectural smells in a multiple case study. They analyzed four software projects in the same company. Their approach consisted of using a tool called Arcan [31] to identify architectural smells leading to ATD and then asking the practitioners to prioritize the debts found. Their approach was limited to ATDs that can be identified through architectural smells. On the other hand, our approach was tested with microservice-specific ATDs and used a risk-based approach for prioritization, combining information from the present, the foreseen future, and the importance given by practitioners.

Pigazzini *et al.* [32] presented an approach also supported by Arcan [31] to identify candidate microservices in monolithic Java projects. The approach begins with the identification of architectural smells that can hinder migration. Despite their relation to migration, the identified anti-patterns were detected in the monolithic architecture before migration. By contrast, we only considered ATDs in microservices during and after migration.

Panichella *et al.* [33] proposed metrics to compute and visualize the coupling between microservices, which may help evaluate the cost of coupling in a given context. Their approach can help practitioners find and measure microservice coupling, but it is focused on a single MS-ATD and does not propose ways to prioritize debts. Our case study highlights situations in which practitioners might not have promising approaches for measuring microservice coupling. The method proposed by Panichella *et al.* [33] may be useful in such cases.

VII. CONCLUSION

This paper presents our investigation of how ATD issues specific to microservices, MS-ATDs, accumulate in three companies that are migrating to such an architectural style. We carefully conducted our study with the practitioners by giving them presentations on the MS-ATDs and double-checking that they were on the same page with the explanation of the MS-ATDs. We then asked practitioners which MS-ATDs they considered difficult to remove (tackle) and which ones were the riskiest in the present and future of their projects. We discussed the answers given by the practitioners considering their contexts and created rankings for the most found, foreseen, difficult to solve, and important MS-ATDs for each of the participating companies. We also proposed an approach to prioritize MS-ATDs based on risk and discussed it with participants. The participants also reported that the results and the prioritization method were useful and may be used in their contexts. For example, participants from one company said that our approach would help people become aware of the issues they have not yet seen. Thus, they are be able to take action to mitigate the negative consequences (interest) of the debts.

Our most important findings related to individual MS-ATDs were as follows: (i) "Misusing shared libraries" is a common debt during migration to microservices. Companies start using shared libraries because of the convenience of reusing code from their original architectures. However, these companies may be good candidates for high costs due to misuse of such libraries in the future, as identified in previous studies [2], [7]. (ii) It is common for companies to share databases during the early stages of migration to microservices to accelerate the migration process. (iii) Microservice coupling occurred frequently, possibly related to practitioners' lack of experience in creating microservices. However, they postpone the discussion of the debt to the late stages,

possibly because the costs of the debt are small at the beginning of the migration. One possible interpretation of this result is that practitioners might not have proper ways or tools to measure the growth of microservice coupling.

Given that "Misusing shared libraries" and "Microservice coupling" were common MS-ATDs in our study, other organizations may also consider them. The procedure of creating rankings reported in this paper may help other organizations prioritize fixing or avoiding MS-ATDs during migration before their costs increase. Overall, we believe that our results will help understand the relationship between ATDs and microservices.

Our study is fully replicable by researchers. The raw data are also available, allowing researchers to use such data in their approaches and facilitating comparison of the results.

Future work includes: running in-depth studies on the companies participating in this study to understand the consequences of their current architectural decisions; analyzing other companies migrating to microservices; investigating possible metrics; quantifying debts and costs; exploring deeper the aspects that practitioners emphasize when they consider an MS-ATD important.

REFERENCES

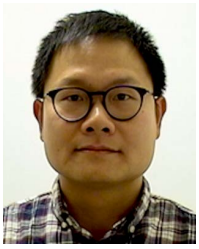
[1] M. Fowler. (Jul. 2015). *Microservice Trade-Offs*. [Online]. Available: https://Martinfowler.com/articles/microservice-trade-offs.html
[2] S. S. de Toledo, A. Martini, and D. I. K. Sjøberg, "Identifying architectural technical debt, principal, and interest in microservices: A multiple-case study," *J. Syst. Softw.*, vol. 177, Jul. 2021, Art. no. 110968.
[3] T. Besker, A. Martini, and J. Bosch, "A systematic literature review and a unified model of ATD," in *Proc. 42th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Aug. 2016, pp. 189–197. [Online]. Available: http://ieeexplore.ieee.org/document/7592796/
[4] B. Foote and J. Yoder, "Big ball of mud," in *Proc. 4th Patterns Lang. Program. Conf. (PLoP)*, N. Harrison, B. Foote, and H. Rohnert, Eds. Monticello, IL, USA: Addison-Wesley, 2000, pp. 653–692.
[5] J. Lewis and M. Fowler. (2014). *Microservices: A Definition of This New Architectural Term*. [Online]. Available: https://www.Martinfowler.com/articles/microservices.html
[6] V. Bushong, A. S. Abdelfattah, A. A. Maruf, D. Das, A. Lehman, E. Jaroszewski, M. Coffey, T. Cerny, K. Frajtak, P. Tisnovsky, and M. Bures, "On microservice analysis and architecture evolution: A systematic mapping study," *Appl. Sci.*, vol. 11, no. 17, p. 7856, Aug. 2021.
[7] D. Taibi, V. Lenarduzzi, and C. Pahl, "Microservices anti-patterns: A taxonomy," in *Microservices: Sci. Eng.*, A. Bucchiarone, N. Dragoni, S. Dustdar, P. Lago, M. Mazzara, V. Rivera, and A. Sadovykh, Eds. Cham: Springer International Publishing, 2020, pp. 111–128.
[8] V. Lenarduzzi, F. Lomio, N. Saarimäki, and D. Taibi, "Does migrating a monolithic system to microservices decrease the technical debt?" *J. Syst. Softw.*, vol. 169, Nov. 2020, Art. no. 110710.
[9] J. Bogner, T. Boceck, M. Popp, D. Tschechlov, S. Wagner, and A. Zimmermann, "Towards a collaborative repository for the documentation of service-based antipatterns and bad smells," in *Proc. IEEE Int. Conf. Softw. Archit. Companion (ICSA-C)*, Mar. 2019, pp. 95–101.
[10] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," *J. Syst. Softw.*, vol. 150, pp. 77–97, Apr. 2019.
[11] P. D. Francesco, I. Malavolta, and P. Lago, "Research on architecting microservices: Trends, focus, and potential for industrial adoption," in *Proc. IEEE Int. Conf. Softw. Archit. (ICSA)*, Apr. 2017, pp. 21–30.
[12] T. Besker, A. Martini, and J. Bosch, "The pricey bill of technical debt: When and by whom will it be paid?" in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2017, pp. 13–23.
[13] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *J. Syst. Softw.*, vol. 101, pp. 193–220, Mar. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121214002854
[14] S. Newman, *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*, 1st ed. Sebastopol, CA, USA: O'Reilly Media, 2019.
[15] P. Di Francesco, P. Lago, and I. Malavolta, "Migrating towards microservice architectures: An industrial survey," in *Proc. IEEE Int. Conf. Softw. Archit. (ICSA)*, Apr. 2018, pp. 29–38.
[16] O. Zimmermann, "Mircroservices tenets: Agile approach to service development and deployment," *Comput. Sci.-Res. Develop.*, vol. 32, nos. 3–4, pp. 301–310, Jul. 2017. [Online]. Available: http://link.springer.com/10.1007/s00450-016-0337-0
[17] F. Montesi and J. Weber, "Circuit breakers, discovery, and API gateways in microservices," 2016, *arXiv:1609.05830*.
[18] P. Niblett and S. Graham, "Events and service-oriented architecture: The oasis web services notification specification," *IBM Syst. J.*, vol. 44, no. 4, pp. 869–886, 2005. [Online]. Available: https://ieeexplore.ieee.org/document/5386704
[19] F. Rademacher, S. Sachweh, and A. Zundorf, "Differences between model-driven development of service-oriented and microservice architecture," in *Proc. IEEE Int. Conf. Softw. Archit. Workshops (ICSAW)*, Apr. 2017, pp. 38–45. [Online]. Available: https://ieeexplore.ieee.org/document/7958454
[20] J. W. Yoder and P. Merson, "Strangler patterns," in *Proc. 27th Conf. Pattern Lang. Programs*, 2020, p. 25. [Online]. Available: https://hillside.net/plop/2020/papers/yoder.pdf
[21] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing technical debt in software engineering (Dagstuhl Seminar 16162)," *Dagstuhl Rep.*, vol. 6, no. 4, pp. 110–138, 2016.
[22] N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord, and I. Gorton, "Measure it? Manage it? Ignore it? Software practitioners and technical debt," in *Proc. 10th Joint Meeting Found. Softw. Eng.*, New York, NY, USA, Aug. 2015, pp. 50–60.
[23] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *IEEE Softw.*, vol. 29, no. 6, pp. 18–21, Nov. 2012.
[24] Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. Q. Da Silva, A. L. Santos, and C. Siebra, "Tracking technical debt—An exploratory case study," in *Proc. 27th IEEE Int. Conf. Softw. Maintenance (ICSM)*, Sep. 2011, pp. 528–531.
[25] A. Martini and J. Bosch, "An empirically developed method to aid decisions on architectural technical debt refactoring: AnaConDebt," in *Proc. IEEE/ACM 38th Int. Conf. Softw. Eng. Companion (ICSE-C)*, May 2016, pp. 31–40.
[26] R. K. Yin, *Case Study Research and Applications: Design and Methods*, 6th ed. Newbury Park, CA, USA: Sage, 2018.
[27] W. S. Cleveland, *The Elements Graphing Data*. Monterey, CA, USA: Wadsworth Advanced Books and Software, 1985.
[28] Centers for Disease Control and Prevention. (Jul. 2020). *Calculated Variables in the 2019 Behavioral Risk Factor Surveillance System (BRFSS)*. Department of Health and Human Services. [Online]. Available: https://www.cdc.gov/brfss/annual_data/2019/pdf/2019-calculated-variables-version4-508.pdf
[29] A. Martini and J. Bosch, "A multiple case study of continuous architecting in large agile companies: Current gaps and the CAFFEA framework," in *Proc. 13th Work. IEEE/IFIP Conf. Softw. Archit. (WICSA)*, Apr. 2016, pp. 1–10.
[30] A. Martini, F. A. Fontana, A. Biaggi, and R. Roveda, "Identifying and prioritizing architectural debt through architectural smells: A case study in a large software company," in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Lecture Notes in Computer Science), vol. 11048, Madrid, Spain: Springer, 2018, pp. 320–335.
[31] F. A. Fontana, I. Pigazzini, R. Roveda, D. Tamburri, M. Zanoni, and E. Di Nitto, "Arcan: A tool for architectural smells detection," in *Proc. IEEE Int. Conf. Softw. Archit. Workshops (ICSAW)*, Apr. 2017, pp. 282–285. [Online]. Available: http://ieeexplore.ieee.org/document/7958506/
[32] I. Pigazzini, F. Arcelli Fontana, and A. Maggioni, "Tool support for the migration to microservice architecture: An industrial case study," in *Software Architecture* (Lecture Notes in Computer Science), vol. 11681. Cham, Switzerland: Springer, Sep. 2019, pp. 247–263.
[33] S. Panichella, M. Rahman, and D. Taibi, "Structural coupling for microservices," in *Proc. 11th Int. Conf. Cloud Comput. Services Sci.*, 2021, pp. 280–287.

SAULO SOARES DE TOLEDO received the B.Sc. degree in computer science from the Federal University of Campina Grande, Paraíba, Brazil, the M.Sc. degree in computer science from the Federal University of Campina Grande, in 2016, and the Licentiate degree in informatics from the State University of Paraíba, Paraíba. He is currently pursuing the Ph.D. degree with the University of Oslo, Norway, focusing on architectural technical debt in microservices and collaborating with several large European companies.

He is also an Experienced Software Developer, who has worked in several roles, including as a Full-Stack Developer, a Tech Lead, and a Software Architect, mainly with agile teams. His research interests include technical debt, software architecture, and microservices.

ANTONIO MARTINI (Member, IEEE) received the M.Sc. degree from the University of Parma, Italy, in 2011, and the Ph.D. degree from the Chalmers University of Technology, Gothenburg, Sweden, in 2015.

He is currently an Associate Professor with the University of Oslo, Norway. He is also a part-time Researcher with the Chalmers University of Technology. His experience covers software engineering and management in several contexts, such as large, embedded software companies, small, web companies, business to business companies, and startups. He has also received several funds for the commercialization transfer to practice of research results. He has worked as a Principal Strategic Researcher with CA Technologies (now Broadcom) for a technology transfer project in the context of a Marie Curie EU project related to the Tecniospring+ program. He has led a ten year project focusing on managing technical debt, collaborating with several large companies in Northern and Central Europe. He is also a Scientific Advisor for a startup and a Creator of a tool to manage technical debt. He also worked as a Software Developer and an Independent Consultant. He has published more than 50 contributions to top software engineering conferences and journals. His research interests include technical debt, software architecture, technical leadership, and agile software development. He is active in the research community as an editor and a reviewer for top IEEE software engineering conferences and journals, such as IEEE Software and IEEE Transactions on Software Engineering. He is involved in the organization of IEEE conferences, such as TechDebt @ ICSE, ICSA, and SEAA Euromicro.



PHU H. NGUYEN received the B.Sc. degree from the Hanoi University of Science and Technology, Vietnam, in 2005, the M.Sc. degree from the Eindhoven University of Technology, The Netherlands, in 2010, and the Ph.D. degree from the University of Luxembourg, Luxembourg, in 2015.

He is currently a Researcher at SINTEF, member of the Trustworthy Green IoT Software Group, developing new tools and methodologies for software development and operation of intelligent and trustworthy systems spanning across the IoT, edge, and cloud continuum, with a particular focus on sustainability. He has experience from working in international research projects as well as research and development projects with industry in Norway. He is an active reviewer and an organizer/PC member of high-impact journals, conferences, and workshops. He was awarded a certificate for exceptional contributions, support, and commitment in the organization of the Sixth IEEE International Conference on Software Testing, Verification, and Validation (ICST 2013), IBM Award for displaying exceptional personal dedication, teamwork, and contribution to the ibm.com project 2007, and the first prize at the LuxDoc Science Slam 2014 for communicating research to public.

DAG I. K. SJØBERG (Member, IEEE) received the M.Sc. degree in computer science from the University of Oslo, in 1987, and the Ph.D. degree in computing science from the University of Glasgow, in 1993.

Since 1999, he has been a Full Professor in software engineering with the University of Oslo. He has five years of industry experience as a Developer and a Group Leader. He has been the co-founder of four start-up companies. In 2001, he formed the Simula Research Laboratory, Software Engineering Department, and was its leader, until 2008. He has been an Associate Editor of the IEEE Transactions on Software Engineering and Empirical Software Engineering. His research interests include the software life cycle, including agile and lean development processes, software quality, skill assessment, and empirical research methods in software engineering.