Summary: Identifying architectural technical debt, principal and interest in microservices – A multiple-case study[1]

Saulo S. de Toledo[2], Antonio Martini[3] and Dag I.K. Sjøberg[4]

[2]*University of Oslo, Oslo, Norway*
[3]*University of Oslo, Oslo, Norway*
[4]*University of Oslo, Oslo, Norway*

Abstract

Background: Using a microservices architecture is a popular strategy for software organizations to deliver value to their customers fast and continuously. However, scientific knowledge on how to manage architectural debt in microservices is scarce.

Objectives: In the context of microservices applications, this paper aims to identify architectural technical debts (ATDs), their costs, and their most common solutions.

Method: We conducted an exploratory multiple case study by conducting 25 interviews with practitioners working with microservices in seven large companies.

Results: We found 16 ATD issues, their negative impact (interest), and common solutions to repay each debt together with the related costs (principal). Two examples of critical ATD issues found were the use of shared databases that, if not properly planned, leads to potential breaks on services every time the database schema changes and bad API designs, which leads to coupling among teams. We identified ATDs occurring in different domains and stages of development and created a map of the relationships among those debts.

Conclusion: The findings may guide organizations in developing microservices systems that better manage and avoid architectural debts.

Keywords

Cost of software, Cross-company study, Software quality, Software maintainability, Qualitative analysis

1. Introduction

The microservices architectural style is becoming increasingly popular in the industry. Microservices are small and independent components, each with a single responsibility and developed for scalability [1]. Despite several advantages, microservices are still an emerging technology. There are drawbacks, and companies are still learning how to migrate their previous solutions to microservices properly [2]. The challenges involved in such an architectural style lead to

[1]Use the original publication when citing this work: S. S. de Toledo, A. Martini, D. I. K. Sjøberg, Identifying architectural technical debt, principal, and interest in microservices: A multiple-case study, in: Journal of Systems and Software, vol. 177, Jul. 2021. URL: https://doi.org/10.1016/j.jss.2021.110968.

15th European Conference on Software Architecture (ECSA 2021), 13-17 September 2021 (virtual)

saulos@ifi.uio.no (S. S. de Toledo); antonima@ifi.uio.no (A. Martini); dagsj@ifi.uio.no (D. I.K. Sjøberg)
0000-0002-0747-4052 (S. S. de Toledo); 0000-0002-0669-8687 (A. Martini)

Architectural Technical Debt (ATD), a metaphor used to describe architectural sub-optimal decisions that have a benefit in the short term but increase the overall costs in the long run [3].

We conducted a multiple-case study in seven international Europe-based companies to investigate ATD in some of their microservices projects through the following research questions:

- **RQ1:** What are the most critical ATD issues in microservices?
- **RQ2:** What are the negative impacts of such ATD issues?
- **RQ3:** What are possible solutions to repay or avoid such ATD issues?

We aimed to support practitioners' decision-making in projects involving microservices. This article is a continuation of a previous single case study [4] by adding six additional companies, reorganizing the originally proposed five debts, and adding several other debts.

2. Background

2.1. Microservices

The most accepted definition of the microservices architecture was proposed by Lewis and Fowler [5]: *an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API* [5]. Microservices may also be considered a way of implementing Service Oriented Architecture (SOA) [6]. They have both advantages, such as being easier to scale and having shorter cycles for testing, build and release, and disadvantages, such as the additional operational complexity [7].

2.2. Architectural Technical Debt

Architectural Technical Debt (ATD) is a type of technical debt (TD) related to software architecture. ATD is considered the most challenging type of TD to be unveiled and managed, mostly because of the lack of research and tool support [8]. As any TD, the term involves three main concepts: the *debt*, which is the sub-optimal solution with short-term benefits, but that generate future additional costs; the *interest*, which is the extra cost generated by the debt; and the *principal*, that is the cost of developing a solution that avoids the debt or the cost of refactoring such debt [9]. Accumulating the debt might be useful in some circumstances [10]. Understanding the debt and the costs involved might help in the decision-making process.

- **Debt:** The debt is the sub-optimal solution with short-term benefits, but that generates future additional costs (the interest). A solution that was suitable before might also become a debt later.
- **Interest:** The interest is the extra cost that must be paid because of a debt or the amount that will be saved if there is no such debt.
- **Principal:** The principal is the cost of developing a solution that avoids the debt or the cost of refactoring such debt.

Accumulating the debt might be useful in some circumstances [10]. Understanding the debt and the costs involved might help in the decision-making process.

Table 1
Companies context

Context	Company						
	A	**B**	**C**	**D**	**E**	**F**	**G**
Application domain	Finance	Cloud	IoT	Health	Public services	Transport	Transport
Approx. num. employees	> 30000	> 7000	> 30000	> 30000	> 30000	320	> 20000
Approx. num. employees on IT	> 2000		20000	1200	250	150	
Approx. num. employees in the case	2000	200	500	250	150	150	
Approx. num. unique microservices	1000	50	80	40	400	600	3000
Age of the product	>10 years	>2 years	>2 years	>1.5 years	>10 years	>4 years	>10 years

3. Methodology

This study identified the most common and critical ATD issues in projects using microservices. We also identified the interests and principals related to those ATDs. We conducted an *exploratory multiple-case study* in seven different software products, each one developed in a distinct company in a different context. Table 1 shows a summary of the studied companies and project contexts. We identify the companies by letters from *A* to *G*.

We prepared an interview guide and performed 25 interviews with 22 employees in different roles. New aspects emerged as we progressed with the interviews. Therefore, we updated the interview guide and updated the missing details from previous interviews during complementary interviews with the previous companies.

4. Results and Conclusions

We found 16 ATDs and discussed their interest and principal. Table 2 presents the ATDs found and the respective companies. We also discussed the interests and principals of each of those ATDs. Distinct companies have different ATDs. Not all companies have the solution for the issues found. Thus, the experiences from other companies might be helpful for them. Some ATDs seem to be more common than others.

Some of the ATD found caused substantial interest. Some were related to cascading breaks, unnecessary complexity, coupling, and dependencies among teams. Our results should help practitioners manage ATD by learning from the experience of other companies.

Table 2
The ATDs found and the respective companies

ID	Debt	Companies
1.	Insufficient metadata in the messages	
1.1.	Insufficient message traceability	A, E
1.2.	Poor dead letter queue growth management	A, E
2.	Microservice coupling	A, B, C, E, F
3.	Lack of communication standards among microservices	A
4.	Inadequate use of APIs	
4.1.	Poor RESTful API design	B, C, D
4.2.	Use of complex API calls when messaging is a simpler solution	D
5.	Use of inadequate technologies to support the microservices architecture	A, C
6.	Excessive diversity or heterogeneity in the technologies chosen across the system	A, E, F, G
7.	Manual per service handling of network failures when target services are unavailable	B, C
8.	Unplanned data sharing and synchronization among services	
8.1.	Sharing persistence or database schema	C, D, G
8.2.	Unplanned database synchronization	C
9.	Use of business logic in communication among services	A
10.	Reusing third-party implementations	
10.1.	Many services using different versions of the same internal shared libraries	A, D, E, F
10.2.	External dependencies with various licenses requiring approval	B
11.	Overwhelming amount of unnecessary settings in the services	A, B, C, D, E, F, G
12.	Excessive number of small products	B, E

References

[1] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, L. Safina, Microservices: Yesterday, today, and tomorrow, Springer International Publishing, Cham, 2017, pp. 195–216. URL: https://doi.org/10.1007/978-3-319-67425-4_12. doi:10.1007/978-3-319-67425-4_12.

[2] J. Bogner, J. Fritzsch, S. Wagner, A. Zimmermann, Assuring the Evolvability of Microservices: Insights into Industry Practices and Challenges, in: IEEE International Conference on Software Maintenance and Evolution (ICSME), Cleveland, Ohio, USA, 2019, pp. 546–556. URL: https://ieeexplore.ieee.org/document/8919247. doi:10.1109/ICSME.2019.00089.

[3] R. Verdecchia, I. Malavolta, P. Lago, Architectural technical debt identification: The research landscape, in: Proceedings - International Conference on Software Engineering, 2018, pp. 11–20. URL: https://dl.acm.org/doi/10.1145/3194164.3194176. doi:10.1145/3194164.3194176.

[4] S. S. de Toledo, A. Martini, A. Przybyszewska, D. I. Sjoberg, Architectural Technical Debt in Microservices: A Case Study in a Large Company, in: 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), IEEE, Montreal, Quebec - CA, 2019, pp. 78–87.

URL: https://ieeexplore.ieee.org/document/8786035/. doi:10.1109/techdebt.2019.00026.

[5] J. Lewis, M. Fowler, Microservices: a definition of this new architectural term, 2014. URL: https://www.martinfowler.com/articles/microservices.html.

[6] O. Zimmermann, Microservices tenets: Agile approach to service development and deployment, Computer Science - Research and Development 32 (2017) 301–310. URL: http://link.springer.com/10.1007/s00450-016-0337-0. doi:10.1007/s00450-016-0337-0.

[7] M. Fowler, Microservice Trade-Offs, 2015. URL: https://martinfowler.com/articles/microservice-trade-offs.html.

[8] P. Kruchten, R. L. Nord, I. Ozkaya, Technical debt: From metaphor to theory and practice, IEEE Software 29 (2012) 18–21. URL: https://ieeexplore.ieee.org/document/6336722. doi:10.1109/MS.2012.167.

[9] P. Avgeriou, P. Kruchten, I. Ozkaya, C. Seaman, Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162), Dagstuhl Reports 6 (2016) 110–138. URL: http://drops.dagstuhl.de/opus/volltexte/2016/6693. doi:10.4230/DagRep.6.4.110.

[10] T. Besker, A. Martini, R. Edirisooriya Lokuge, K. Blincoe, J. Bosch, Embracing Technical Debt, from a Startup Company Perspective, in: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2018, pp. 415–425. URL: https://ieeexplore.ieee.org/document/8530048/. doi:10.1109/ICSME.2018.00051.