Identifying and mapping architectural technical debt to bad smells in microservices: A case study of start-ups

MUGOYA DIHFAHSIH

August 2022

Abstract

Software companies especially startups aim to achieve quick delivery of software products to constantly provide value to their customers. The best framework in terms of architecture is to use microservices to achieve this target. Consequently, this microservice architecture is not inevitable to effects of technical debt due to hasty decisions taken thus limiting the desired continuous delivery of software products to the customers.

Objectives: The aim of this paper is to identify the developer practices (bad smells), causes, solutions as related to causing architectural technical debt in microservices.

Method: We base our analysis of bad practices on an empirical study that was made to determine practitioners' handling of bad smells in software development. The practitioners reported a total of 265 different bad practices together with the solutions they had applied to overcome them.

Conclusion: We found issues, solutions and possible causes of microservices architectures not yet encountered in the current literature. The results of this study might be useful for the practitioners who want to minimize the occurrence of Architectural Technical debt in their startups.

Index Terms: Technical Debt, Microservices, Bad smells, Startups

1 Introduction

Software companies especially startups aim to achieve quick delivery of software products to constantly provide value to their customers that are pivotal to their survival. Of recent the most startups due to desire to develop software using the trending and advanced technologies, they have opted for using microservices as their architectural strategy [6].
This is done as a way to mitigate future company growth with many teams

each developing and deploying loosely-coupled services independently. However microservice strategy is a new advancement in technology and as a result which implies a lack of empirical data. This lack of data constrains the definition of what actually constitutes a good microservice architecture thus sub-optimal solutions by startups which leads to a costly Architectural Technical Debt (ATD). Use of architecture in microservices is based on certain qualities such as communication layered services of messages using a Service Bus, such knowledge is lacking for the startups to make informed decisions instead of taking short-cuts which are harmful in the long run.
Bad smells are commonly known as Architectural smells are a type of Architectural Technical Debt (ATD) as they result into increased complexity of the software and can make future changes impossible or so costly. According to the study by Verdecchia et al [4], ATD are the most studied type of all technical debt types and interest in studying them has increased recently.

Understanding which bad practices (Bad smells) cause ATD, negative effects and their solutions would be useful for any startup adopting microservices.

- RQ1: What are Bad smells (BSs) in microservices?
- RQ2: What is the impact of ATD in microservices in start-ups?
- RQ3: What are the best practices to minimize the occurrence of ATD in microservices?

The remainder of this paper is organized as follows. Section 2 brings to life the concepts on microservices, ATD and the startups company context. Section 3 presents the Literature review, Section 4 presents the research methodology, analyzing the empirical data from the previous research. Section 5 presents the results. Section 6 presents the conclusions.

2 Background

2.1 Microservices and Technical Debt (TD)

Microservice architecture is a variant of Service Oriented architecture structural style that arranges an application as a collection of loosely coupled services. The goal of microservices is that teams can bring their services to life independent of others. [1]

When the number of services in such architecture grows, it is common to find for ways to standardize the communication. One solution that has been adopted in Service Oriented Architecture (SOA) projects [7]

Technical Debt (Technical gap) is a metaphor introduced by Cunningham [3] to represent sub-optimal design or implementation of solutions that yield a benefit in the short term but make changes more costly or even impossible in the

medium to long term, consequently affecting its testability and maintainability. An example is the use of a database solution that does not meet all needs of the system but is easier to use immediately.

Architectural Technical Debt (ATD) is a kind of TD that is concerned with the system architecture [8]. According to [8], ATD is the most challenging type of TD that needs more attention as it occurrence in a system results into the company repaying this debt high costs that are spent on code refactoring hence a stress to a startup which could end closing the operations [9]

2.2 Startups context

Giardino et al. [10], carried out an empirical study addressing how startups choose software development strategies, he explains that to be faster, startups may simply embrace TD as an investment, whose repayment may never come due, with the long-term negative effects on developer behaviors, productivity, and product quality as well as maintainability. Further, in their study it is stated that "Startups achieve high development speed by radically ignoring aspects related to documentation, structures, and processes", and that "instead of traditional requirement engineering activities, startups make use of informal specification of functionalities through ticket-based tools to manage low-precision lists of features to implement, written in the form of self-explanatory user stories" such practices introduces the concept of technical debt in a project as early as the design stage.

3 Literature Review

ATD in microservices is a new research area and it still remains a challenging field as more studies are needed [12]. The research [13] conduct a study in a view of finding the root causes of ATD in software development generally and did not look at either start-ups or microservices yet we need more research in this area to get an understanding as to why ATD is so prominent in start-up companies. [6]

Martini et al. [15] conducted a research of the ATD issues in large international companies and came up with a classification of the most dangerous bad practices in software development in general. The focus of our paper is on explicitly ATD in microservices in start-up companies whose survival could be threatened by a flux of complexities in TD [6]

Taibi et al. [14] presented a classification of bad smells on microservices. This proves that there exist a relationship between ATD and bad smells. Smells are just indicators that point at the likelihood of a TD. Though this study corre-

lates TD and bad practices, it does not look at the start-up nature of businesses which are so vulnerable to ATD [6]

Francesco et al. [15] conducted a research on microservices architecture that showed that there are still gaps in studies carried out to find evidences of practitioners' lack of evaluation of research on microservices and quality requirements to help establish the causes of ATD in microservices, in our paper we provide these causes and we give an overview of the bad smells specifically the anti-patterns.

4 Research Methodology

We use the exploratory case-study data from a report [11] that analyzed 88 start-ups revealing three anti-patterns which are part of bad smells in microservices. All the companies analyzed in this report implement the microservice oriented architecture with several issues that were included in the report. This data takes a look at how these start-ups leverage Minimum Viable Product (MVP) to deliver the software product as soon as possible to the market to meet the customer expectations. In the reports analyzed, some of the start-ups after realizing ATD in their communications layers, they replaced these services with new ones and in the due process these services being replaced were costly in terms of code refactoring due to lack of proper documentation of previous changes made to the system. The study [11] further used qualitative technique to analyze the causes, the cost and the likely solutions to ATD due to BSs.

4.1 Data collection

The data from experience reports of start–ups by the practitioners according to the report [11] depending on requirement analysis, designing and implementation was collected. This data used in the report [11] was not intended for this paper but the insights from the primary data provides a clear understanding of how start-ups use the microservice architecture to support their operations. Using the data provided in the report by the practitioners, we identify;

- ATD was present in the systems whose reports were analyzed
- The negative impacts caused by the use of anti-patterns in start-ups
- The causes of ATD due to bad smells in start-ups projects.

4.2 Screening and Data Coding

The reports that were analyzed did not all provide the relevant information about the bad practices thus only 5 reports from 93 were staged for data coding phase thus providing the relevant data that was relied on to identify the three anti-patterns whose impact is analyzed in the subsequent sections of this paper.

The qualitative data analysis was used to help in differentiating from the symptoms or indicators of a problem or anti-pattern and the problem itself which could lead into a technical debt in future. The coded data from the reports was used as an evidence to answer our paper research questions.

4.3 Data Analysis

Analyzing the data collected from the reports by the 88 start-up companies, it is evident that most of these companies suffered almost the same effects of bad smells in software development. We highlight them in the following section D identifying their causes and suggesting the solutions as a remedy to each anti-pattern.

4.4 Start-up engineering microservice anti-patterns.

4.4.1 Anti-pattern I: Releasing unworthy product to market

Releasing an MVP to the market to test the customer preferences and then feedback used in designing a better product stands out as one of the main cause of bad smells in the start-ups. This leads into an ATD caused by

- Poor design decisions such as unrealized quality requirements, use of immature or unfamiliar technologies.
- Poor requirements gathering due to vague understanding of the product
- Frequent changes in the product direction to target the customer direction.

4.4.2 Anti-pattern II: Wrong cuts

From the data analyzed in the report [11], it is evident that most of the 88 sampled start-ups opted for using microservice architecture but due to hasty implementation to meet market demands, the system is separated into strongly coupled and dependent services. The result is a large number of microservices that are dependent on each other and all we get in the end is a distributed monolith system. This is caused when practitioners ignore best practices and resort to;

- Not thinking monolithic first before implementing the microservice architecture [1]
- Neglecting domain driven design while implementing the microservices
- Ignoring the use of messages as a communication strategy between services to reduce coupling between the services

4.4.3 Anti-pattern III: Entangled Data

This anti-pattern is found in scenarios where all the services have full access to all database objects. This is a problem as it makes it hard to scale a module in a given application. This leads to performance issues when making structural changes to the database because the data is entangled. This is caused due to;

- Ignoring abstraction of data inside the application domain in a distinctive manner.
- Neglecting the use of access policies while accessing the database
- Not isolating relevant data that is so specific to a service.

4.4.4 Anti-pattern IV: Improper versioning

Most start-ups in the report analyzed had implemented microservices with poor versioning strategy due to speed to deliver the product to the market as early as possible to beat of competition. This in turn makes the code over time difficult to manage thus causing the ATD as service upgrades and management processes quickly become complicated. This is caused due to;

- Not explicitly putting in place versioning practices for the various services in the start-up.
- Not designing for change due to quick and unreasoned design decisions thus making service changes hard making versioning inflexible overtime

5 Results

This section discusses the results of each research questions RQ

5.1 What are Bad smells in Microservices? RQ.1

From the data in reports[11] by the practitioners, Bad smells are seen as indicators of situations such as undesired patterns, anti-patterns or bad practices that negatively affect software quality attributes such as understandability, extendibility, testability, reusability and maintainability of system under development [1]. Practitioners indicated in their start-up reports that BSs show themselves in the system as undesired dependencies, distribution of responsibilities in unbalanced manner, and excessive coupling between modules and in many other forms that break one or more software design principles and good practices, ultimately affecting maintainability [2].

5.2 What is the negative impact of ATD in microservices in start-ups? RQ.2

In the report analyzed, the practitioners pointed out several negative effects of ATD in start-ups. There are high costs for maintaining the microservice systems in the small star-ups that is often constrained by resources. From the report data investigated, practitioners expressed concerns of complexities in implementing the microservices thus rendering the architecture hard to change.

Coupling between services, this is in violation of the principle of lose coupling between services in a microservices architecture thus making services highly dependent on each other. This cripples the start-ups efforts of implementing and leveraging the use of microservice architecture.
Unnecessary dependency between development teams, As a result of desire to beat market, practitioners in the report [11] indicate that developers have to work in sync since these services have been implemented dependent on each other which undermines the sole goal of microservices of having independent services.

5.3 What are the best practices to minimize the occurrence of the ATD in microservices? RQ.3

5.3.1 Use the right technology with the microservices

Due to lack of enough resources, startups tend to employ developers that lack enough experience and as a result these tend not to give importance to the technology or language without thinking about the impact such will have on the overall project. Therefore technology should be implemented iteratively and direct so as to create room for future changes. The project leaders should ensure that the tools used to develop the software is secure and is well supported by community so as to help the team in case of any need. To select the technology to be used; these should act as the baseline for decision making such as Maintainability, Fault-tolerance, Scalability, Cost of architecture and Ease of deployment.

5.3.2 Consider using Domain Driven Design [16]

This helps to identify the correct boundaries of our system. This is the object oriented programming applied to business model. This framework can help the startups to aggregate what is common and create for it an independent service. This can help in defining the size of startup microservice—not the LOC size, the size in terms of functional scope. Using this model is helpful in avoiding many microservices that would kill design causing the adoption of the architecture that would be costly to change

5.3.3 Leverage the benefits of using Rest API

The REST (Representational State Transfer) APIs can work wonders for microservices as developers need not install any additional software or libraries while creating a REST API. At the same time, they provide a great deal of flexibility since the data is not tied to any particular method or resource. The result is an ability to handle multiple types of calls, return different data formats such as JSON, XML, TXT, and the structure is formatted so that it can be used by a particular by a given implementation of microservices. The developers don't even need a framework or SDK since HTTP requests are relatively sufficient. Out of the four levels of REST, simply begin at level 0 and make the way up to level 3.

6 Conclusion and Future work

The exploratory study of analysis on the report of practitioners from the 88 startups shows the correlation between the bad practices and architectural technical debt. We analyzed the data in the report and identified the causes of bad smells in the microservices of the startups that included wrong cuts, entangled data, poor or no documentation.
We also found out the best practices that the startups can adopt to minimize the occurrence of ATD in microservices We found out that bad smells can cause ATD in microservices as follows: releasing unworthy product to the market, the wrong cuts, Improper API versioning, and entangled data These Anti-patterns caused ATD, such as need of rewriting the code, extra effort to handle different technologies, and coupling between services. Our results contribute to an increased understanding of the relationship between ATD and microservices.
Our future work includes analyzing actual data that we would collect from the different startups as opposed to the use of reported data which we based on for this paper.

7 References

[1] On the Definition of Microservice Bad Smells Davide Taibi and Valentina Lenarduzzi, Tampere University of Technology

[2] Refactoring in Large Software Projects Performing Complex Restructurings Successfully Lippert M, Roock S.John Wiley Sons, Inc 2006

[3] Cunningham, W., 1992. The wycash portfolio management system. SIGPLAN OOPS Mess. 4 (2), 29–30

[4] Verdecchia R, Malavolta I, Lago P. Architectural technical debt identification: the research landscape. In: 2018 ACM/IEEE International Conference on Technical Debt. Gothenburg, Sweden; 2018:11-20

[5] A. Martini, T. Besker, and J. Bosch, "Technical Debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations," Science of Computer Programming, vol. 163, pp. 42–61, oct 2018

[6] Designing Microservices Architecture For Software Product In Startup Rikza Nashrulloh, M., Setiawan, R., Heryanto, D., Sutedi, A. and Elsen, R. 2022

[7] Benjamin Kanagwa, Ezra Kaahwa Mugisa (2007):Architecture Analysis of Service Oriented Architecture. Software Engineering Research and Practice 2007: 658-663

[8] G. Hohpe and B. WOOLF, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, ser. The Addison-Wesley Signature Series. Prentice Hall, 2004

[9] T. Besker, A. Martini, R. Edirisooriya Lokuge, K. Blincoe and J. Bosch, "Embracing Technical Debt, from a Startup Company Perspective," 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2018, pp. 415-425, doi: 10.1109/ICSME.2018.00051

[10] C. Giardino, N. Paternoster, M. Unterkalmsteiner, T. Gorschek and P. Abrahamsson, "Software Development in Startup Companies: The Greenfield Startup Model", IEEE Transactions on Software Engineering, vol. 42, no. 6, pp. 585-604, 2016

[11] Software Engineering Anti-patterns in Start-ups E. Klotins, M. nterkalmsteiner, T. Gorschek SERL, Blekinge Institute of Technology, Karlskrona, Sweden

[12] A. Martini and J. Bosch, "On the interest of architectural technical debt: Uncovering the contagious debt phenomenon," Journal of Software: Evolution and Process, vol. 29, no. 10, pp. 1–18, 2017.

[13] R. Kazman, Y. Cai, R. Mo, Q. Feng, L. Xiao, S. Haziyev, V. Fedak, and A. Shapochka, "A Case Study in Locating the Architectural Roots of Technical Debt," in 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, may 2015, pp. 179–188. [Online

Available: http://ieeexplore.ieee.org/document/7202962/ [14] D. Taibi and V. Lenarduzzi, "On the Definition of Microservice Bad Smells," IEEE Software, vol. 35, no. 3, pp. 56–62, may 2018.

[15] Towards Microservice Smells Detection

[16] Modeling Microservices with DDD Paulo Merson –pmerson@acm.org Joe Yoder –joe@refactory.com