**Minimizing Architectural Technical Debt in microservice framework by start-ups**

Abstract

Software companies especially startups aim to achieve quick delivery of software products to constantly provide value to their customers. These companies are faced with tightness of time, resource constraints and stiff competition as a result they often opt for sub-optimal solutions such as using microservices before they fully understand the principles surrounding this architecture.

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. This position paper discusses crucial questions that need to be addressed in order to reduce technical debt in startups especially those that opt for using microservices architecture as opposed to monolithic design.

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. 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]. 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.

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 ATD in microservices?

RQ3: What are the causes of BSs that result into ATD?

RQ4: What are the consequences of ATD in microservices?

RQ5: What is the solution for solving the ATD in microservices?

The remainder of this paper is organized as follows. Section II brings to life the concepts on microservices, ATD and the startups company context. Section III presents the research methodology, analyzing the empirical data from the previous research. Section IV presents the findings for each RQ. Section V presents the implications for research and industry. Section VI presents related work.

Lastly. Section VII presents the conclusions.

1. Background
2. 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]

1. Company 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.

1. 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.

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.

1. 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.

1. 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.

1. Start-up engineering microservice anti-patterns.

These anti-patterns are used as a way to find an optimal solution or profit to the company but leading into ATD because most times the design phase is done partially which has a lasting impact on the quality of the product to be delivered to the customers. From the report, we were able to identify three common anti-patterns that were reported by the practitioners.

* 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.
  1. 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

1. Results

Bad smells (BSs) are defined 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]

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].

The existence of BSs does not show that there is a problem in the system but they indicate places or points in the system architecture that should be analyzed [2]

BSs 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.

Microservices

Literature review refs.

* + 1. [Conferences](https://ieeexplore.ieee.org/browse/conferences/title/) >[2021 IEEE 18th International ...](https://ieeexplore.ieee.org/xpl/conhome/9426689/proceeding)The Influence of Cognitive Biases on Architectural Technical Debt
    2. Microservices ant-patterns, a case study for large companies
    3. Characterizing Technical Debt and Antipatterns in AI-Based Systems: A Systematic Mapping Study

# The entrepreneurial logic of startup software development: A study of 40 software startups

* + 1. Start-up failures due to Quality Failures Saadullah Aleem Institute of Business Administration, Karachi, 75270, Pakistan E-mail: [saadullah.aleem@khi.iba.edu.pk](mailto:saadullah.aleem@khi.iba.edu.pk)

# Infinite technical debt Author links open overlay panel [MelinaVidonia](https://www.sciencedirect.com/science/article/abs/pii/S0164121222000772#!)[ZadiaCodabux](https://www.sciencedirect.com/science/article/abs/pii/S0164121222000772" \l "!)[b](https://www.sciencedirect.com/science/article/abs/pii/S0164121222000772" \l "!)[Fatemeh H.Fard](https://www.sciencedirect.com/science/article/abs/pii/S0164121222000772" \l "!)[c](https://www.sciencedirect.com/science/article/abs/pii/S0164121222000772" \l "!)

# Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation

# You don't need a Microservices Architecture (yet): Monoliths may do the trick.

# The Monolith Strikes Back: Why Istio Migrated From Microservices to a Monolithic Architecture

# The Negative Implications of Technical Debt on Software Startups: What they are and when they surface

**References**

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

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

[3] Cunningham, W., 1992. The wycash portfolio management system. SIG-  
PLAN 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