**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 in a continuous fashion. These new and immature 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 proper usage of such an 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 identifies the gaps in current research about minimizing technical debt in microservices in startups, the paper proposes a methodology that should be used in future research to reduce technical debt in startups especially those that opt for using microservices architecture as opposed to monolithic design.

1. Introduction

Technical Debt is a well-known concept in software engineering, while performing a technical task in a software development, a suboptimal solution is used to accomplish the task in the shortest time possible. [14] This creates a compromise which affects how to deal with a software in future thus technical debt has to be paid. In the context of the startups, the rushed decisions [15] are made due to the pressure from the stiff competition, constrained resources, anti-patterns or bad practices in software development [16] and business uncertainty. This makes the startups take up wrong architectural decisions which are always hard or impossible to change in future. However, not mitigating and minimizing such decisions result into a harmful outcome such as death of the startup itself.

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. Technical Debt (TD)

Commonly known as developer Technical gap is a metaphor introduced by Cunningham Ward [17] 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. TD is categorized in two forms, first form is intentional debt where the company willingly makes a decision fully aware that it will cost it in future, the other form is unintentional debt which is non-strategic as a result of lack of knowledge of implementation of technologies, poor logical thinking by developers. Most startups incur both forms technical debt due to the pressing business factors. Architectural Technical Debt (ATD) is a kind of TD that is concerned with the system architecture [ref]. ATD is the most challenging type of TD that needs more attention as its occurrence in a system results into the company repaying this debt high costs that are spent on trying to change the structure of the software hence a stress to a startup which could end closing the operations [5]. TD has a short term advantage for example foster time to market [10] but using a wrong architecture can be disastrous to a startup

1. Startup Context

A start up is a new or an immature focuses on developing innovations that are customer driven to enable it grow exponentially faster [4]. It is faced with many challenges stemming from which idea will propel the company into penetrating the market. The fit for market is also one of the earlier concerns of a startup which calls for rapid use of methods for building, testing and iterating software development [5]. Uncertainty about the market requirements, processes not well understood, limited resources which lures startups to hire novice developers who contribute enormously to occurrence of TD [10].

1. Microservices Architecture(MA)

MA is a variant of service oriented architecture structural style that arranges an application as a collection of loosely coupled, independently deployed services. The goal is to have different module of the software handled independent of the other application services [7]. MA is purposefully used in startups as a design strategy to deliver continuously the customer value in a faster fashion [11]. Microservice usage in startups does not have enough body of knowledge in research, therefore architects applying this architecture as a framework of design must fully understand what they are doing otherwise the decision of either to use monolithic architecture or MA must be made with caution because the design might be hard or impossible to replace in future which could cost the start when there is need to add in a new customer feature [8].

1. ATD and MA an issue of concern for startups.

As a way to get faster paced feedback from the prospective customers, startups opt for use of microservices to design a Minimal Viable Product (MVP), if the feedback is negative, the prototype is revised and new changes are incorporated. Due to hasty nature of pressing market factors, these new changes end up not reflected in the documentation of the product, this causes TD in future as it will make it costly to refactor the code or change the design of the product leading to ATD. The startups hire novice developers with limited knowledge about implementation of MA and as a result they use sub-optimal solutions to get the work done, such practices make it inevitable for startups to survive severe effects of TD that threaten the survival of a startup [5], [10].

1. Critiquing the relevant literature

Kara Borowa et al. [1] Describe the sole cause of ATD in any business product is due to human biased or intentional actions in planning, implementation or monitoring the software product. The study further highlights the role of organizational culture in avoidance or minimizing TD in software development processes. The complexities in mechanisms used by people or human decisions such as sub-optimal solutions such as anti-patterns to create software products lead to TD in companies

References

* + 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. 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
    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

* + 1. Identifying architectural technical debt, principal, and interest in microservices: A multiple-case study
    2. Technical decision-making in startups and its impact on growth and technical debt Master of Science Thesis TRITA-ITM-EX 2021:240 KTH Industrial Engineering and Management Industrial Management SE-100 44 STOCKHOLM

# How do Technical Debt Payment Practices Relate to the Effects of the Presence of Debt Items in Software Projects?

* + 1. Avgeriou, P., Kruchten, P., Ozkaya, I., Seaman, C., 2016b. Managing technical debt in software engineering (dagstuhl seminar 16162). Dagstuhl Rep.6 (4), 110–138. nstitute of Technology, Karlskrona, Sweden
    2. How do Technical Debt Payment Practices Relate to the Effects of the Presence of Debt Items in Software Projects?
    3. 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
    4. Cunningham, W., 1992. The wycash portfolio management system. SIGPLAN OOPS Mess. 4 (2), 29–30