Minimizing occurrence of architectural technical debt in microservice frameworks used by startups

MUGOYA DIHFAHSIH

Student No. 2100702353 — Reg No. 2021/HD05/2353U

Abstract

Software companies especially startups aim at achieving 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 architectural frameworks. 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 architecture as used by startups, the paper proposes a methodology that should be used in future research to reduce technical debt in startups especially those that opt for microservices architecture as opposed to monolithic design.

Keywords— Architectural Technical Debt, Startups, Micro-services

1 Introduction

Technical Debt is a well-known concept in software engineering, while performing a technical task in a software development, a sub-optimal solution is used to accomplish the task in the shortest time possible.[8] This creates a compromise which affects how to deal with a software in future thus technical debt has to be repaid with interest. In the context of the startups, the rushed decisions [10] are made due to the pressure from the stiff competition, constrained resources, anti-patterns or bad practices in software development [13] and business uncertainty. This makes the startups make wrong architectural decisions which are always hard or impossible to change in future. However, not mitigating and minimizing such decisions results into a harmful outcome such as death of the startup itself.

1.1 Background

1.1.1 Technical Debt (TD)

Commonly known as developer technical gap is a metaphor introduced by Cunningham Ward [7] 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 test-ability 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, and the other form is unintentional debt which is non-strategic as a result of lack of knowledge of implementation of technologies and poor logical thinking by the software development practitioners. Most startups incur both forms of technical debt due to the pressing business factors. Architectural Technical Debt (ATD) is a type of TD that is concerned with the system architectural defects that makes changes in design hard [9]. 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 with high costs spent on trying to change the design of the software hence a stress to a startup which could end closing the operations or fail it to move into a maturity phase [3]. TD has a short term advantage for example foster time to market [2] but using a wrong architecture can be disastrous to a startup.

1.1.2 Startup Context

A start up is a new or an immature company with small teams [8] focused on developing innovations that are customer driven to enable it grow exponentially faster [14], these teams can work independently on different modules of a product thus opting for microservices architecture. In early stages, it is faced with many challenges stemming from which idea will propel the company into penetrating the market by developing a model for products, processes and services. 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 [3]. 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 [2].

1.1.3 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 modules of the software handled independently from other application services [4]. MA is purposefully used in startups as a design strategy to deliver continuously the customer value in a faster fashion [17]. 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 [11].

1.1.4 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 [1] 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 [3], [2].

2 Literature Review

Kara Borowa et al.[6] 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 like using anti-patterns instead of proper design patterns to create software products lead to TD in companies. However the study looks at the general organizational culture of minimizing TD, it majorly focuses on techniques used by established companies and thus does not address the concerns of human bias in planning or decision making in startups which are severely affected by the occurrence on ATD [2]. A study [13] on bad practices in large companies clearly highlights a distinction between those anti-patterns that cause ATD in microservices in large companies and those in startups. This shows that causes of TD are highly correlated to the nature of the business whether established or startup because the impact of ATD [3] in startups is not the same as that in large or established companies.

Justus Bogne et al. [5] conceptualized and mapped TD to ant-patterns in Artificial Intelligence Based Systems. The study shows that the practitioners' bad practices or ant-patterns such as neglecting the documentation of the product, not commenting in the code during implementation as well as not doing testing before deploying the software are the leading causes of ATD. Continued use of these bad practices have a negative impact on the product being developed as it limits architectural changes for addition of any feature as deemed by the customers. If such happens to a startup whose main objective for using a microservice to come up with a MVP for the customers can end up collapsing. The study however does not categorically highlight how AI-based startups can minimize such practices especially wrong decisions in early stages of product development such as selecting the architectural design.

Saadullah et al. [3] conducted a study of failure of startups due to severe effects of TD. This study highlights the need to address quality requirements issues such as testing, code quality and incorporating security in Software Development Life Cycle (SDLC), when these and other best practices are addressed, the occurrence of TD is minimized. The study also ties the responsibility of minimizing the ATD to practitioners such as Project Managers, Developers and Designers depending on the design decisions they make about the product. The study suggests broad actions towards reducing ATD by practitioners but it does not make any mention of how to handle and minimize TD in startups which are affected most with the occurrence of unmanaged debt.

Melina Vidoni et al.[16] developed a mathematical approach of managing TD. The research further highlights ways of incorporating TD management in SDLC as early as possible when developing a product. This is one of the best ways of mitigating and

evaluating the occurrence of TD thus selecting the right architecture to use for the product to reduce consequences of ATD in future. The study describes practitioners' mindset change about taking quick decisions or sub-optimal solutions as a guiding tool to reducing TD in software development. However the study fails to address the reasons as to why practitioners in startups take sub-optimal solutions and incur a debt willingly as a way of taking risks so as to thrive and penetrate through the market. Until such issues are addressed, the occurrence of TD in microservices will not be addressed as most practitioners take TD as a risk that can help them get the market share of their products so quickly thus continue to use anti-patterns throughout the SDLC.

3 Justification for future research

3.1 Gaps in the current wok

- Studies conducted on management of TD are mostly focused on handling or minimizing in established companies but not startups which are prone to TD whose actions or decisions at the product inception such as choosing a wrong architecture can result into ATD which is hard or impossible to refactor to add a new product feature.
- The available literature only focuses on general architecture and don't explicitly look at the anti-patterns leading to ATD in each of the available architectural style chosen as a design framework either monolithic or microservices. Since microservices are new, there is limited research on the occurrence of ATD and the practices that should be avoided to minimize the occurrence of TD in usage of MA to design a software product by a startup.
- The studies don't suggest a framework that can be used by startups in early stages of product inception to foster making of informed decisions by the practitioners to enable them reduce TD as early as possible than incurring significant losses when it occurs in future.
- The studies don't come up a mitigation strategy that can enable startups to foresee the likelihood of occurrence of ATD in MA when they use anti-patterns to develop early product prototypes for the customers.

3.2 Position for future work

The use of microservices in a correct way can boost the startups approaches of developing an MVP to get feedback on what should be included as part of the product. MA supports this as independent modules [10] are developed which can be easy to change code unlike the use of one block monolithic architecture. Designing a framework that enables startups minimize the occurrence of ATD would be a great contribution to body of knowledge as such is not sufficient about the usage of MA [7] in startups and how ATD can be properly mitigated at the product inception.

My position is that the available studies don't address the concerns of startups in regard to use of MA in developing their MVP. The startups end up making wrong decisions about the usage of MA in developing products which leads to ATD [15],

[8]. This can be minimized by a more empirical study which enables the startups practitioners to mitigate the occurrence of ATD if they make decisions such as use of anti-patterns[7] as the optimal solutions in developing products using MA framework.

4 Methodology to be used

4.1 Qualitative approach

Differently from a quantitative approach, a qualitative study can be executed in startups with the focus on software practitioners such as developers, project managers, architects and decision making leaders. This qualitative method makes it possible to achieve the main aspects behind the causes of using sub-optimal solutions and their effects on the startups and how these practitioners avoid using the software development bad practices in startups. The researchers can improve conclusions through collecting and analyzing of different data from different startups both those that hire novice developers and those with senior ones. This is in contrast to quantitative approaches that focus on analyzing of just variables, controlled groups and statistical data which sometimes are inconsistent with startups usage of MA.

4.2 Method

Tools such as interviews can be used to gather data about the causes of use of sub-optimal solutions in startups. The interview questions should be structured in a way that each category of the practitioners give different view of why they take these quick fixes instead of standard solutions. The interview tool should have a pyramid where the first questions are specific such as experience on using the MA in startups, knowledge about anti-patterns and whether they know about ATD [15]. The focus of the data collection should be centered on decisions made at inception stage of the product rather than after the product has been already developed. The data collected can be codified and analyzed with concern on use of MA as architectural framework.

4.3 Results

Using the analyzed data collected, it is possible to map the likelihood of the ATD on the knowledge of developers on the use of MA. It is also possible to make conclusions on the survival of the startups that take on TD willingly [1]. These results can then be used by the startups not only to understand the effects of TD on architectural decisions but also when they are likely to occur and the relationship with various aspects of setups such as hiring of developer teams [12]

4.4 Conclusion

Increased interest in conducting empirical studies about TD mitigation frameworks for startups at early stages of software development will make it possible to minimize the occurrences of ATD for startups that opt to use MA. These studies help to guide the startups in making informed decisions to avoid taking sub-optimal solutions that offer short term fixes but make software refactoring costly in future.

References

[1] Abdullah Aldaeej. *Towards Effective Technical Debt Decision Making in Software Startups: A Multiple Case Study of Web and Mobile App Startups*. PhD thesis, University of Maryland, Baltimore County, 2021.

[2] Abdullah Aldaeej and Carolyn Seaman. The negative implications of technical debt on software startups: What they are and when they surface. 2022.

[3] Saadullah Aleem, Waqas Mahmood, et al. Start-up failures due to quality failures. *International Journal of Engineering and Manufacturing*, 11(2):1–13, 2021.

[4] Grzegorz Blinowski, Anna Ojdowska, and Adam Przybyłek. Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE Access*, 10:20357–20374, 2022.

[5] Justus Bogner, Roberto Verdecchia, and Ilias Gerostathopoulos. Characterizing technical debt and antipatterns in ai-based systems: A systematic mapping study. In *2021 IEEE/ACM International Conference on Technical Debt (TechDebt)*, pages 64–73. IEEE, 2021.

[6] Klara Borowa, Andrzej Zalewski, and Szymon Kijas. The influence of cognitive biases on architectural technical debt. In *2021 IEEE 18th International Conference on Software Architecture (ICSA)*, pages 115–125. IEEE, 2021.

[7] Orges Cico, Terese Besker, Antonio Martini, Anh Nguyen Duc, Renata Souza, and Jan Bosch. Toward a technical debt relationship with the pivoting of growth phase startups. In *International Conference on Product-Focused Software Process Improvement*, pages 265–280. Springer, 2021.

[8] Orges Cico, Renata Souza, Letizia Jaccheri, Anh Nguyen Duc, and Ivan Machado. Startups transitioning from early to growth phase-a pilot study of technical debt perception. In *International Conference on Software Business*, pages 102–117. Springer, 2020.

[9] Saulo S de Toledo, Antonio Martini, and Dag IK Sjøberg. Identifying architectural technical debt, principal, and interest in microservices: A multiple-case study. *Journal of Systems and Software*, 177:110968, 2021.

[10] Sávio Freire, Nicolli Rios, Boris Pérez, Darío Torres, Manoel Mendonça, Clemente Izurieta, Carolyn Seaman, and Rodrigo Spínola. How do technical debt payment practices relate to the effects of the presence of debt items in software projects? In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 605–609. IEEE, 2021.

[11] Dimitrios Gravanis, George Kakarontzas, and Vassilis Gerogiannis. You don't need a microservices architecture (yet) monoliths may do the trick. In *2021 2nd European Symposium on Software Engineering*, pages 39–44, 2021.

[12] Carl Hultberg. Technical decision-making in startups and its impact on growth and technical debt, 2021.

[13] Antonio Martini, Terese Besker, and Jan Bosch. Technical debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations. *Science of Computer Programming*, 163:42–61, 2018.

[14] Anh Nguyen-Duc, Kai-Kristian Kemell, and Pekka Abrahamsson. The entrepreneurial logic of startup software development: A study of 40 software startups. *Empirical Software Engineering*, 26(5):1–55, 2021.

[15] Mohamed Soliman, Paris Avgeriou, and Yikun Li. Architectural design decisions that incur technical debt—an industrial case study. *Information and Software Technology*, 139:106669, 2021.

[16] Melina Vidoni, Zadia Codabux, and Fatemeh H Fard. Infinite technical debt. *Journal of Systems and Software*, page 111336, 2022.

[17] Lu Wang, Yu Xuan Jiang, Zhan Wang, Qi En Huo, Jie Dai, Sheng Long Xie, Rui Li, Ming Tao Feng, Yue Shen Xu, and Zhi Ping Jiang. The operation and maintenance governance of microservices architecture systems: A systematic literature review. *Journal of Software: Evolution and Process*, page e2433.