Microservices Architectures and Technical Debt: A Self-adaptation View

Ananya Chhonker and Rogério de Lemos

University of Kent, UK

Recognised benefits of microservice architectures (MSA) include increase in productivity, scalability and maintainability, with the overall goal of promoting easier deployment. However, the perceived complex nature of MSA may raise challenges related to data sharing, testing, security management, communication optimisation, and system performance [7]. These challenges might lead to technical debt [1], which arises from design decisions that consciously or unconsciously compromise system-wide quality attributes.

Since not much has been investigated regarding MSA in the context of technical debt [3], the goal of this work is to assess the impact of MSA into TD, and probe whether self-adaption can be seen as a solution to reducing TD of MSA-based software systems. Self-adaptation enables a system to modify their behaviour and/or structure in response to changes that occur to the system, its environment, or even its goals. MSA principles may be able to reduce maintenance and evolution cycles, however the same claim cannot be generalised towards other quality attributes.

The lack of primary or secondary studies analysing the principles of MSA in light of TD was a motivating factor to modify the basis for selecting primary studies. In the approach taken, we have looked into key primary sources that summarises the state-of-the-art of TD and MSA, in order to identify correlations between these two fields regarding qualities attributes.

Understanding TD in MSA is essential to make appropriate decisions when building software systems either from scratch or refactoring an existing system. Either way, it is important to properly decide the size, scope and the capabilities of MSA. Within these decisions it is important to agree on the type and acceptable level of TD intake. However, there are no guidelines or framework that can shed some light on the various reasons or key points to consider while evaluating the debt capacity. This could lead to developers making assumptions or making use of methods that could negatively affect an MSA-based software system during its lifetime. As a consequence, software quality may be degraded, re-work costs increased, team productivity decreased, which would result in loss of business and gain of TD. However, if the teams have knowledge of challenges in MSA, and awareness of TD and its implications, they could use this to their advantage and be more cautious while making decisions regarding MSA.

Technical debt is sourced by leveraging the speed of releasing a software product against its quality, so these, as the TD landscape illustrates, are two tightly linked concepts. Since software architecture is driven by the system quality attributes, it is at the architecture level that quality of attributes of an MSA should be analysed because of their potential to incur TD. It is recognised that a bad architecture is the most occurred and hardest TD to pay [6]. If an architecture is poorly build, the system will not only incur heavy TD but may fail completely. Therefore studying the characteristics of MSA can help measure the risk and the amount of TD in a system.

In our study, we have focused on some less studied, but yet important, MSA quality attributes [5] that can affect TD. In the following, we cover three of these attributes, namely,

[1] *Technical debt* (TD) is a metaphor that is used to communicate the consequences of poor software development practices to non-technical stakeholders [1]

testability, security and reliability, already in the context of existing self-adaptation solutions that can be ported to MSA. However, the support for self-adaption requires well reasoned architectural models to be manipulated at run-time, and this could be beneficial to mitigating TD.

Considering the continuous evolution of an MSA-based software system, testability becomes an important factor, particularly integration and regression testing. Integration testing can be challenging in MSA, especially in a large system with a complex configuration. This can make it difficult to test the component services, and anomalies associated with their collaborations. Self-adaptation is able to handle integration testing [4].

Security is a major concern in MSA because of its distributed nature. MSA is exposed to an increased surface attack area as the software application is broken down into component services that communicate through APIs. Moreover, MSA promotes service reuse which is incredibly useful but equally risky because it can introduce vulnerabilities. Another factor is the fuzzy boundaries in MSA, which make it easy to break trust relationships. If a single component service is attacked it can compromise the whole MSA. System reconfiguration is a mechanism to self-protect the system against attacks [8].

The distributed nature of MSA may also affect the overall reliability if redundancies are not properly provided and managed since the failure of a component service may affect the reliability of the whole system. Communication over network brings extra complexity to the system, and it can also be a source of failure. Self-healing is a solution for managing redundancies that support adaptation [2].

In this paper, we briefly discussed the impact that technical debt (TD) may have on MSA regarding some quality attributes, and hypothesise how self-adaptation could be useful in dealing with some aspects of TD. The next step is to map some of these solutions into an MSA-based software system.

References

[1] W. N. Behutiye, P. Rodríguez, M. Oivo, and A. Tosun. Analyzing the concept of technical debt in the context of agile software development: A systematic literature review. *Information and Software Technology*, 82:139 – 158, 2017.

[2] J. Cámara, P. Correia, R. de Lemos, D. Garlan, P. Gomes, B. Schmerl, and R. Ventura. Incorporating architecture-based self-adaptation into an adaptive industrial software system. *J. Syst. Softw.*, 122(C):507–523, December 2016.

[3] A. Chhonker. Impact of microservices architecture into technical debt: A systematic literature review. Master's thesis, School of Computing, University of Kent, UK, 2018.

[4] C. E. da Silva and R. de Lemos. Dynamic plans for integration testing of self-adaptive software systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, pages 148–157, New York, NY, USA, 2011. ACM.

[5] P. D. Francesco, I. Malavolta, and P. Lago. Research on architecting microservices: Trends, focus, and potential for industrial adoption. In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 21–30, April 2017.

[6] Z. Li, P. Liang, and P. Avgeriou. Architectural technical debt identification based on architecture decisions and change scenarios. In *12th Working IEEE/IFIP Conference on Software Architecture*, pages 65–74, May 2015.

[7] J. Thönes. Microservices. *IEEE Software*, 32(1):116–116, Jan 2015.

[8] E. Yuan, S. Malek, B. Schmerl, D. Garlan, and J. Gennari. Architecture-based self-protecting software systems. In *Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures*, QoSA '13, pages 33–42, New York, NY, USA, 2013. ACM.