Does Migrate a Monolithic System to Microservices Decreases the Technical Debt?

Nyyti Saarimäki, Francesco Lomio, Valentina Lenarduzzi, Davide Taibi

TASE - Tampere Software Engineering, Tampere University, Finland

Abstract

Background. The migration from monolithic systems to microservices involves deep refactoring of the systems. Therefore, the migration usually has a big economical impact and companies tend to postpone several activities during this process, mainly to speed-up the migration itself, but also because of the need to release new features.

Objective. We monitored the Technical Debt of a small and medium enterprise while migrating a legacy monolithic system to an ecosystem of microservices to analyze changes in the code technical debt before and after the migration to microservices.

Method. We conducted a case study analyzing more than four years of the history of a big project (280K Lines of Code) where two teams extracted five business processes from the monolithic system as microservices, by first analyzing the Technical Debt with SonarQube and then performing a qualitative study with the developers to understand the perceived quality of the system and the motivation for eventually postponed activities.

Result. The development of microservices helps to reduce the Technical

Email addresses: nyyti.sarimaki@tuni.fi (Nyyti Saarimäki), francesco.lomio@tuni.fi (Francesco Lomio), valentina.lenarduzzi@tuni.fi (Valentina Lenarduzzi), davide.taibi@tuni.fi (Davide Taibi)

Debt in the long run. Despite an initial spike in the Technical Debt, due to the development of the new microservice, after a relatively short period the Technical Debt tends to grow slower than in the monolithic system.

Keywords: microservice, cloud, cloud-native, technical debt, software quality, software evolution, continuous architecture, refactoring

1. Introduction

Migration to microservices has become very popular over the last years. Companies migrate for different reasons, expecting to improve the quality of their system and to ease the software maintenance [1].

Companies commonly adopt an initial migration strategy to extract some components from the monolithic system as microservices, making use of simplified microservices patterns [1], [2] . As an example, companies commonly directly connect the microservices to the legacy monolithic system and do not adopt any message bus at the beginning. When the system starts to grow in complexity, they usually start re-architecting their systems, considering different architectural patterns [1],[2][3]. The migration from a monolithic system to microservices is commonly performed on systems that are currently maintained. Therefore, in several cases, the development of new features has the priority over refactoring of the code, generating Technical Debt (TD) every time an activity is postponed [4], [5] and increasing the software maintenance cost.

Companies migrate to microservice to ease the maintenance. [1] However, recent surveys confirm that maintenance costs increase after the migration [1] [6]. Therefore, the goal of this paper is to understand if the overall code

TD of a system increases or decreases after the migration to microservices, and if the type of TD varies after the migration.

In this work, we report a case study where we monitored the TD of a small and medium enterprise that migrated their legacy monolithic system to an ecosystem of microservices. We monitored the evolution of TD by using SonarQube[1], the most commonly used tool to analyze code TD. The analysis focused on the three types of TD proposed by SonarQube: reliability remediation cost (time to remove all the issues that can generate faults), maintainability remediation cost (time to remove all the issues that increase the maintenance effort) and security vulnerability remediation cost (time to remove all the security issues). Moreover, we performed a qualitative study performing a focus group with two development teams, the software architect, and the product manager to deeply understand the cause of the change of the TD and the motivations for possible postponed activities.

To the best of our knowledge, no studies exist on the impact of postponed activities on the TD, especially in the context of microservices. This work will help companies to understand how TD grows and changes over time while at the same time opening up new avenues for future research on the analysis of TD interest.

This paper is structured as follows: Section 2 briefly introduces the background and related works on microservices and TD. In Section 3, we present the case study design defining the research questions, describing the study context with the data collection and data analysis protocol. In Section 4, we show the achieved results followed by a discussion of them in Section 5. Section 6 identifies the threats to the validity of our study and

[1]SonarQube: www.sonarqube.org

Section 7 draws conclusions and possible future works.

2. Background and Related Work

2.1. Microservices

Microservice architecture has become more and more popular over the last years. Microservices are small, autonomous, and independently deployed services, with a single and clearly defined purpose [7]. The independent deployment provides a lot of advantages. They can be developed in different programming languages, they can scale independently from other services, and they can be deployed on the hardware that best suits their needs. Moreover, because of their size, they are easier to maintain and more fault-tolerant since the failure of one service will not break the whole system, which could happen in a monolithic system. Another characteristic of microservices is, as they are cloud native applications, the support of the IDEAL properties: Isolation of state, Distribution, Elasticity, Automated management, and Loose Coupling [7]. Moreover, microservices propose to vertically decompose the applications into a subset of business-driven services. Every service can be developed, deployed and tested independently by different development teams, and by means of different technology stacks. The responsibility of the development of a microservice belongs only to a single team, which is in charge of the whole development process, including deploying, operating and upgrading the service when needed. At the best of our knowledge, no work has investigated the TD or compared the quality between monolithic systems and microservices.

2.2. Technical Debt

The concept of TD was introduced for the first time in 1992 by Cunningham as *"The debt incurred through the speeding up of software project development which results in a number of deficiencies ending up in high maintenance overheads"* [4].

Different approaches and strategies have been suggested to evaluate TD. Nugroho et al.[8] proposed an approach to quantify debts in terms of cost to fix technical issues and the its interest. They monitored data from 44 systems. They empirically validated the approach in a real system. Seaman et al. [9] proposed a TD management framework that formalizes the relationship between cost and benefit in order to improve software quality and help decision making process during maintenance activities. Zazworka et al. [10] investigated automated identification of TD. They asked the developers to identify TD items during the development process and compared the manual identification with the results from automatic detection. Zazworka et al. [11] examined source code analysis techniques and tools to identify code debt in software systems, focusing on TD interest and TD impact on increasing defect- and change-proneness. They applied four TD identification techniques (code smells, automatic static analysis issues, grime buildup, and modularity violations) on 13 versions of the Apache Hadoop open source software project. They collected different metrics, such as code smells and code violations. The results showed good correlation between some metrics and defect- and change-proneness, such as Dispersed Coupling and modularity violations. Guo et al. [12] investigated the TD cost of applying a new approach on an on-going software project. They found a higher start-up cost that decreased over time.

Different approaches or strategies have been proposed to manage TD. Fa-

lessi et al. [13] proposed CMMI Maturity Level 5 company to manage TD. Guo et al. [14] proposed a portfolio approach in order to help the software manager in decision making. This approach provides a new perspective for TD management. Nord et al. [15] defined a measurement-based approach to develop metrics in order to strategically manage TD. This approach could optimize the development cost over time without stopping the development process. They successfully applied the approach to an ongoing system development effort. Ernst et al. [16] conducted a survey among 1,831 highly experienced participants from three large organizations. The results showed the harmfulness of important architectural decisions in leading TD issues.

Some secondary studies on managing TD were conducted over the last years. Li et al. [5] investigated TD management strategies conducting a systematic mapping study among 94 studies. They classified TD type, activities and evaluation tools. They identified a lack of empirical studies on the strategies and approaches to manage TD. Ampatzoglou et al. [17] analyzed the financial aspect of TD in order to better manage it. They conducted a systematic literature review on financial aspects definition and on software engineering concepts, among 69 studies. They provided a classification schema of the financial approaches applied to manage TD. Alves et al. [18] investigated TD management strategies focusing on their maturity levels. They conducted a systematic mapping study evaluating 100 studies. They also highlighted some points still requiring further investigation.

3. Case Study Design

In this case study, we compared the Technical Debt (TD) and its trend in a project before and after migrating to microservices. The study was

made based on the following research questions (**RQs**):

RQ1: Is the TD of a monolithic system growing with the same trend as a microservices-based system?

RQ2: Does the distribution of TD Issue types (bugs, code smells and security vulnerabilities) change after the migration to microservices?

3.1. Context

We monitored the development of a document management system developed by an Italian small and medium enterprise. The system is being developed in Java, deployed on the Microsoft Azure Cloud, and delivered to the customers as a web application, together with a desktop application to support document uploading and synchronization. The monolithic system was composed of 280K lines of code and had been developed for more than twelve years. The company decided to migrate to microservices in order to make maintenance easier by separating each business process and to reduce the need for synchronization between the two development teams. The company started to discuss about the migration to microservices in June 2016, evaluating the different architectural solutions. In January 2017, a team composed of four developers and another one composed of five developers, started the migration extracting a set of features, related to a business process, into a new microservice. We monitored the migration process until September 2018, where the teams extracted five business processes from the monolithic system into five independent microservices.

3.2. Study Execution and Data Collection

The study was performed in two steps. First, we collected data on the TD of the system before and during the migration. Then, we interviewed the developers to ask their feedback on the results.

We collected data on the TD by analyzing each commit over the two years prior and during the migration. The TD was analyzed by means of SonarQube[2] (version 7.0), the most commonly adopted static code analysis tool, by applying the standard quality profile. We analyzed the TD provided by SonarQube considering the distribution of TD issue types classified by SonarQube as "Code Smells", "Bugs" and "Security Vulnerabilities". Moreover, since the TD is calculated as a sum of remediation time for each TD issue, we calculated the remediation time for each type of TD issue.

The second step was conducted using a focus group, based on a face-to-face semi-structured interview. The goal of the focus group was to discuss if the change to microservices was beneficial, and whether they experienced any of the benefits expected from the migration to microservices. The focus group was moderated by one of the authors. Each participant reported their answers on Post-it notes, and stitched them to a white board. Then, each participant described their answer and grouped the answer together with similar ones, with the help of the group discussion. Finally, each participant ranked the importance of the answer from 1 to 5, where 5 means very important and 1 not important. We structured the questions of the interview into three main questions:

Q1: What benefits and issues have you perceived after the migration to microservices? Please write the issues on the red Post-it notes and the benefits on the yellow ones. The goal of this question is to understand if the results can be biased because of some issues in the application of the microservices pattern.

Q2: Has the quality of the code increased after the migration to microser-

[2]https://www.sonarqube.org

vices?

Q2.1: Which component has the highest and which has a lower quality? Please write the ones with the highest quality on red Post-it notes, and the one with the lower quality on the yellow Post-it notes

Q3: Has the overall TD decreased after the migration to microservice? Please motivate the answer on the Post-it notes. *Q4: Have you postponed any technical activities during the migration to microservices?* Please, report the postponed activities (or group of activities) on the Post-it notes.

3.3. Data Analysis

In all of the done analyses, for before migration we use data from 23rd August 2014 to 30th April 2016, and for after the migration from 3rd January 2017 - 20th September 2018. The time period between May and December 2016 is left out, as during the time the monolithic system was only maintained. This resulted as an approximately constant amount of TD, as can be seen from Figure 1.

In order to answer RQ1, we analyze the growth of the total TD in minutes before and after migrating to microservices. The rate of the TD growth is analyzed using linear regression, and a line is fitted to the data before and after migrating to microservices. The growth rates of the two regression lines are compared by inspecting the slope coefficients of the lines. The R^2 values for the regression lines are also determined.

For RQ2, we determine the relative proportion of each TD type of the total TD in each commit. It is then determined whether the relative distribution of TD types is statistically different before and after migrating to microservices by applying the Mann-Whitney test. It is a non-parametric test, for which the null hypothesis is that the distributions for both tested

groups are identical and there is an even probability that a random sample from one group is larger than a random sample from the other group [19]. The results are considered statistically significant if the p-value is smaller than 0.01.

In order to determine the magnitude of the measured differences, we use Cliff's Delta, which is a non-parametric effect size test for ordinal data. The results are interpreted using guidelines provided by Grissom and Kim [20]. The effect size is small if $0.100 \leq |d| < 0.330$, medium if $0.330 \leq |d| < 0.474$, and large if $|d| > 0.474$.

4. Results

The TD of the monolithic system before the migration is lower than the TD immediately after the migration (Figure 1). This is probably due to the need of writing the code connecting the monolithic system to the microservice. After a limited time, the TD of each microservice tends to be more stable and to increase slower in each microservice compared to the TD of the whole system. Immediately after the migration, the total TD (sum of the TD of the monolithic system and all the microservices) grows faster compared to the growth of the TD before the migration. However, TD in each microservice tends to decrease after a relatively short time. The result is that, once the TD is stabilized after the extraction of a feature from the monolithic system as new microservice, the TD trend grows much slower than before the migration.

Figure 1 shows the linear regression lines fitted to the data before and after the migration of microservices. The number of data points, slope coefficients for the regression lines, and their R^2 values are reported in Table 1.

The table shows that the slope coefficient drops significantly after migrating to microservices, and the coefficient after migration is 89.76% smaller than when using monolithic systems. This implies a remarkable drop in the TD growth rate.

Since the vast majority of business processes are still in the monolithic system, we expect the TD of the whole system to be lower than the TD of the monolithic system after all of the business processes have been migrated.

Table 1: TD slope coefficients before and after the migration to microservices.

	n	**Slope coeff.**	R^2
Before migration	626	14.709	0.93
After migration	617	1.5064	0.64

Table 2: The descriptive statistics for TD types before and after the migration and the results from Mann-Whitney and Cliff's Delta tests.

		Type		
		Code smell	Bug	Vulnerability
Before migration	mean	0.95	0.012	0.035
	median	0.95	0.012	0.037
	stdev	0.01	0.003	0.005
After migration	mean	0.97	0.004	0.028
	median	0.97	0.005	0.027
	stdev	0.01	0.001	0.005
Mann-Whitney	U	18,066	0	57,269
	p-value	0	0	0
Cliff's Delta	d	-0.91	1	0.70
	CI	-0.93 - -0.88	1.00 - 1.00	0.66 - 0.74

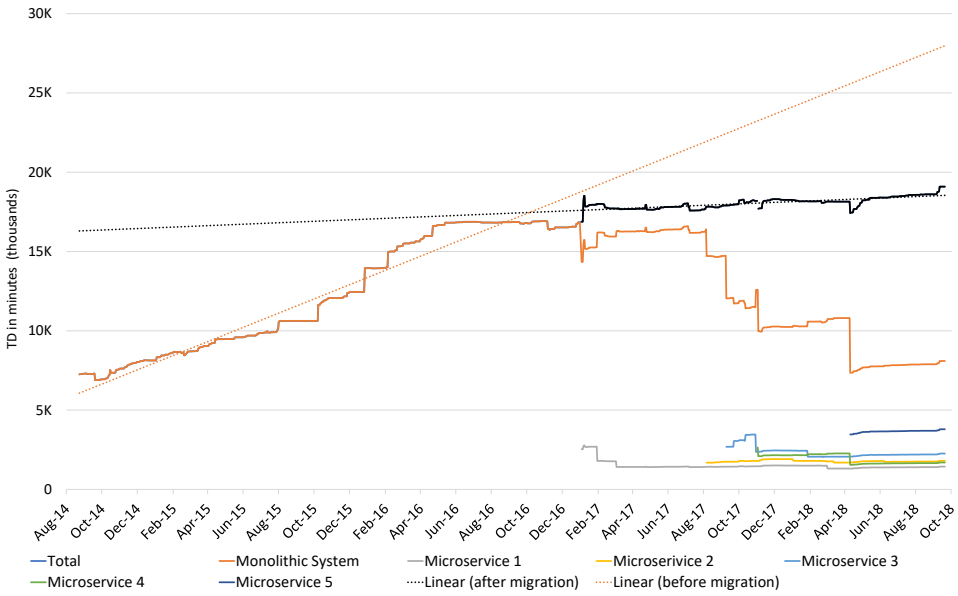

Figure 1: TD evolution of the five microservices and the monolithic system

Figure 2 shows the relative distribution of the different TD types detected by SonarQube while Table 2 presents the descriptive statistics before and after migrating to microservices.

In order to determine if the change in the relative distributions for the different types was statistically significant, we applied the Mann-Whitney test for each type and the results are reported in Table 2. As the p-values are smaller than 0.01 for all types, we conclude that the relative distributions of TD types do change after migrating to microservices. The results from the Cliff's Delta test are also presented in Table 2. All measured values of $|d|$ were greater than 0.474 and thus the effect size is considered large for all types. Also the confidence intervals (CI) for d using 0.99 confidence are greater than 0.474.

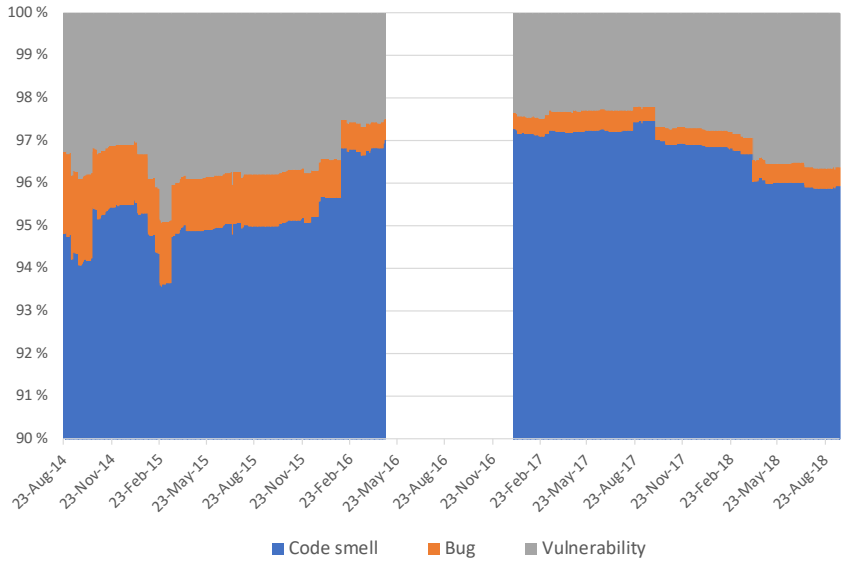

Figure 2: Relative distribution of the TD types.

The focus group helped to better understand the reason of the TD evolution. Eleven members of the company participated to the focus group: four developers from one team, five developers from an another team, the software architect, and the project manager.

The main benefit perceived by the participants was the decrease of the bug fixing time and a big increase of the system understandability. Other minor benefits were the reduced need of synchronization between teams and the possibility to deploy new features without re-compiling and re-deploying the whole system.

As for the main issues, the project manager and the software architect highlighted a higher development cost, probably because the developers used microservices for the first time. Developers highlighted the complexity of connecting the different microservices to each other or to the monolithic system. The monolithic system had the advantage of local calls, while with

microservices they had to rely on a distributed system.

As for the component quality, they highlighted the new five microservices as the components with the highest quality, even tough the developers pointed out that the migration did not involve a complete rewriting of the code, but they largely reused the existing code, with a certain amount of refactoring.

As for the amount of perceived TD, developers, the project manager, and the software architect have different opinions. The software architect and the project manager perceived the increase of effort as a negative issue, since the overall development cost increased by more than 20%. However, the software architect reported that several technical decisions have been postponed during the development, as releasing new features requested by customers was more important. As an example, in two microservices the team kept the old SQL database in place, despite it was planned to migrate the database (SQL Server) to a NoSQL database. Another issue was in the outdated libraries in the existing monolithic system. Upgrading the library to a newer version would have required several changes in the code, and the team decided to postpone this activity because of time issues. Yet another important postponed activity is the refactoring of the code extracted into the microservice 5 (See Figure 1). The code was extracted from the monolithic system with the intention of deeply refactoring it before using the code in production. However, an urgent new feature request that impacted the microservice 5 resulted in the postponement of the refactoring. The team reported that they had the choice of implementing the new feature in the monolithic system and then apply all the changes to the almost migrated microservice, or to implement the new feature directly in the new microservice. Another important postponed decision is the development of

an API-Gateway. The company actually used the RabbitMQ message bus as API-Gateway, but they are planning to migrate to a proper API-Gateway in the future. Overall, despite all the voluntarily postponed activities, all the participants perceived the overall TD as decreased.

5. Discussion

The migration to microservices is a non-trivial task that requires a deep re-engineering of the whole system that heavily impacts on the whole project cost, but should also ease the maintenance in the long run. The case study on the migration to microservice provided both unexpected and expected results. On one hand, we expected to have significant decrease of the maintenance predictors, reducing the remediation effort related to the "code smells". On the other hand, we confirmed that, in the long run, the total code TD is growing slower after the migration to microservices.

The qualitative analysis also helped us to draw three lessons learned that can help others to reduce TD while migrating to microservices:

1. The creation of microservice template is important. The initial increase of TD after the introduction of each microservice is probably due to the initial lack of a service template. The second microservice was not easy to develop, but it was easier than developing the first one because of the availability of the template developed in the first microservice. We recommend to invest more effort on the definition of a set of service templates, as it will dramatically ease the development of new microservices in the future

2. The identification of the migration strategy must be defined upfront, even if the company is using an agile methodology. A clearly defined a migration strategy helps to avoid rework.

3. Do not postpone architectural decisions. Continuous architects recommend to postpone architectural decisions until they are really needed [21]. However, we experienced an important increase of effort to re-architect the systems due to postponed architectural decisions. An example was the usage of the lightweight message bus (RabbitMQ) as API-Gateway instead of using a proper API-Gateway. The SME of our case study is still using RabbitMQ as API-Gateway, but they are aware that the implementation of a proper API-Gateway from the beginning would have cost much less than migrating from RabbitMQ after two years.

6. Threats to Validity

In this Section, we introduce the threats to validity, following the structure suggested by Yin [22], reporting construct validity, internal validity, external validity, and reliability. Moreover, we will also debate the different tactics adopted to mitigate them.

Construct Validity concerns the identification of the measures adopted for the concepts studied in this work. We analyzed the TD using the model provided by SonarQube. Therefore, different tools and approaches might provide different results. We are aware that other types of TD, such as requirements TD or architectural TD, can heavily impact the TD of a system. However, SonarQube was the only tool that we were allowed to use in the company, since they already had it in place in their DevOps tool-set. We are also aware that important postponed activities could have created a big amount of TD. We mitigate this threat by performing the focus group and by asking the teams to discuss on other possible types of TD. A more thorough discussion on the amount of TD that these postponed activities will generate

will be part of our future work. Threats to *Internal Validity* concern factors that could have influenced the obtained results. The postponed activities were collected using a group discussion. It is possible that some developers did not want publicly expose some activities they had postponed. Threats to *External Validity* concern the generalization of the obtained results. The result of this paper are based on the monitoring of the development process of a single company. The results could be slightly different in other companies. However, based on previous studies on microservices, developers confirm that microservices increase the maintenance easiness and increase code readability and system understandability [1]. Therefore, we expect that also other systems could benefit of a decrease of the TD when migrating to microservices. Threats to *Reliability* refers to the correctness of the conclusion reached in the study. This study was a preliminary study, and therefore we applied simple statistical techniques to compare the trends of the TD before and after the migration. Results of the statistical technique applied is also confirmed by Figure 1 and Figure 1. We are aware that more accurate statistical techniques for time series could have provided a more accurate estimate of the difference of the slopes. However, we do not expect that different statistical techniques could provide an opposite result (TD decrease of 89.76% after the introduction of microservices).

7. Conclusion

In this work we compared the Technical Debt (TD) before and after the migration to microservices of a big software project (280K lines of code) developed by a small and medium enterprise.

We conducted a case study analyzing the code TD of the system under

development for a time-frame of four years (two years before and for two years after the migration). Then we conducted a focus group to analyze in-deep the postponed activities and to get more insights on the results. The first immediate result is that TD grows 90% slower after the development of microservices. After the initial introduction of each microservice, TD grows for a limited period of time, mainly because of the new development activities. When the code of the microservice stabilizes, TD decreases and starts growing linearly, with a growing trend much lower than the monolithic system.

Unexpectedly, when comparing the distribution of TD types before and after the introduction of microservices, important and statistically significant differences emerge. The proportion of SonarQube rules classified as bugs and security vulnerabilities decreased while code smells (maintainability issues) increased. Since microservices are supposed to ease software maintenance, we expected a reduction of code smells.

Developers confirmed the overall results, perceiving a reduced maintenance complexity. The overall development effort increased after the introduction of microservices, probably because of the extra effort due to the re-development of the system. However, the manager confirmed that the increased velocity and the increased team freedom compensate for the extra effort required.

Future work includes the investigation of the impact of other types of TD during the migration to microservices. We aim at analyzing the same projects using tools detecting architectural smells. Moreover, we aim at investigating the TD due to temporary architectural decisions. Our next goal is to understand how long different activities could be postponed before the benefit of postponing an activity is canceled out by the increased

effort needed to refactor it. As an example, if an activity has an interest equal to zero (i.e., if the development/refactoring effort does not increase if postponed), it can be postponed until it is needed, whereas if an activity has a monthly interest of 10% (i.e., 10% extra interest per month), it should be refactored as soon as possible before getting too expensive.

References

References

[1] D. Taibi, V. Lenarduzzi, C. Pahl, Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation, IEEE Cloud Computing 4 (2017) 22–32.

[2] D. Taibi, V. Lenarduzzi, P. C., Architectural patterns for microservices: A systematic mapping study, in: 8th International Conference on Cloud Computing and Services Science (CLOSER), p. 8.

[3] H. Knoche, W. Hasselbring, Using microservices for legacy software modernization, IEEE Software 35 (2018) 44–49.

[4] W. Cunningham, The wycash portfolio management system, in: OOPSLA '92.

[5] Z. Li, P.Avgeriou, P. Liang, A systematic mapping study on technical debt and its management, Journal of Systems and Software 101 (2015) 193 – 220.

[6] J. Soldani, D. A. Tamburri, W.-J. V. D. Heuvel, The pains and gains of microservices: A systematic grey literature review, Journal of Systems and Software 146 (2018) 215 – 232.

[7] S. Newman, Building Microservices, O'Reilly Media, Inc., 1st edition, 2015.

[8] A. Nugroho, J. Visser, T. Kuipers, An empirical model of technical debt and interest, in: Workshop on Managing Technical Debt, MTD '11, pp. 1–8.

[9] C. Seaman, Y. Guo, Chapter 2 - measuring and monitoring technical debt, volume 82 of *Advances in Computers*, Elsevier, 2011, pp. 25 – 46.

[10] N. Zazworka, R. O. Spínola, A. Vetro', F. Shull, C. Seaman, A case study on effectively identifying technical debt, in: Int. Conf. on Evaluation and Assessment in Software Engineering, EASE '13, pp. 42–47.

[11] N. Zazworka, A. Vetro', C. Izurieta, S. Wong, Y. Cai, C. Seaman, F. Shull, Comparing four approaches for technical debt identification, Software Quality Journal 22 (2014) 403–426.

[12] Y. Guo, R. Spínola, C. Seaman, Exploring the costs of technical debt management – a case study, Empirical Software Engineering 21 (2016) 159–182.

[13] D. Falessi, M. A. Shaw, F. Shull, K. Mullen, M. S. Keymind, Practical considerations, challenges, and requirements of tool-support for managing technical debt, in: Int. Workshop on Managing Technical Debt (MTD), pp. 16–19.

[14] Y. Guo, C. Seaman, A portfolio approach to technical debt management, in: Workshop on Managing Technical Debt, MTD '11, pp. 31–34.

[15] R. L. Nord, I. Ozkaya, P. Kruchten, M. Gonzalez-Rojas, In search of a

metric for managing architectural technical debt, in: WICSA-ECSA, pp. 91–100.

[16] E. A. Neil, S. Bellomo, I. Ozkaya, R. L. Nord, I. Gorton, Measure it? manage it? ignore it? software practitioners and technical debt, in: Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, pp. 50–60.

[17] A. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, P. Avgeriou, The financial aspect of managing technical debt: A systematic literature review, Information and Software Technology 64 (2015) 52 – 73.

[18] N. Alves, T. Mendes, M. de Mendonça, R. O. Spínola, F. Shull, C. Seaman, Identification and management of technical debt: A systematic mapping study, Information and Software Technology 70 (2016) 100 – 121.

[19] P. E. McKnight, J. Najab, Mann-whitney u test, The Corsini encyclopedia of psychology (2010) 1–1.

[20] R. J. Grissom, J. J. Kim, Effect sizes for research: A broad practical approach., Lawrence Erlbaum Associates Publishers, 2005.

[21] M. Erder, P. Pureur, Continuous Architecture: Sustainable Architecture in an Agile and Cloud-Centric World, Morgan Kaufmann, Amsterdam, 2016.

[22] R. Yin, Case Study Research: Design and Methods, 4th Edition (Applied Social Research Methods, Vol. 5), SAGE Publications, Inc, 4th edition, 2009.