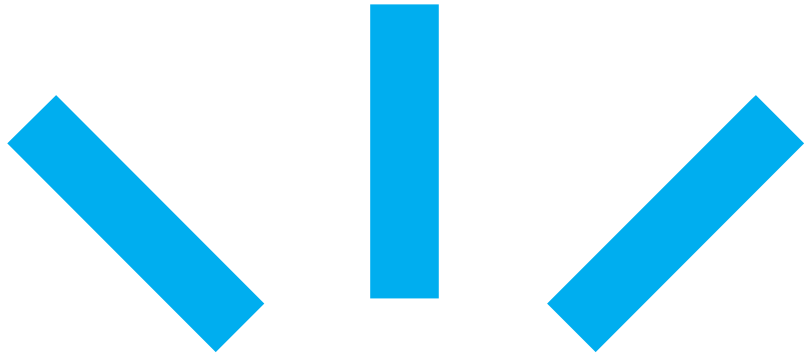


Technical Debt and Microservices

Valentina Lenarduzzi, Ph.D.
University of Oulu

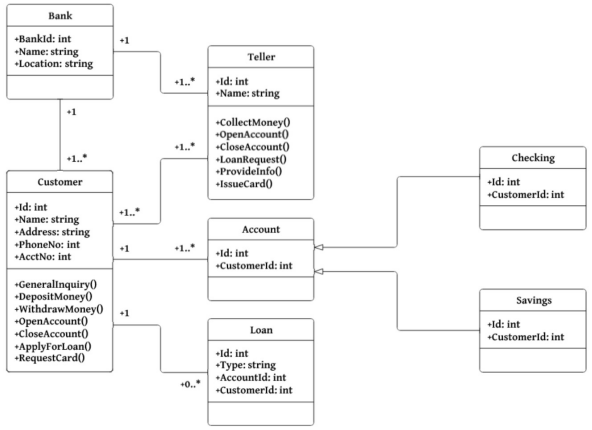
- Assistant Professor Tenure Track (University of Oulu – Finland)
- Technical Debt, Software Quality, Maintenance and Evolution
  - Microservices Bad Smells definition
  - Processes, Motivations, and Issues for Migrating to Microservices
- 19th in the earlier stage career in software engineering domain*

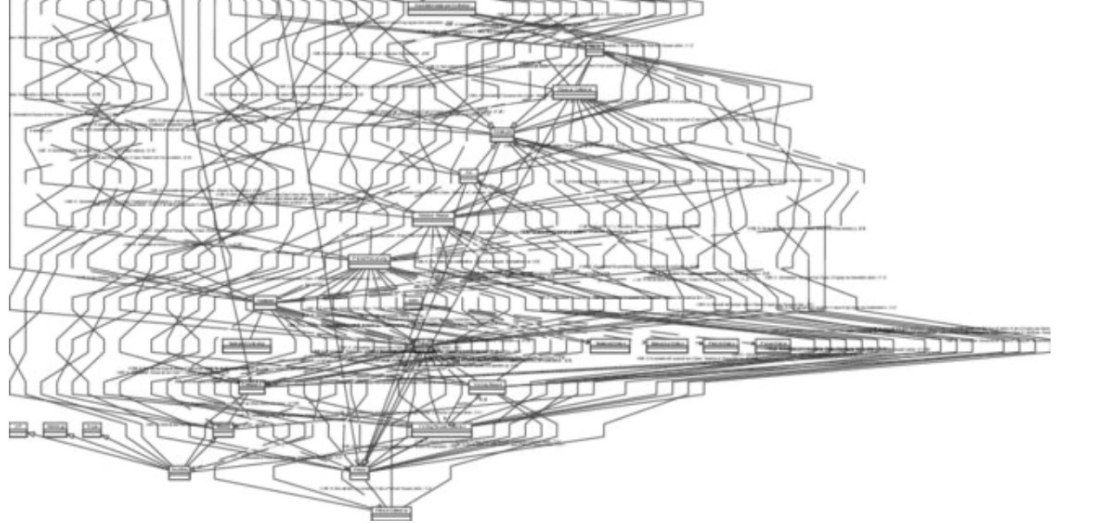
*W. Eric Wong, Nikolaos Mittas, Elvira Maria Arvanitou, Yihao Li. A bibliometric assessment of software engineering themes, scholars and institutions (2013–2020), Journal of Systems and Software, Volume 180, 2021,


mirror_mod.use_x = False
mirror_mod.use_y = True
mirror_mod.use_z = False
elif operation == "MIRROR_Z":
mirror_mod.use_x = False
mirror_mod.use_y = False
mirror_mod.use_z = True
selection at the end -add back the deselected mirror modifier object
mirror_ob.select= 1
modifier_ob.select=1
bpy.context.scene.objects.active = modifier_ob
print("Selected" + str(modifier_ob)) # modifier ob is the active ob

Software Evolves

Software Degradates





Migration to Microservice

- Migration prioritization
  - Only new features (Strangler pattern)
  - Most problematic features
  - Less problematic features

Strangler Pattern

- Only new features are implemented as microservices
- The core of the software will be never “strangled”
- Lack of complete overview on the migration path

Legacy business processes need to be reengineered

Aging Microservices

- The oldest Microservices are aging
  - Becoming the legacy

REFATORING
IS FUNDAMENTAL

But …

Posponed Activites

- **During Rearchitecting / refactoring / evolution of systems several activities are postponed**
  - Lack of time
  - Lack of resources
  - New Features are prioritized

Olaf Zimmermann, Mirko Stocker, Daniel Lübke, Cesare Pautasso, Uwe Zdun. **Microservice API Patterns.** https://microservice-api-patterns.org

Davide Taibi, Valentina Lenarduzzi and Claus Pahl. **Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation**. *IEEE Cloud Computing*, vol. 4, no. 5, pp. 22-32, 2017

The dark side of Developing

Technical Debt

Ward Cunningham

"Shipping first time code is like going into debt”

“A little debt speeds development so long as it is paid back promptly with a rewrite...”

“Every minute spent on not-quite-right code counts as interest on that debt”

Technical Debt Definition

Debt = sub-optimal solution

Save time by non-applying the optimal solution

- You gain a benefit now (borrow money)
- But, you pay the consequences later (you will pay the interest)

Technical Debt, why?

- People commonly check their health (blood analysis, XRays …)
- Machines are commonly checked for their health (

Why they do not do with code, architecture ….?

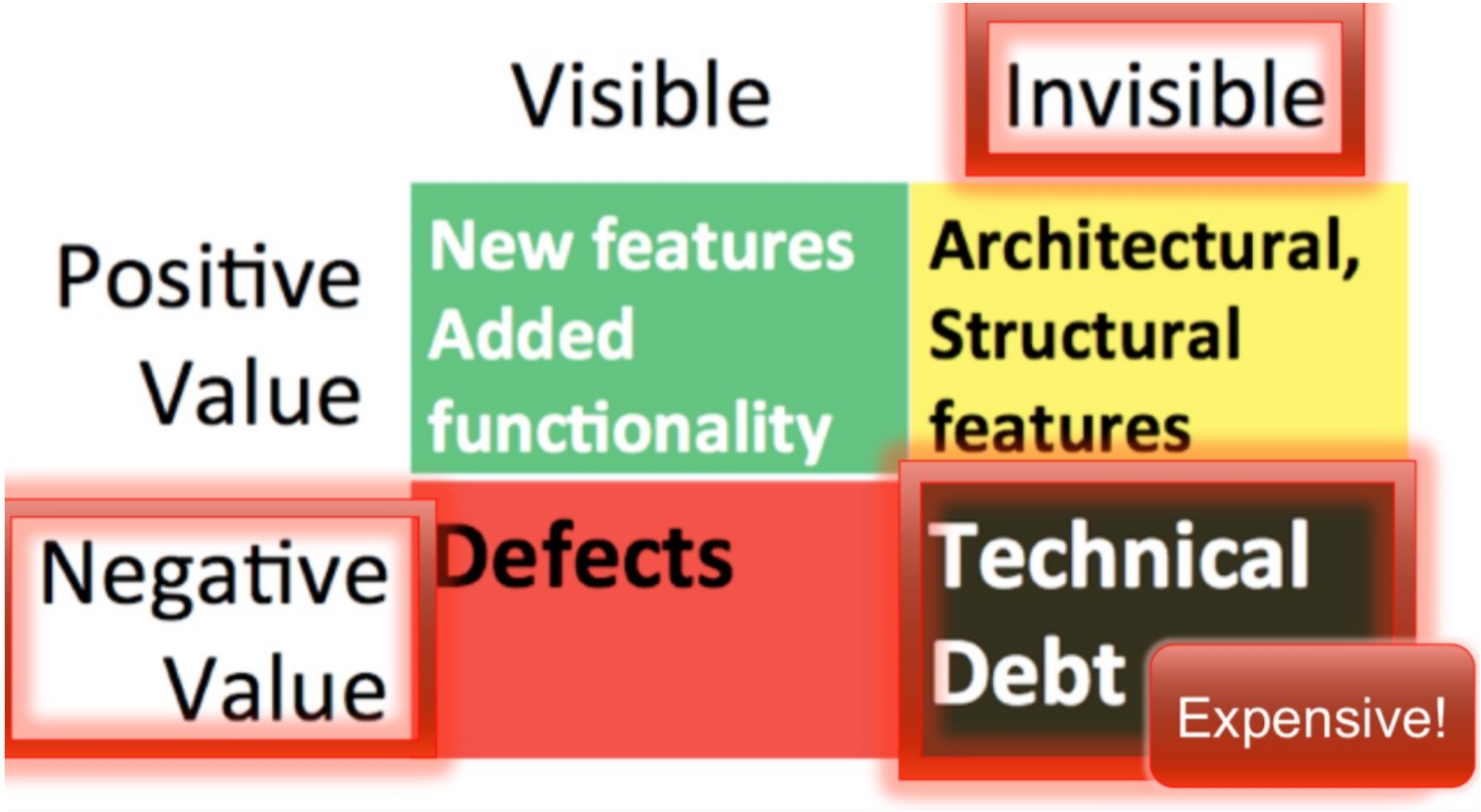
- Having a continuous check since from the beginning of the development process can prevent issues that could became unmanageable if you do not react immediately

Technical Debt

The Debt Metaphor

- Use a credit card to obtain something now (short term)
- Pay for it later (future payment due)
- Plus interest (the cost of being able to do this)

Technical Debt



P. Kruchten, R. L. Nord, and I. Ozkaya. ***Technical Debt: From Metaphor to Theory and Practice***. IEEE Software

Technical Debt

Principal

- Cost of fixing problems remaining in the code after release that must be remediated

Interest

- Continuing IT costs attributable to the violations causing technical debt, i.e, higher maintenance costs, greater resource usage, etc.

Technical Debt

The consequences of:

- Slapdash architecture
- Poor design
- Hasty coding (versus rapid)
- Lack of quality focus
- Others?

The danger occurs when the debt is not repaid quickly. Every minute spent on not-quite-right code results in interest on that debt.

Technical Debt

$$\sum Time\ to\ solve\ violation$$

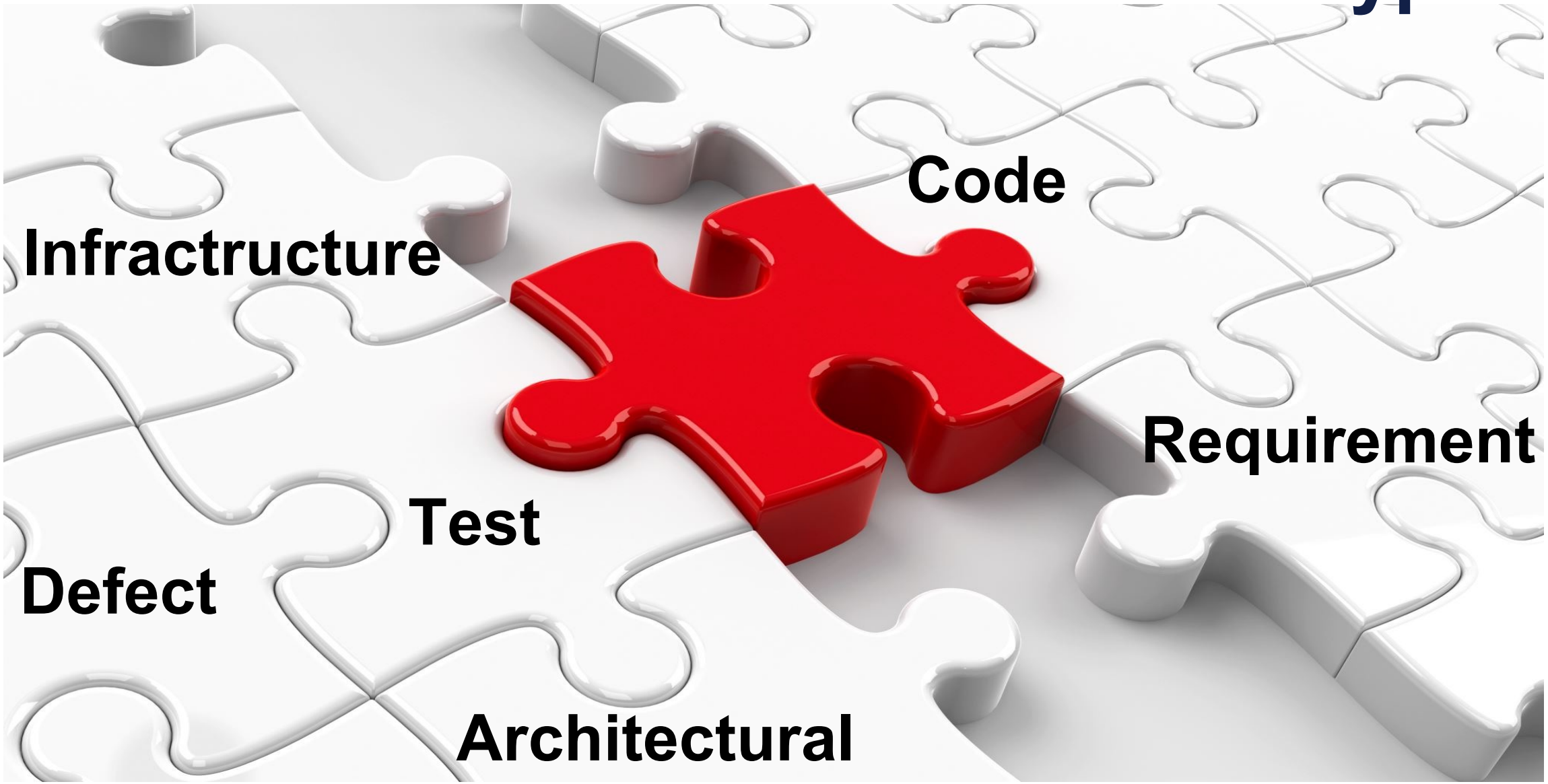
Violations include:

- Syntactic violations;
- Smells,
- Other Violations considered “harmful” by the TD tool vendor

Technical Debt Issues

- Not only one Technical Debt
- Technical Debt is unavoidable
- What to do first?
  - Prioritization

Technical Debt Type



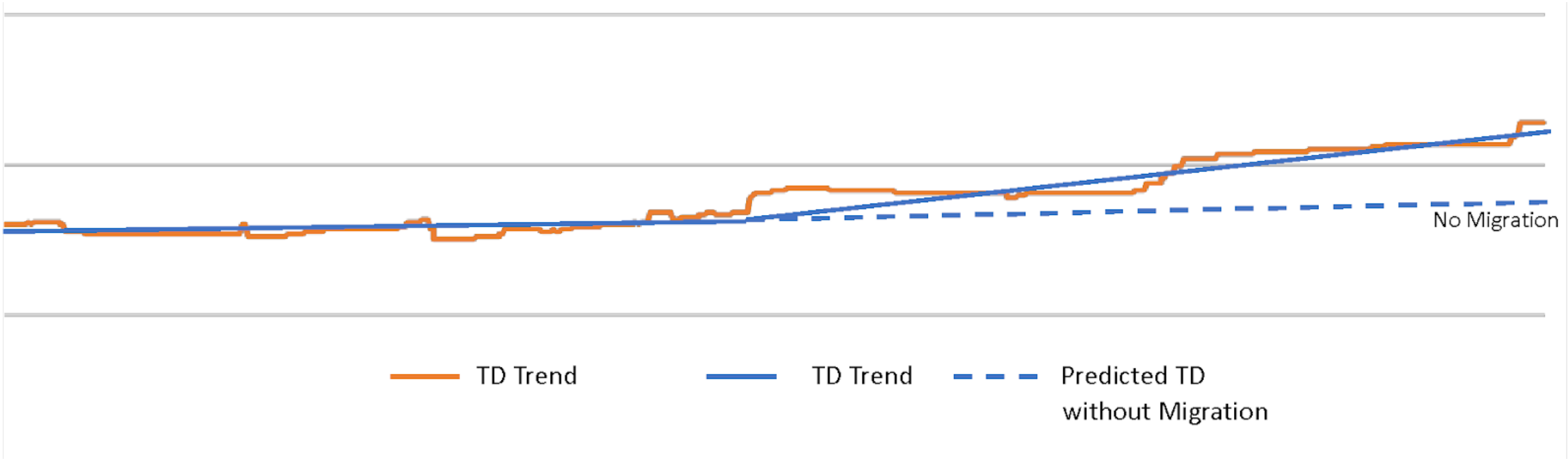
Z. Li, P. Avgeriou, P. Liang. ***A systematic mapping study on technical debt and its management.*** Journal of Systems and Software (2015)

Architectural Debt

- **Architectural Degradation**
  - Introduction of architectural smells
  - Violation of architectural guidelines
- **Postponed architectural decisions**

Code Debt

- TD increases when migrating to Microservices
- Several moving parts, more code, potentially more issues



Valentina Lenarduzzi, Francesco Lomio, Nyyti Saarimäki, Davide Taibi. **Does migrating a monolithic system to microservices decrease the technical debt?** Journal of Systems and Software, Volume 169, 2020.

Testing Debt

- Testing is more complex.
- Several companies only perform unit test and end-to end test.
- Regression tests are too expensive.
- Hybrid test is often performed
- Some services are hard to test (mocking is not always possible)

Infrastructure Debt

- Lower in Microservices
- Infrastructure before starting the development

How to start

FOCUS: MICROSERVICES

On the Definition of Microservice Bad Smells

Davide Taibi and Valentina Lenarduzzi, Tampere University of Technology

// *To identify microservice-specific bad smells, researchers collected evidence of bad practices by interviewing developers experienced with microservice-based systems. They then classified the bad practices into 11 microservice bad smells frequently considered harmful by practitioners.* //

MICROSERVICES ARE CURRENTLY enjoying increasing popularity and diffusion in industrial environments, being adopted by several big players such as Amazon, LinkedIn, Netflix, and SoundCloud. Microservices are relatively small and autonomous services that work together, are modeled around a business capability, and have a single and clearly defined purpose.[1,2] Microservices enable independent deployment, allowing small teams to work on separated and focused services by using the most suitable technologies for their job that can be deployed and scaled independently.[1,2] Microservices are a newly developed architectural style. Several patterns and platforms such as nginx (www.nginx.org) and Kubernetes (kubernetes.io) exist on the market. During the migration process, practitioners often face common problems, which are due mainly to their lack of knowledge regarding bad practices and patterns.[3,4]

In this article, we provide a catalog of bad smells that are specific to systems developed using a microservice architectural style, together with possible solutions to overcome these smells. To produce this catalog, we surveyed and interviewed 72 experienced developers over the course of two years, focusing on bad practices they found during the development of microservice-based systems and on how they overcame them. We identified a catalog of 11 microservice-specific bad smells by applying an open and selective coding[5] procedure to derive the smell catalog from the practitioners' a

The goal of th practitioners avoi tices altogether o more efficiently w migrating monolit based systems.

As with code smells, which a monly considered design,[1,6] we de specific bad smel service smells" indicators of situa desired patterns, a practices—that software quality understandability, bility, reusability, of the system unde

Background

Several generic detection tools a been defined in Moreover, sever specific architect been defined.[10] H of our knowledge work and, in parti studies have prop antipatterns, or concerning micros

However, some started to discus microservices. In *services AntiPatt*

Smells and Refactorings for Microservices Security: A Multivocal Literature Review

FRANCISCO PONCE, Universidad Técnica Federico Santa María, Chile
JACOPO SOLDANI, University of Pisa, Italy
HERNÁN ASTUDILLO, Universidad Técnica Federico Santa María, Chile
ANTONIO BROGI, University of Pisa, Italy

Context: Securing microservice-based applications is crucial, as many IT companies are de businesses through microservices. If security "smells" affect microservice-based applications, the suffer from security leaks and need to be **refactored** to mitigate the effects of security smells th

Objective: As the currently available knowledge on securing microservices is scattered across di of white and grey literature, our objective here is to distill well-known smells for securing together with the refactorings enabling to mitigate the effects of such smells.

Method: To capture the state of the art and practice in securing microservices, we conducted review of the existing white and grey literature on the topic. We systematically analyzed 58 stu from 2014 until the end of 2020.

Results: Ten bad smells for securing microservices are identified, which we organized in a taxonon each smell with the security properties it may violate and the refactorings enabling to mitigate

Conclusions: The security smells and the corresponding refactorings have pragmatic value for who can exploit them in their daily work on securing microservices. They also serve as a star researchers wishing to establish new research directions on securing microservices.

1 INTRODUCTION

Microservices are on the rise for architecting enterprise applications nowadays, with b IT (e.g., Amazon, Netflix, Spotify, and Twitter) already delivering their core busines microservices [80]. This is mainly because microservice-based applications are cloud better exploiting the potentials of cloud hosting, and since they fully twin with continuous delivery practices [2]. Microservices also bring various other advantages, of deployment, resilience, and scalability [52]. Together with their gains, however, bring also some pains, and securing microservice-based applications is certainly one of those [76].

Microservice-based applications are essentially service-oriented applications adhering to an extended set of design principles [88], e.g., shaping services around business concepts, decentralisation, and ensuring the independent deployability and horizontal scalability of microservices, among others. Such additional principles make microservice-based applications not only service-oriented, but also highly distributed and dynamic. As a result, other than the classical security issues and best-practices for service-oriented applications, microservices bring new security challenges [76]. For instance, being much more distributed than traditional service-oriented applications, microservice-based application expose more endpoints, thus enlarging the surface prone to security attacks [39]. It is also crucial to establish trust among the microservices forming an application and to manage distributed secrets, whereas these concerns are of much less interest in traditional web services or monolithic applications [85]. Another example follows from the many communications occurring among the microservices forming an application, which —if not properly handled— can

Authors' addresses: Francisco Ponce, francisco.ponceme@sansano.usm.cl, Universidad Técnica Federico Santa María, Valparaiso, Chile; Jacopo Soldani, jacopo.soldani@unipi.it, University of Pisa, Pisa, Italy; Hernán Astudillo, hernan@inf.utfsm.cl, Universidad Técnica Federico Santa María, Valparaiso, Chile; Antonio Brogi, antonio.brogi@unipi.it, University of Pisa, Pisa, Italy.

SICS Software-Intensive Cyber-Physical Systems (2020) 35:3–15
https://doi.org/10.1007/s00450-019-00407-8

SPECIAL ISSUE PAPER

Design principles, architectural smells and refactorings for microservices: a multivocal review

Davide Neri[1] · Jacopo Soldani[1] · Olaf Zimmermann[2] · Antonio Brogi[1]

Published online: 3 September 2019

Abstract
Potential benefits such as agile service delivery have led many companies to deliver their business capabilities through microservices. Bad smells are however always around the corner, as witnessed by the considerable body of literature discussing architectural smells that possibly violate the design principles of microservices. In this paper, we sy white and grey literature on the topic, in order to identify the most recognised architectural smells fo discuss the architectural refactorings allowing to resolve them.

Keywords Microservices · SOA · Architectural principles · Architectural smells · Refactorings

1 Introduction

Microservices architectures, first discussed by Lewis and Fowler [30], bring various advantages such as ease of deployment, resilience, and scaling [34]. Many IT companies deliver their core business through microservice-based solutions nowadays, with Amazon, Facebook, Google, LinkedIn, Netflix and Spotify being prominent examples. To deliver on their promises, microservices must be designed in quality and style, which is unfortunately not always the case [47].

Microservice-based architectures can be seen as peculiar extensions of service-oriented architectures, characterized by an extended set of design principles [39,55]. These principles include shaping services around business concepts, decentralising all development aspects of microservice-based solutions (from governance to data management), adopting a culture of automation, ensuring the independent deployability and high observability of micro failures [34]. A key research questio

How can architectural smells af ciples of microservices be detect refactoring?

The currently available informatio indicating possible violations of th microservices is scattered over a con erature. Unfortunately, this makes it body of knowledge on the topic, bot ing to investigate on microservices an working with them.

Our objective here is to systematic ature, in order to identify the most rec as architectural refactorings for reso ring in an application [54]. In partic design principles dealing with the d interactions between microservices process viewpoint, as per the 4+1 v More precisely, we consider the indep microservices, their horizontal scalab ures and decentralisation.

As recommended by Garousi e both the state of the art and the the field, we conducted a multivo of the existing literature, including (i.e., peer-reviewed papers) and gre posts, industrial whitepapers and be

✉ Davide Neri
davide.neri@di.unipi.it ; davide.neri@unipi.it
Jacopo Soldani
jacopo.soldani@unipi.it
Olaf Zimmermann
ozimmerm@hsr.ch
Antonio Brogi
antonio.brogi@unipi.it

[1] University of Pisa, Pisa, Italy
[2] University of Applied Sciences of Eastern Switzerland (HSR FHO), Rapperswil, Switzerland

Migrating towards Microservices: Migration and Architecture Smells

Andrés Carrasco
University of Antwerp
Antwerp, Belgium
andres.carrasco@student.uantwerpen.be

Brent van Bladel
University of Antwerp
Antwerp, Belgium
brent.vanbladel@uantwerpen.be

Serge Demeyer
University of Antwerp
Antwerp, Belgium
serge.demeyer@uantwerpen.be

ABSTRACT
Migrating to microservices is an error-prone process with deep pitfalls resulting in high costs for mistakes. Microservices is a relatively new architectural style, resulting in the lack of general guidelines for migrating monoliths towards microservices. We present 9 common pitfalls in terms of bad smells with their potential solutions. Using these bad smells, pitfalls can be identified and corrected in the migration process.

CCS CONCEPTS
• **Software and its engineering → Extra-functional properties**; *Software architectures*; *Software creation and management*;

KEYWORDS
Architecture Smells, Migration Smells, MicroServices, Literature Study

ACM Reference Format:
Andrés Carrasco, Brent van Bladel, and Serge Demeyer. 2018. Migrating towards Microservices: Migration and Architecture Smells. In *Proceedings of the 2nd International Workshop on Refactoring (IwoR '18), September 4, 2018, Montpellier, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3242163.3242164

1 INTRODUCTION

Microservices is an architectural style for developing an application in independently and automatically deployable services communicating with lightweight mechanisms [23]. The term microservices has become a buzzword nowadays. While some debate is still ongoing on whether microservices is an architectural style itself, or simply a way of doing Service-Oriented Architectures (SOA), there is a concrete distinction on its realization [61].

There is not a one-size-fits-all strategy for microservices, i.e., each solution has a different strategy in place. This plethora of strategies makes the outlining of its characteristics difficult. However, some characteristics are common among microservices, such as the componentization via services, smart endpoints with dumb pipes, and decentralization [23].

IwoR '18, September 4, 2018, Montpellier, France

ACM ISBN 978-1-4503-5974-0/18/09…$15.00
https://doi.org/10.1145/3242163.3242164

The microservices architectural style has grown in popularity for the last few years, due to its potential benefits, such as technology heterogeneity, resilience, scalability, eased deployment, productivity, reusability, and replaceability among others [51]. Moreover, some research has reported reduced complexity, lower coupling, higher cohesion, simpler integration, better reusability, and performance increase after migrating to a microservices architecture [9, 27]. However, the benefits of adopting a microservices architecture come with the complexities of distributed systems, such as the need for resilience, scaling, and data consistency [43]. Many new technologies have emerged in recent years for dealing with these complexities, such as containerization, automated deployment, and scaling of applications; these technologies are considered enablers for the growth of microservices. Moreover, rapid provisioning, basic monitoring and rapid application deployment are prerequisites for any microservices application [21]. Such requirements are inherently available in the cloud, thus becoming the default home for microservices.

Regardless of the complexities inherent in microservices, a trend on migrating monolithic applications towards microservices architectures has become apparent. Multiple development teams have published their experience migrating to a microservices architecture, including some success stories. However, due to the nature of microservices, following such advices may not be suitable for every strategy. Therefore, publicly available knowledge in this migration trend, such as best practices, success stories, and pitfalls should be collected. The subsequent consolidation of this knowledge in form of migration and architecture smells can provide useful information for teams looking to migrate their applications into microservices.

In this paper, we present 5 new architecture and 4 new migration bad smells found by digesting 58 different sources from the academia and grey literature. The rest of this paper is structured as follows. Section 2 provides an overview of related work. Section 3 presents the research questions, and Section 4 explains our methodology. Section 5 presents the 5 new architecture bad smells, whereas the 4 new migration bad smells are presented in Section 6. Section 7 discusses the threats to validity, and Section 8 concludes.

2 RELATED WORK

Refactoring is part of the Software Engineering Body of Knowledge (SWEBOK). Initially refactoring was intended for restructuring code. However, Stal extended the concept of refactoring to include software architecture refactoring [53]. When refactoring an architecture, the software is changed in a holistic manner for addressing architecture smells.

Smells Harmfullnes?

Possible Solutions

- **Define clear architectural guidelines**
  - No over-engineering
- **Adopt architectural patterns**
  - https://microservice-api-patterns.org*
- **Keep Anti-Patterns and Bad-Smells under control**
- **Identify a whitelist and blacklist of allowed technologies**
- **Define guidelines for adding new services**

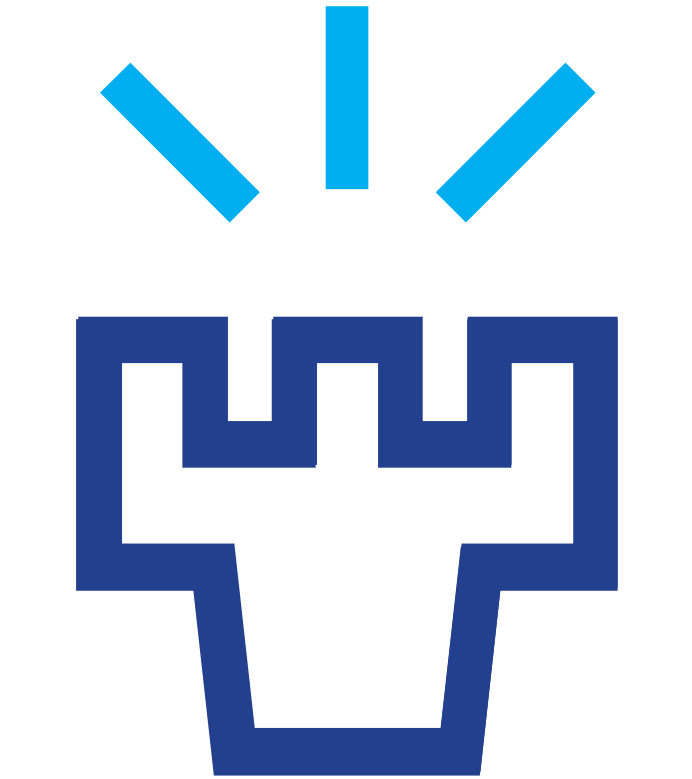
[*] Olaf Zimmermann, Mirko Stocker, Daniel Lübke, Cesare Pautasso, Uwe Zdun. **Microservice API Patterns.** https://microservice-api-patterns.org

Main issues

- **Architectural guidelines need to be updated (continuously)**
- **Lack of tools for detecting architectural patterns and anti-patterns**
- **Very powerful technologies might be tempting**
  - E.g. Service meshes vs API Gateway

Conclusions

- **Microservices are now mainstream**
  - Systems are aging
- **Need to control their evolution**
  - Keep Technical Debt under control
- **Need for tools**
  - Patterns, anti-patterns
  - Architectural guidelines compliancy



OULUN
YLIOPISTO