Discovering the Undercurrents: A Glimpse at Technical Debt in Self-Adaptive Systems

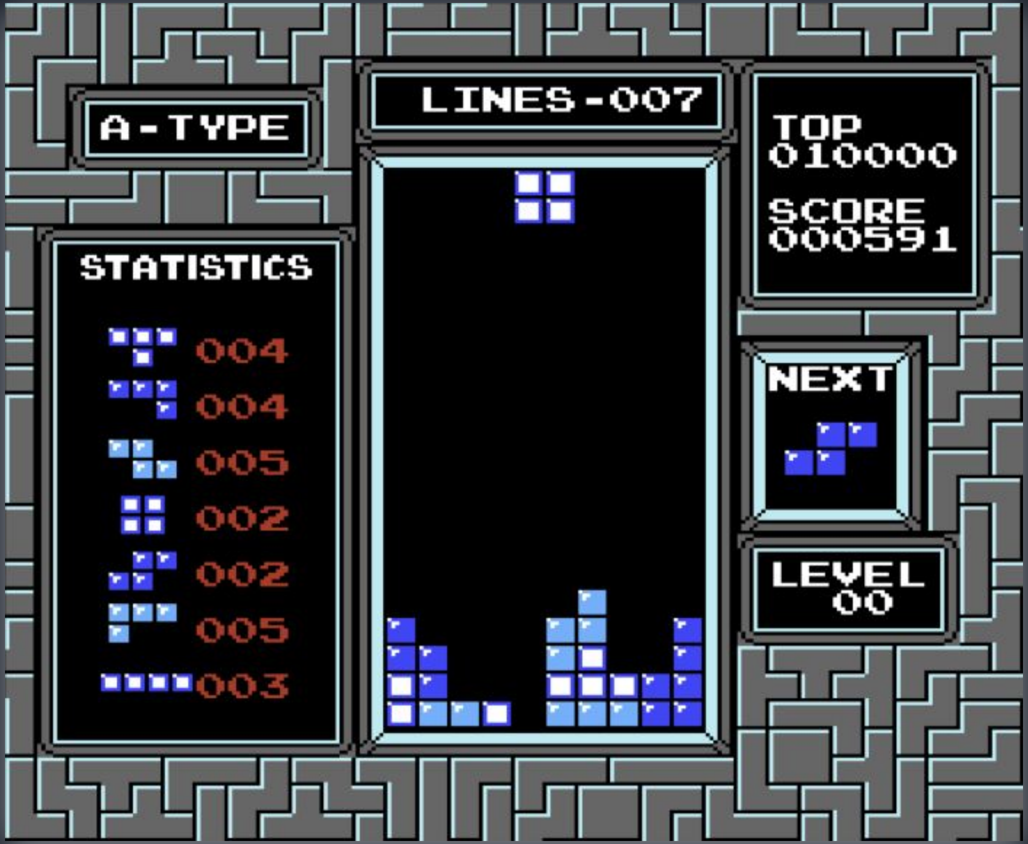
Roberto Verdecchia
University of Florence

{ST LAB} Software Technologies Laboratory
UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

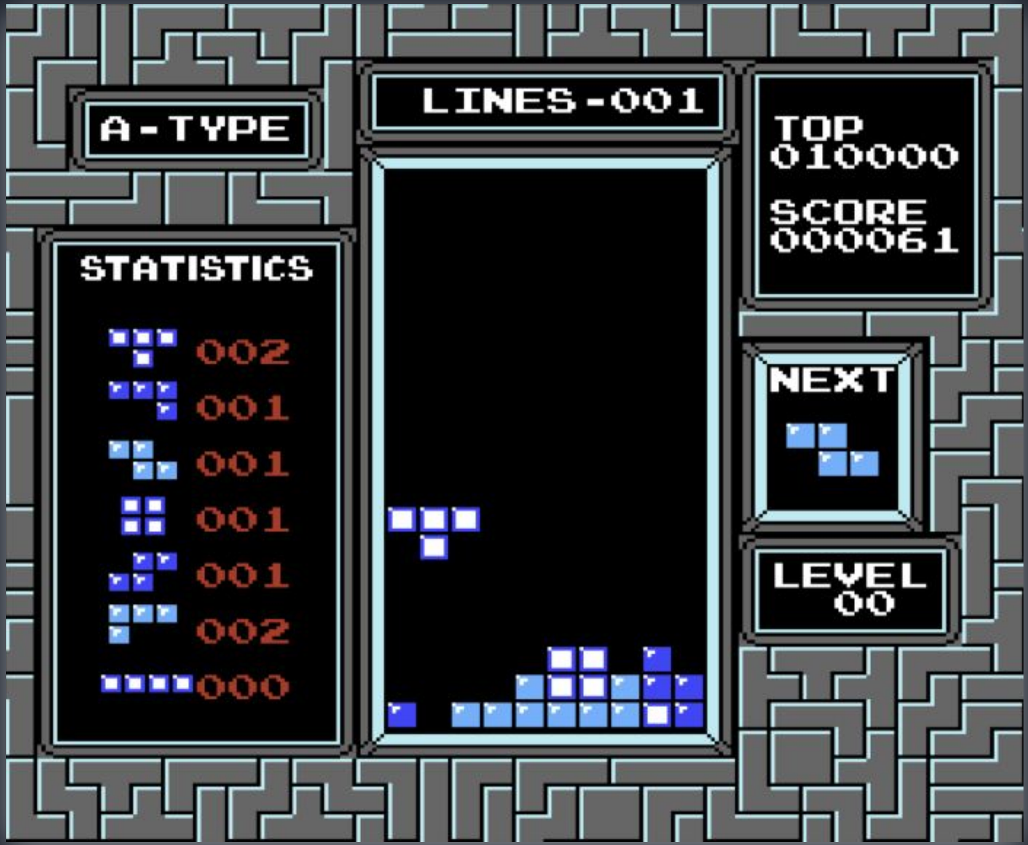
Technical Debt is a game of Tetris™

https://medium.com/s/story/technical-debt-is-like-tetris-168f64d8b700

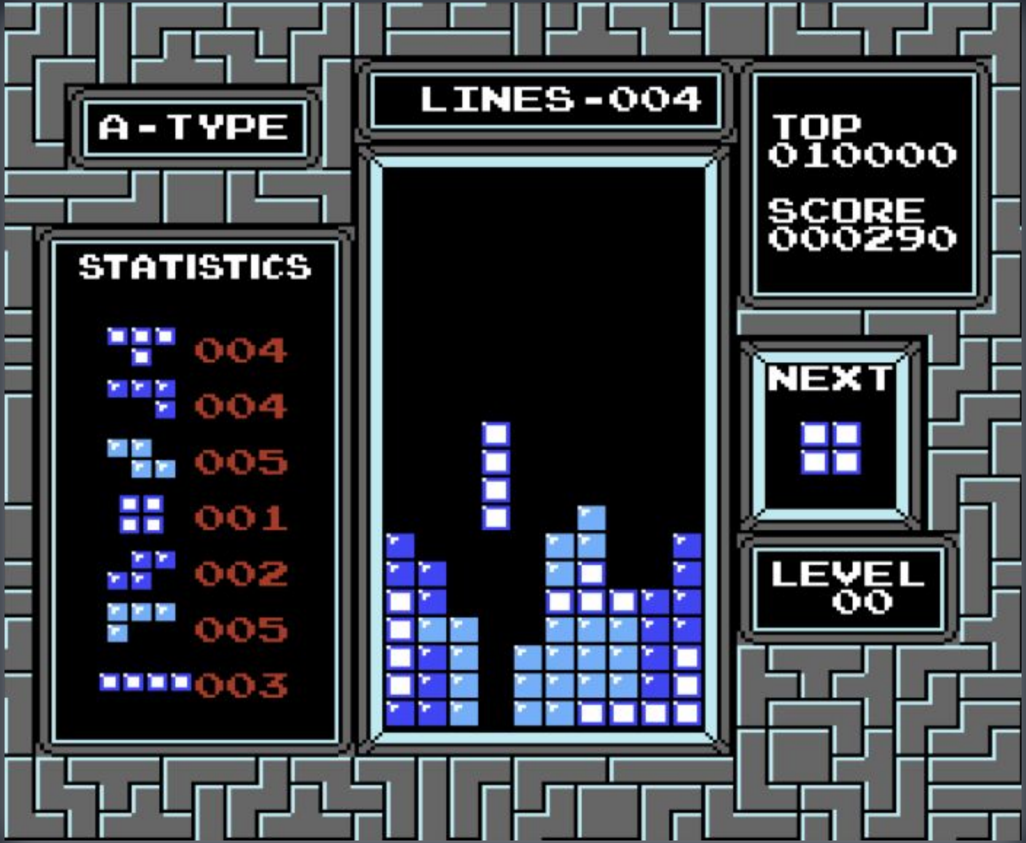
Technical Debt is a game of Tetris™



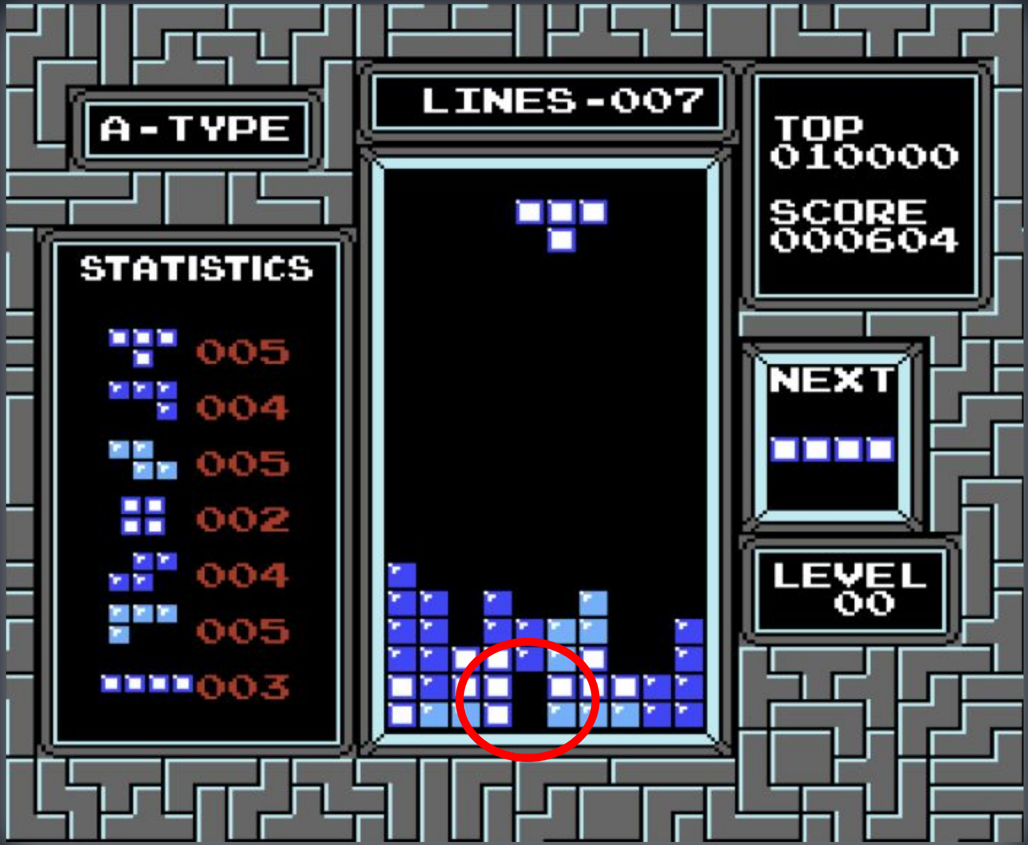
At the start complexity is low...



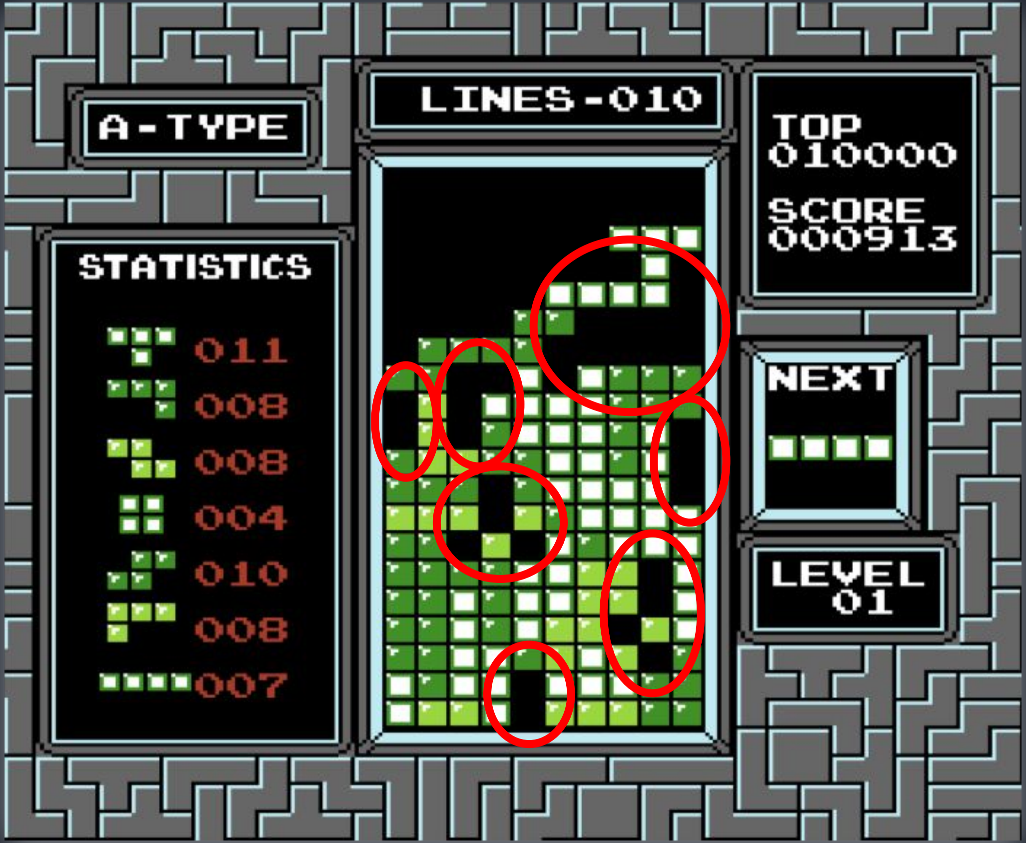
Building on the side might be a sound strategy



Some gaps make the game more difficult, but are acceptable

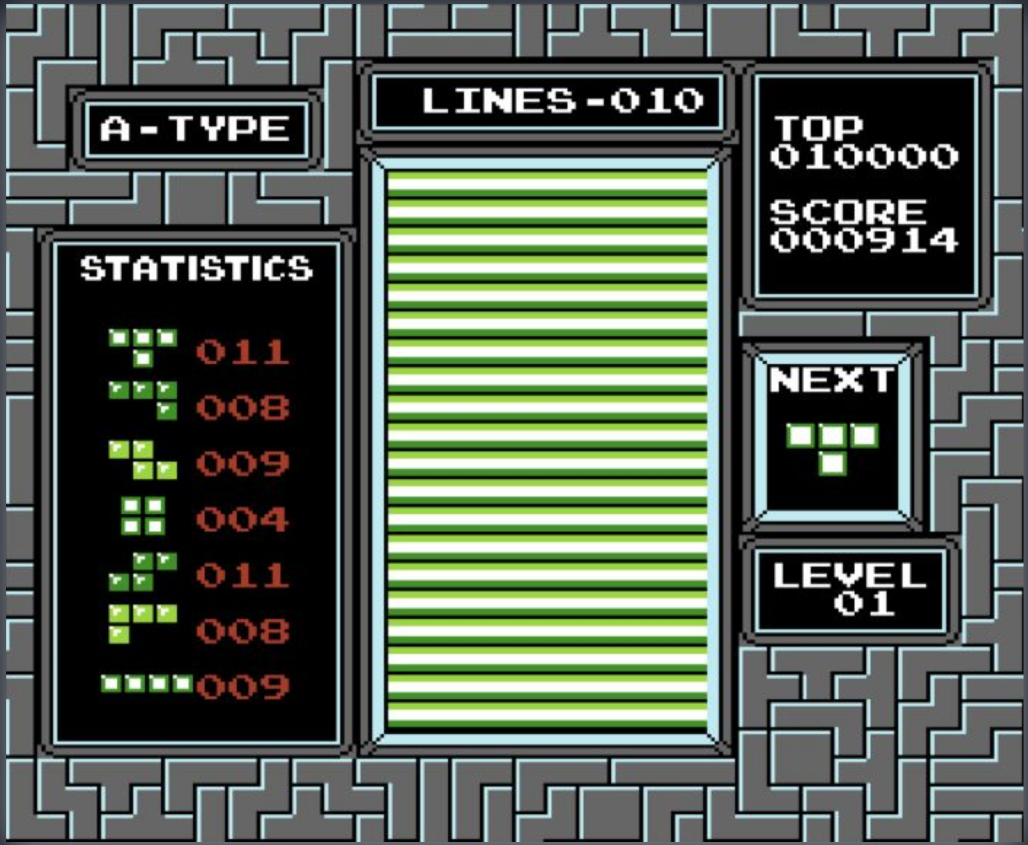


Too many gaps hinder positioning new blocks



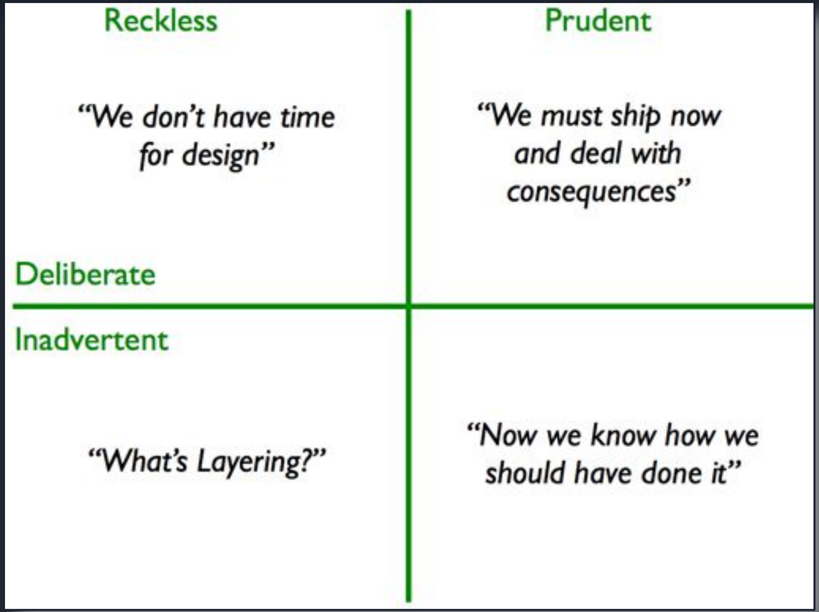
Unable to place new blocks?

Game Over!



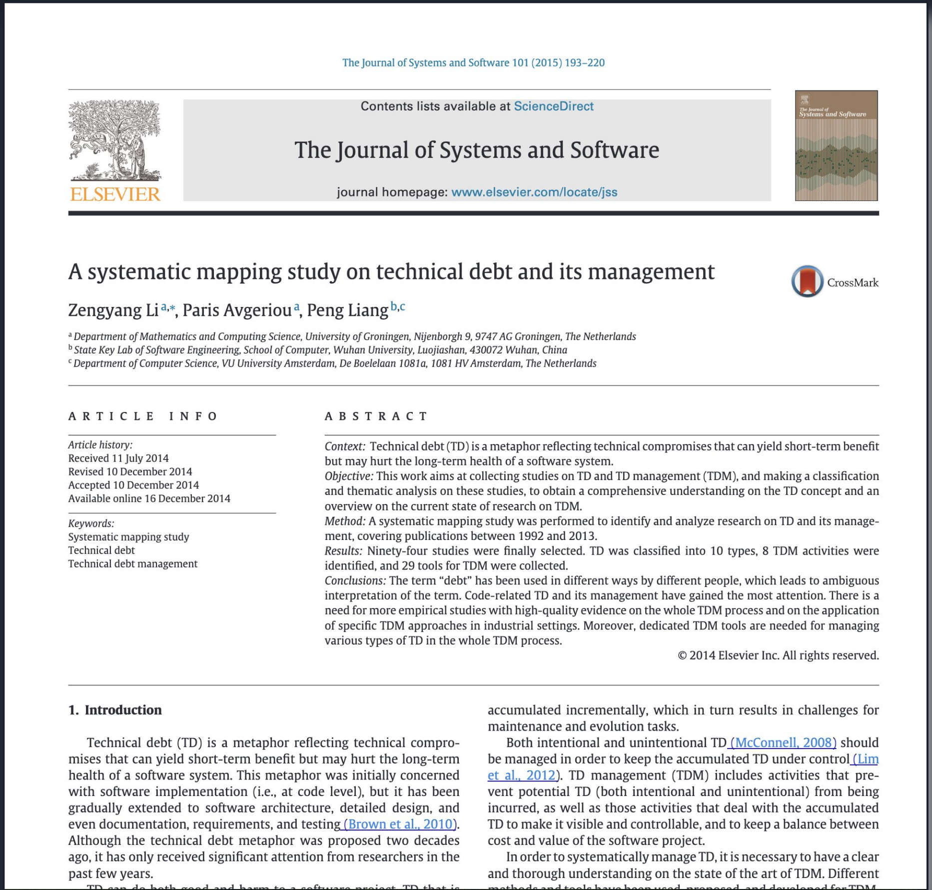
Technical debt core concepts

- **Technical debt principal:** Cost of fixing the short-term expedient
- **Technical debt interest:** Extra effort needed to extra effort needed to maintain the system



https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

Technical debt types



The Journal of Systems and Software 101 (2015) 193–220

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

ELSEVIER

A systematic mapping study on technical debt and its management

CrossMark

Zengyang Li[a,*], Paris Avgeriou[a], Peng Liang[b,c]

[a] Department of Mathematics and Computing Science, University of Groningen, Nijenborgh 9, 9747 AG Groningen, The Netherlands
[b] State Key Lab of Software Engineering, School of Computer, Wuhan University, Luojiashan, 430072 Wuhan, China
[c] Department of Computer Science, VU University Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Received 11 July 2014
Revised 10 December 2014
Accepted 10 December 2014
Available online 16 December 2014

Keywords:
Systematic mapping study
Technical debt
Technical debt management

ABSTRACT

Context: Technical debt (TD) is a metaphor reflecting technical compromises that can yield short-term benefit but may hurt the long-term health of a software system.
Objective: This work aims at collecting studies on TD and TD management (TDM), and making a classification and thematic analysis on these studies, to obtain a comprehensive understanding on the TD concept and an overview on the current state of research on TDM.
Method: A systematic mapping study was performed to identify and analyze research on TD and its management, covering publications between 1992 and 2013.
Results: Ninety-four studies were finally selected. TD was classified into 10 types, 8 TDM activities were identified, and 29 tools for TDM were collected.
Conclusions: The term "debt" has been used in different ways by different people, which leads to ambiguous interpretation of the term. Code-related TD and its management have gained the most attention. There is a need for more empirical studies with high-quality evidence on the whole TDM process and on the application of specific TDM approaches in industrial settings. Moreover, dedicated TDM tools are needed for managing various types of TD in the whole TDM process.

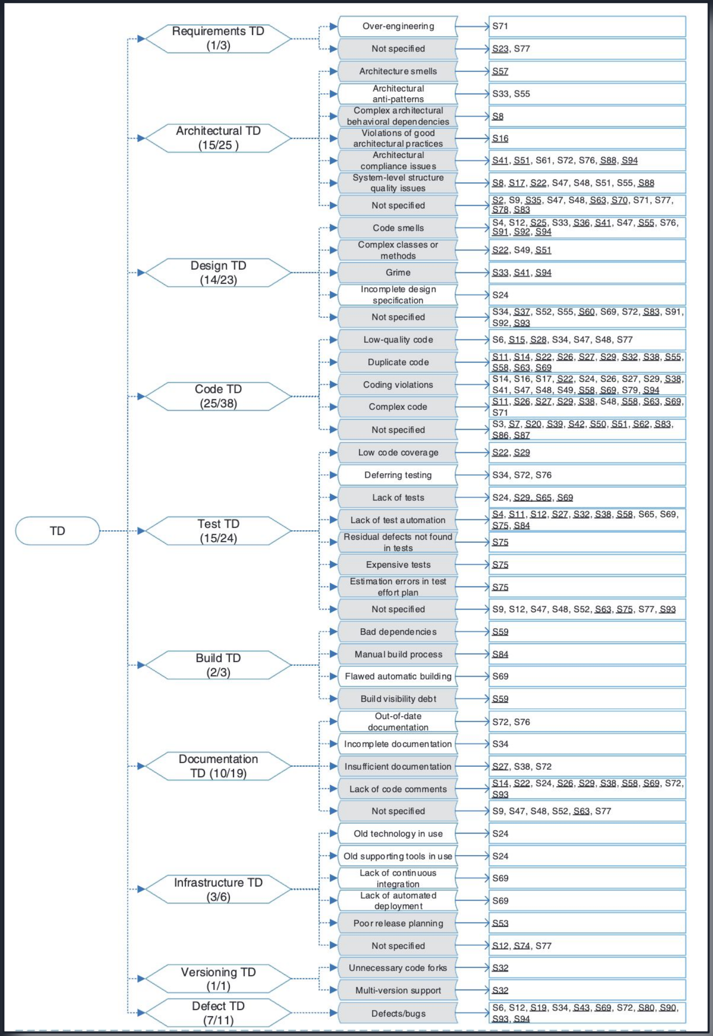
1. Introduction

Technical debt (TD) is a metaphor reflecting technical compromises that can yield short-term benefit but may hurt the long-term health of a software system. This metaphor was initially concerned with software implementation (i.e., at code level), but it has been gradually extended to software architecture, detailed design, and even documentation, requirements, and testing (Brown et al., 2010). Although the technical debt metaphor was proposed two decades ago, it has only received significant attention from researchers in the past few years.

accumulated incrementally, which in turn results in challenges for maintenance and evolution tasks.

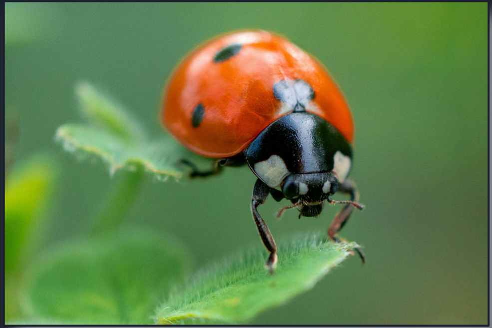
Both intentional and unintentional TD (McConnell, 2008) should be managed in order to keep the accumulated TD under control (Lim et al., 2012). TD management (TDM) includes activities that prevent potential TD (both intentional and unintentional) from being incurred, as well as those activities that deal with the accumulated TD to make it visible and controllable, and to keep a balance between cost and value of the software project.

In order to systematically manage TD, it is necessary to have a clear and thorough understanding on the state of the art of TDM. Different



Technical debt types

Code TD: complex code, code violations, dead code, …



Test TD: low coverage, flaky tests, underperforming tests, …

Architecture TD: architectural smells, technology lock-in, stuck on POF, ...

Requirements TD, e.g., ill-defined requirements, simplistic context definition…

Requirements TD, e.g., unclear requirements, simplistic context definition…

Technical debt analysis approaches: Quantitative analyses

Technical debt analysis approaches: Qualitative analyses

Information and Software Technology 139 (2021) 106669

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Architectural design decisions that incur technical debt — An indu… study

Mohamed Soliman *, Paris Avgeriou *, Yikun Li

Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, Groningen, The Netherlands

ARTICLE INFO

Keywords:
Technical debt
Architectural design decisions
Architectural knowledge
Architectural technical debt

ABSTRACT

Context: During software development, some architectural desi… deliberately or inadvertently. These have serious impact on the q… significant time and effort to be changed. While current research… architectural design decisions and technical debt separately, debt-in… not been specifically explored in practice.
Objective: In this case study, we explore debt-incurring architect… Specifically, we explore the main types of DADDs, why and how th… how practitioners deal with these types of design decisions.
Method: We performed interviews and a focus group with practiti… software companies, discussing their concrete experience with such…
Results: We provide the following contributions: 1) A categorizatio… current ontology on architectural design decisions. 2) A process on h… 3) A conceptual model which shows the relationships between the… 4) The main factors that influence the way of dealing with DADDs…
Conclusion: The results can support the development of new appr… Debt management from the perspective of Design Decisions. Moreov… architecture knowledge related to DADDs.

1. Introduction

Architectural design decisions (ADDs) have the biggest impact on the quality of a software system, and they are hard to change after their implementation [1]. Some ADDs incur technical debt, i.e. they "set up a technical context that can make future changes more costly or impossible" [2]. We call these *Debt-incurring Architectural Design Decisions (DADDs)*, and their impact is well recognized by both practitioners and researchers [3,4]. DADDs can be either deliberate or inadvertent [5].

Deliberate DADDs are taken because of time pressure or lack of resources: a solution is chosen that is quicker and cheaper but compromises maintainability and evolvability. For example, instead of adhering to the layered structure of the architecture, shortcuts are created that bypass layers. This results in implementing the required features quicker, but those shortcuts create ripple effects when making changes.

Inadvertent DADDs, are decisions that, when taken, do not bear any

effort on maintenance, thus be… example of inadvertent DADDs… becomes obsolete after a few ye… an optimal decision in the past,… and unnecessary complexity.

Related research work on ar… empirically explored different ty… lations) [8], their causes, trends [9] and effects [8]. Moreover, methods were proposed to identify ATD (e.g. through capturing architectural bad smells from existing systems) [10]. Nevertheless, current studies have not examined ATD from the perspective of the Architecture Design Decisions (ADDs) that incur it either deliberately or inadvertently. This perspective is of paramount importance to inform the development of approaches to manage ATD, as well as tools to support the decision making process.

In this paper, we aim at exploring the current state of practice in industry regarding DADDs; we determine types of DADDs, we study the

The Journal of Systems & Software 176 (2021) 110925

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Building and evaluating a theory of architectural technical debt in software-intensive systems☆

Roberto Verdecchia [a], Philippe Kruchten [b], Patricia Lago [a,c], Ivano Malavolta [a,*]

[a] Vrije Universiteit Amsterdam, The Netherlands
[b] University of British Columbia, Vancouver, Canada
[c] Chalmers University of Technology, Gothenburg, Sweden

ARTICLE INFO

Article history:
Received 22 December 2020
Accepted 8 February 2021
Available online 27 February 2021

Keywords:
Software engineering
Software architecture
Technical debt
Software evolution
Grounded theory
Focus group

ABSTRACT

Architectural technical debt in software-intensive systems is a metaphor… design decisions (*e.g.*, choices regarding structure, frameworks, technologies,… being suitable or even optimal when made, significantly hinder progress… types of debt, such as code-level technical debt, can be readily detected by… be refactored with minimal or only incremental efforts, architectural debt… wide-ranging remediation cost, daunting, and often avoided.

In this study, we aim at developing a better understanding of how softw… tions conceptualize architectural debt, and how they deal with it. In order to… we apply a mixed empirical method, constituted by a grounded theory study… With the grounded theory method we construct a theory on architectural… qualitative data from software architects and senior technical staff from a w… software development organizations. We applied the focus group method… theory and refine it according to the new data collected.

The result of the study, *i.e.*, a theory emerging from the gathered da… passing conceptual model of architectural technical debt, identifying and… its symptoms, causes, consequences, management strategies, and commun… conducted focus groups, we assessed that the theory adheres to the four e… grounded theory, *i.e.*, the theory *fits* its underlying data, is able to *work*, has… as new data appears.

By grounding the findings in empirical evidence, the theory provide… tioners with novel knowledge on the crucial factors of architectural tech… industrial contexts.

1. Introduction

Technical Debt (TD) is a concept that has been with us for a long time, at least since 1992 when Cunningham crafted the phrase (Cunningham, 1992), but it only got some real attention from researchers in the last 10 years (Brown et al., 2010). What is technical debt? "In software-intensive systems, technical debt consists of design or implementation constructs that are expedient in the short term, but set up a technical context that can make a future change more costly or impossible. Technical debt is a contingent liability whose impact is limited to internal system

qualities, primarily maintainability and… et al., 2016).

Technical debt can take many differe… velopment, and can be found in many di… et al., 2012). While much of the litera… able today address code-level technica… Architectural Technical Debt (ATD). Th… incurred at the *architectural level* of sof… the decisions related to the choice of stru… composition in subsystems, interfaces), th… (*e.g.*, frameworks, packages, libraries, de… even languages, development process, an… systems grow in size and their lifespan… many of these original design choices b… limit future evolution or even prevents… developers do find workarounds and ofte…

☆ Editor: [RAFFAELA MIRANDOLA].
* Corresponding author.
E-mail addresses: r.verdecchia@vu.nl (R. Verdecchia), pbk@ece.ubc.ca

Science of Computer Programming 163 (2018) 42–61

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Technical Debt tracking: Current state of practice
A survey and multiple case study in 15 large organizations

Antonio Martini [a,b,*], Terese Besker [c], Jan Bosch [c]

[a] CA Technologies Strategic, Research Team Barcelona, Spain
[b] University of Oslo, Programming and Software Engineering, Oslo, Norway
[c] Computer Science and Engineering, Chalmers University of Technology, Göteborg, Sweden

ARTICLE INFO

Article history:
Received 5 November 2017
Received in revised form 20 March 2018
Accepted 25 March 2018
Available online 29 March 2018

Keywords:
Technical Debt
Change management
Software process improvement
Survey
Multiple case study

ABSTRACT

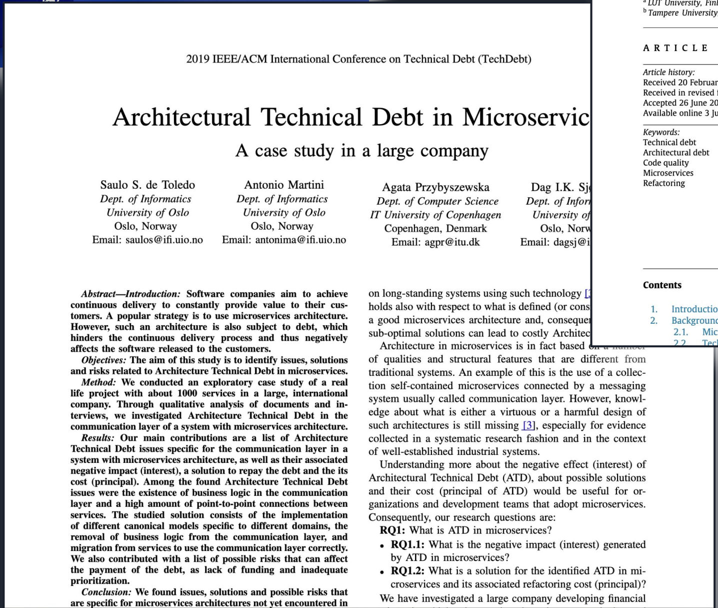
Large software companies need to support continuous and fast delivery of customer value both in the short and long term. However, this can be hindered if both the evolution and maintenance of existing systems are hampered by Technical Debt. Although a lot of theoretical work on Technical Debt has been produced recently, its practical management lacks empirical studies. In this paper, we investigate the state of practice in several companies to understand what the cost of managing TD is, what tools are used to track TD, and how a tracking process is introduced in practice. We combined two phases: a survey involving 226 respondents from 15 organizations and an in-depth multiple case study in three organizations including 13 interviews and 79 Technical Debt issues. We selected the organizations where Technical Debt was better tracked in order to distill best practices. We found that the development time dedicated to managing Technical Debt is substantial (an average of 25% of the overall development), but mostly not systematic: only a few participants (26%) use a tool, and only 7.2% methodically track Technical Debt. We found that the most used and effective tools are currently backlogs and static analyzers. By studying the approaches in the companies participating in the case study, we report how companies start tracking Technical Debt and what the initial benefits and challenges are. Finally, we propose a Strategic Adoption Model for the introduction of tracking Technical Debt in software organizations.

1. Introduction

Large software companies need to support continuous and fast delivery of customer value both in the short and long terms. However, this can be hindered if both the evolution and maintenance of the systems are hampered by Technical Debt.

Technical Debt (TD) has been studied recently in the software engineering literature [1–4]. TD is composed of a *debt*, which is a sub-optimal technical solution that leads to short-term benefits as well as to the future payment of *interest*, which is the extra cost due to the presence of TD (for example, slow feature development or low quality) [5]. The *principal* is regarded as the cost of refactoring TD. Although accumulating Technical Debt might prove useful in some cases, in others, the interest might largely surpass the short-term gain, for example, by causing development crises in the long term [6].

Technical debt is context specific



2019 IEEE/ACM International Conference on Technical Debt (TechDebt)

Architectural Technical Debt in Microservic

A case study in a large company

Saulo S. de Toledo, *Dept. of Informatics*, *University of Oslo*, Oslo, Norway, Email: saulos@ifi.uio.no

Antonio Martini, *Dept. of Informatics*, *University of Oslo*, Oslo, Norway, Email: antonima@ifi.uio.no

Agata Przybyszewska, *Dept. of Computer Science*, *IT University of Copenhagen*, Copenhagen, Denmark, Email: agpr@itu.dk

Dag I.K. Sj

Abstract—Introduction: **Software companies aim to achieve continuous delivery to constantly provide value to their customers. A popular strategy is to use microservices architecture. However, such an architecture is also subject to debt, which hinders the continuous delivery process and thus negatively affects the software released to the customers.**

Objectives: **The aim of this study is to identify issues, solutions and risks related to Architecture Technical Debt in microservices.**

Method: **We conducted an exploratory case study of a real life project with about 1000 services in a large, international company. Through qualitative analysis of documents and interviews, we investigated Architecture Technical Debt in the communication layer of a system with microservices architecture.**

Results: **Our main contributions are a list of Architecture Technical Debt issues specific for the communication layer in a system with microservices architecture, as well as their associated negative impact (interest), a solution to repay the debt and the its cost (principal). Among the found Architecture Technical Debt issues were the existence of business logic in the communication layer and a high amount of point-to-point connections between services. The studied solution consists of the implementation of different canonical models specific to different domains, the removal of business logic from the communication layer, and migration from services to use the communication layer correctly. We also contributed with a list of possible risks that can affect the payment of the debt, as lack of funding and inadequate prioritization.**

Conclusion: **We found issues, solutions and possible risks that are specific for microservices architectures not yet encountered in**

on long-standing systems using such technology [
holds also with respect to what is defined (or cons
a good microservices architecture and, consequen
sub-optimal solutions can lead to costly Architec

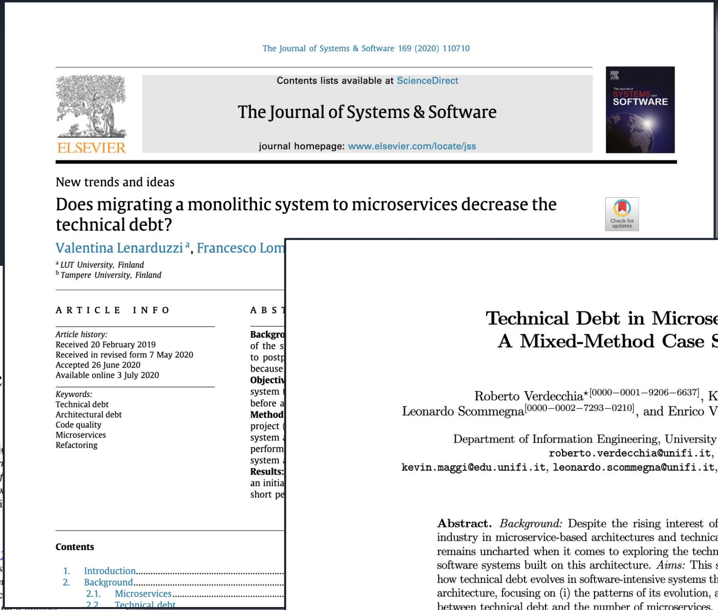
Architecture in microservices is in fact based
of qualities and structural features that are different from traditional systems. An example of this is the use of a collection self-contained microservices connected by a messaging system usually called communication layer. However, knowledge about what is either a virtuous or a harmful design of such architectures is still missing [3], especially for evidence collected in a systematic research fashion and in the context of well-established industrial systems.

Understanding more about the negative effect (interest) of Architectural Technical Debt (ATD), about possible solutions and their cost (principal of ATD) would be useful for organizations and development teams that adopt microservices. Consequently, our research questions are:

RQ1: What is ATD in microservices?

- **RQ1.1:** What is the negative impact (interest) generated by ATD in microservices?
- **RQ1.2:** What is a solution for the identified ATD in microservices and its associated refactoring cost (principal)?

We have investigated a large company developing financial



The Journal of Systems & Software 169 (2020) 110710

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

New trends and ideas

Does migrating a monolithic system to microservices decrease the technical debt?

Valentina Lenarduzzi [a], Francesco Lom

[a] LUT University, Finland

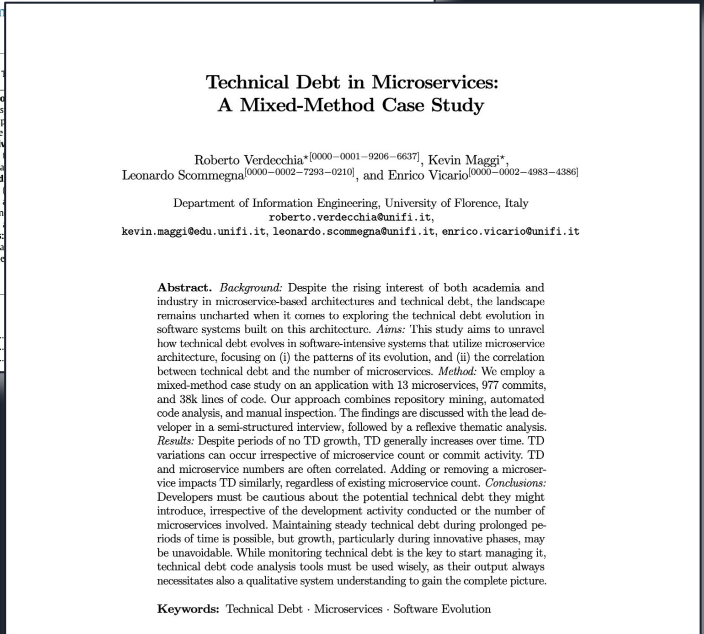
[b] Tampere University, Finland

ARTICLE INFO

Article history:
Received 20 February 2019
Received in revised form 7 May 2020
Accepted 26 June 2020
Available online 3 July 2020

Keywords:
Technical debt
Architectural debt
Code quality
Microservices
Refactoring

Contents



Technical Debt in Microservices: A Mixed-Method Case Study

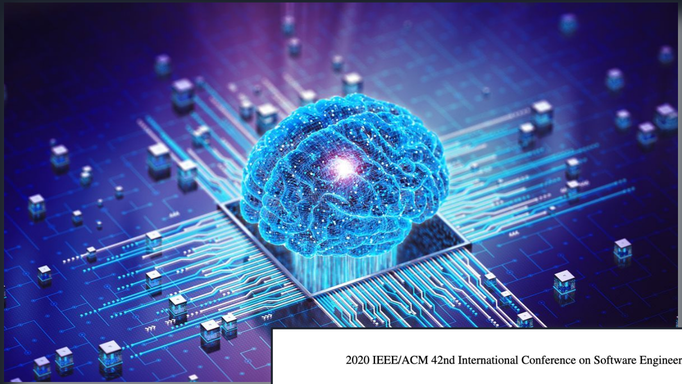
Roberto Verdecchia* [0000-0001-9206-6637], Kevin Maggi*, Leonardo Scommegna [0000-0002-7293-0210], and Enrico Vicario [0000-0002-4983-4386]

Department of Information Engineering, University of Florence, Italy
roberto.verdecchia@unifi.it,
kevin.maggi@edu.unifi.it, leonardo.scommegna@unifi.it, enrico.vicario@unifi.it

Abstract. *Background:* Despite the rising interest of both academia and industry in microservice-based architectures and technical debt, the landscape remains uncharted when it comes to exploring the technical debt evolution in software systems built on this architecture. *Aims:* This study aims to unravel how technical debt evolves in software-intensive systems that utilize microservice architecture, focusing on (i) the patterns of its evolution, and (ii) the correlation between technical debt and the number of microservices. *Method:* We employ a mixed-method case study on an application with 13 microservices, 977 commits, and 38k lines of code. Our approach combines repository mining, automated code analysis, and manual inspection. The findings are discussed with the lead developer in a semi-structured interview, followed by a reflexive thematic analysis. *Results:* Despite periods of no TD growth, TD generally increases over time. TD variations can occur irrespective of microservice count or commit activity. TD and microservice numbers are often correlated. Adding or removing a microservice impacts TD similarly, regardless of existing microservice count. *Conclusions:* Developers must be cautious about the potential technical debt they might introduce, irrespective of the development activity conducted or the number of microservices involved. Maintaining steady technical debt during prolonged periods of time is possible, but growth, particularly during innovative phases, may be unavoidable. While monitoring technical debt is the key to start managing it, technical debt code analysis tools must be used wisely, as their output always necessitates also a qualitative system understanding to gain the complete picture.

Keywords: Technical Debt · Microservices · Software Evolution

Technical debt is context specific



2021 IEEE/ACM International Conference on Technical Debt (TechDebt)

Characterizing Technical Debt and Antipatterns in AI-Based Systems: A Systematic Mapping Study

Justus Bogner*
University of Stuttgart
Institute of Software Engineering
Stuttgart, Germany
justus.bogner@iste.uni-stuttgart.de

Roberto Verdecchia*
Vrije Universiteit Amsterdam
Department of Computer Science
Amsterdam, The Netherlands
r.verdecchia@vu.nl

Ilias Gerostathopoulos*
Vrije Universiteit Amsterdam
Department of Computer Science
Amsterdam, The Netherlands
i.g.gerostathopoulos@vu.nl

nical Debt (TechDebt) | 978-1-6654-1405-0/20/$31.00 ©2021 IEEE | DOI: 10.1109/TechDebt52882.2021.00016

Abstract*—*Background: **With the rising popularity of Intelligence (AI), there is a growing need to build complex AI-based systems in a cost-effective and m way. Like with traditional software, Technical Debt emerge naturally over time in these systems, therefore challenges and risks if not managed appropriately. The of data science and the stochastic nature of AI-base may also lead to new types of TD or antipatterns, whic yet fully understood by researchers and practitioners. The goal of our study is to provide a clear over characterization of the types of TD (both established ones) that appear in AI-based systems, as well as the an and related solutions that have been proposed. *Method:* the process of a systematic mapping study, 21 prima are identified and analyzed. *Results:* Our results sho established TD types, variations of them, and four types (data, model, configuration, and ethics debt) a in AI-based systems, (ii) 72 antipatterns are discuss literature, the majority related to data and model d and (iii) 46 solutions have been proposed, either t specific TD types, antipatterns, or TD in general. *Co* Our results can support AI professionals with reason and communicating aspects of TD present in thei Additionally, they can serve as a foundation for future to further our understanding of TD in AI-based syste**

***Index Terms*—Artificial Intelligence, Machine Learn nical Debt, Antipatterns, Systematic Mapping Study**

2020 IEEE/ACM 42nd International Conference on Software Engineering: Software E

Is Using Deep Learning Framewor Characterizing Technical Debt in Deep Lear

Jiakun Liu*†
Zhejiang University
College of Computer Science and Technology
Hangzhou, Zhejiang, China
jkliu@zju.edu.cn

Qiao Huang
Zhejiang University
College of Computer Science and Technology
Hangzhou, Zhejiang, China
tkdsheep@zju.edu.cn

Emad Shihab
Concordia University
Department of Computer Science and Software Engineering
Montreal, Canada
eshihab@encs.concordia.ca

David Lo
Singapore Management University
School of Information System
Singapore
davidlo@smu.edu.sg

Hangzhou, Zhejiang, China
shan@zju.edu.cn

ABSTRACT

Developers of deep learning applications (shortened as application developers) commonly use deep learning frameworks in their projects. However, due to time pressure, market competition, and cost reduction, developers of deep learning frameworks (shortened as framework developers) often have to sacrifice software quality to satisfy a shorter completion time. This practice leads to technical debt in deep learning frameworks, which results in the increasing burden to both the application developers and the framework developers in future development.

In this paper, we analyze the comments indicating technical debt (self-admitted technical debt) in 7 of the most popular open-source deep learning frameworks. Although framework developers are aware of such technical debt, typically the application developers are not. We find that: 1) there is a significant number of technical debt in all the studied deep learning frameworks. 2) there is design debt, defect debt, documentation debt, test debt, requirement debt,

applications. Based on our findings, we highlight future research directions and provide recommendations for practitioners.

CCS CONCEPTS

• **Software and its engineering → Software evolution**; **Maintaining software**.

KEYWORDS

Self-admitted Technical Debt, Deep Learning, Categorization, Empirical Study

ACM Reference Format:
Jiakun Liu, Qiao Huang, Xin Xia, Emad Shihab, David Lo, and Shanping Li. 2020. Is Using Deep Learning Frameworks Free? Characterizing Technical Debt in Deep Learning Frameworks. In *Software Engineering in Society (ICSE-SEIS'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3377815.3381377

1 INTRODUCTION

The Journal of Systems and Software 216 (2024) 112151

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

ELSEVIER

Technical debt in AI-enabled systems: On the prevalence, severity, impact, and management strategies for code and architecture☆

Gilberto Recupito [a,*], Fabiano Pecorelli [a], Gemma Catolino [a], Valentina Lenarduzzi [b], Davide Taibi [b], Dario Di Nucci [a], Fabio Palomba [a]

[a] *Software Engineering (SeSa) Lab — University of Salerno, Salerno, Italy*
[b] *University of Oulu, Oulu, Finland*

A R T I C L E I N F O

Keywords:
AI technical debt
Software quality
Survey studies
Software engineering for artificial intelligence
Empirical software engineering

A B S T R A C T

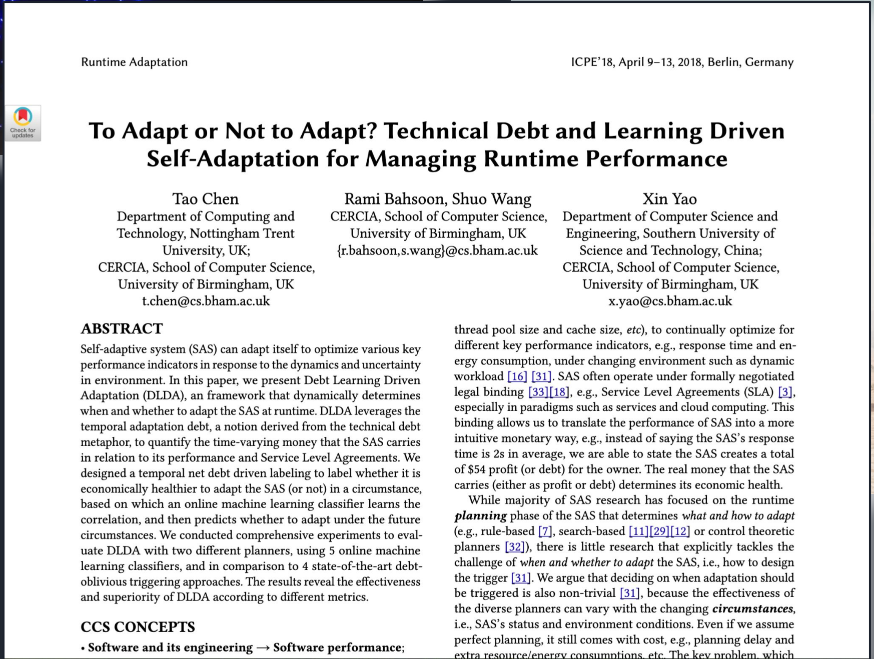
Context: Artificial Intelligence (AI) is pervasive in several application domains and promises to be even more diffused in the next decades. Developing high-quality AI-enabled systems — software systems embedding one or multiple AI components, algorithms, and models — could introduce critical challenges for mitigating specific risks related to the systems' quality. Such development alone is insufficient to fully address socio-technical consequences and the need for rapid adaptation to evolutionary changes. Recent work proposed the concept of AI technical debt, a potential liability concerned with developing AI-enabled systems whose impact can affect the overall systems' quality. While the problem of AI technical debt is rapidly gaining the attention of the software engineering research community, scientific knowledge that contributes to understanding and managing the matter is still limited.

Objective: In this paper, we leverage the expertise of practitioners to offer useful insights to the research community, aiming to enhance researchers' awareness about the detection and mitigation of AI technical debt. Our ultimate goal is to empower practitioners by providing them with tools and methods. Additionally, our study sheds light on novel aspects that practitioners might not be fully acquainted with, contributing to a deeper understanding of the subject.

Method: We develop a survey study featuring 53 AI practitioners, in which we collect information on the practical prevalence, severity, and impact of AI technical debt issues affecting the code and the architecture other than the strategies applied by practitioners to identify and mitigate them.

Results: The key findings of the study reveal the multiple impacts that AI technical debt issues may have on the quality of AI-enabled systems (*e.g.,* the high negative impact that *Undeclared consumers* has on security, whereas *Jumbled Model Architecture* can induce the code to be hard to maintain) and the little support practitioners have to deal with them, limited to apply manual effort for identification and refactoring.

Technical debt is context specific



To Adapt or Not to Adapt? Technical Debt and Learning Driven Self-Adaptation for Managing Runtime Performance

Tao Chen
Department of Computing and Technology, Nottingham Trent University, UK;
CERCIA, School of Computer Science, University of Birmingham, UK
t.chen@cs.bham.ac.uk

Rami Bahsoon, Shuo Wang
CERCIA, School of Computer Science, University of Birmingham, UK
{r.bahsoon,s.wang}@cs.bham.ac.uk

Xin Yao
Department of Computer Science and Engineering, Southern University of Science and Technology, China;
CERCIA, School of Computer Science, University of Birmingham, UK
x.yao@cs.bham.ac.uk

ABSTRACT
Self-adaptive system (SAS) can adapt itself to optimize various key performance indicators in response to the dynamics and uncertainty in environment. In this paper, we present Debt Learning Driven Adaptation (DLDA), an framework that dynamically determines when and whether to adapt the SAS at runtime. DLDA leverages the temporal adaptation debt, a notion derived from the technical debt metaphor, to quantify the time-varying money that the SAS carries in relation to its performance and Service Level Agreements. We designed a temporal net debt driven labeling to label whether it is economically healthier to adapt the SAS (or not) in a circumstance, based on which an online machine learning classifier learns the correlation, and then predicts whether to adapt under the future circumstances. We conducted comprehensive experiments to evaluate DLDA with two different planners, using 5 online machine learning classifiers, and in comparison to 4 state-of-the-art debt-oblivious triggering approaches. The results reveal the effectiveness and superiority of DLDA according to different metrics.

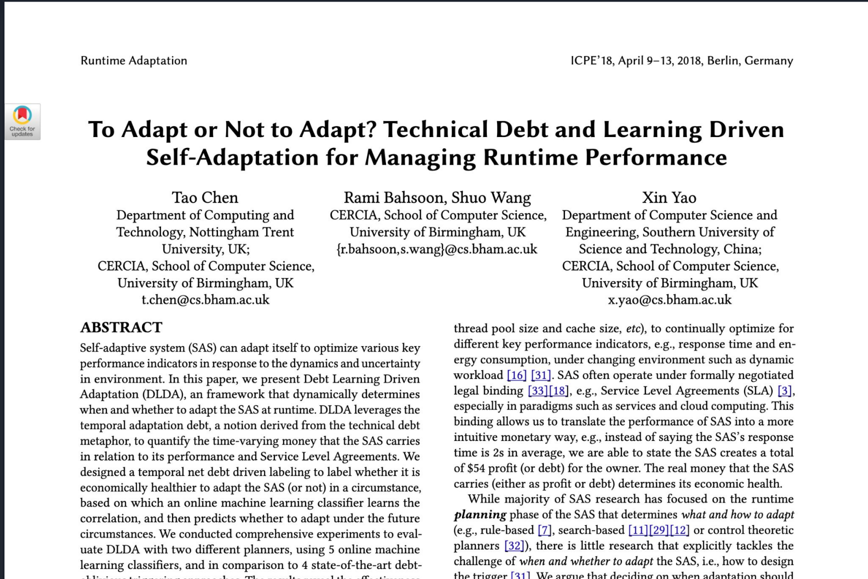
CCS CONCEPTS
• **Software and its engineering** → **Software performance**;

thread pool size and cache size, *etc*), to continually optimize for different key performance indicators, e.g., response time and energy consumption, under changing environment such as dynamic workload [16] [31]. SAS often operate under formally negotiated legal binding [33][18], e.g., Service Level Agreements (SLA) [3], especially in paradigms such as services and cloud computing. This binding allows us to translate the performance of SAS into a more intuitive monetary way, e.g., instead of saying the SAS's response time is 2s in average, we are able to state the SAS creates a total of $54 profit (or debt) for the owner. The real money that the SAS carries (either as profit or debt) determines its economic health.

While majority of SAS research has focused on the runtime ***planning*** phase of the SAS that determines *what and how to adapt* (e.g., rule-based [7], search-based [11][29][12] or control theoretic planners [32]), there is little research that explicitly tackles the challenge of *when and whether to adapt* the SAS, i.e., how to design the trigger [31]. We argue that deciding on when adaptation should be triggered is also non-trivial [31], because the effectiveness of the diverse planners can vary with the changing ***circumstances***, i.e., SAS's status and environment conditions. Even if we assume perfect planning, it still comes with cost, e.g., planning delay and extra resource/energy consumptions, etc. The key problem, which

Technical Debt in SAS: The State of the art

- *When and whether to adapt a Self Adaptive System?*
- **TD modeling**
  - **Principal cost:** Cost of adaptation
  - **Interest:** Penalty due to inability to react to the changing environment
- **Goal:** Adapting the SAS if and only if it can make the SAS economically healthier than not adapting it
- **Approach:**
  - Train a binary classifier
  - Set of temporal debt labels
  - Decide whether it is economically healthier to adapt a SAS or not



To Adapt or Not to Adapt? Technical Debt and Learning Driven Self-Adaptation for Managing Runtime Performance

Tao Chen
Department of Computing and Technology, Nottingham Trent University, UK;
CERCIA, School of Computer Science, University of Birmingham, UK
t.chen@cs.bham.ac.uk

Rami Bahsoon, Shuo Wang
CERCIA, School of Computer Science, University of Birmingham, UK
{r.bahsoon,s.wang}@cs.bham.ac.uk

Xin Yao
Department of Computer Science and Engineering, Southern University of Science and Technology, China;
CERCIA, School of Computer Science, University of Birmingham, UK
x.yao@cs.bham.ac.uk

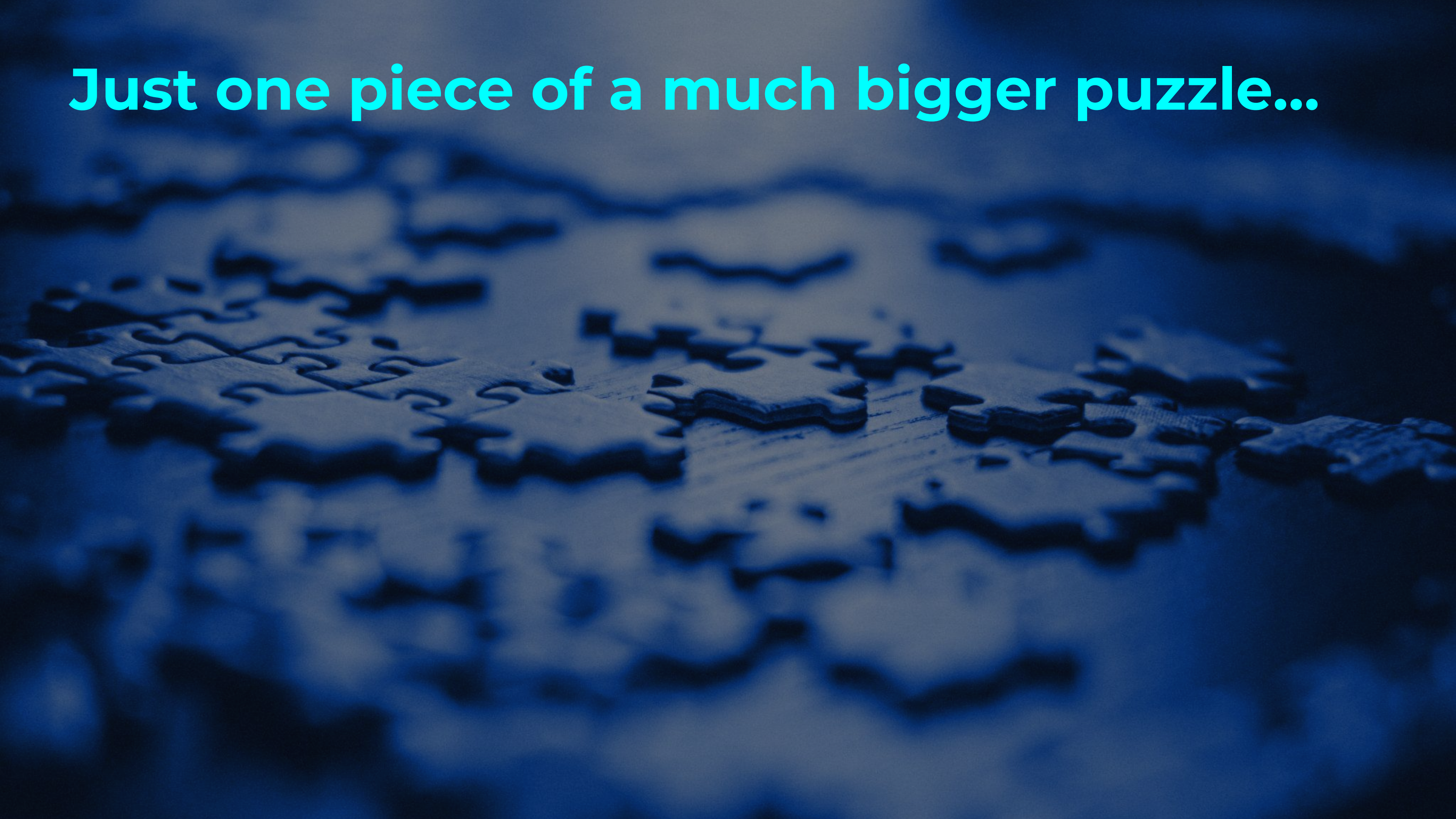
ABSTRACT

Self-adaptive system (SAS) can adapt itself to optimize various key performance indicators in response to the dynamics and uncertainty in environment. In this paper, we present Debt Learning Driven Adaptation (DLDA), an framework that dynamically determines when and whether to adapt the SAS at runtime. DLDA leverages the temporal adaptation debt, a notion derived from the technical debt metaphor, to quantify the time-varying money that the SAS carries in relation to its performance and Service Level Agreements. We designed a temporal net debt driven labeling to label whether it is economically healthier to adapt the SAS (or not) in a circumstance, based on which an online machine learning classifier learns the correlation, and then predicts whether to adapt under the future circumstances. We conducted comprehensive experiments to evaluate DLDA with two different planners, using 5 online machine learning classifiers, and in comparison to 4 state-of-the-art debt-

thread pool size and cache size, *etc*), to continually optimize for different key performance indicators, e.g., response time and energy consumption, under changing environment such as dynamic workload [16] [31]. SAS often operate under formally negotiated legal binding [33][18], e.g., Service Level Agreements (SLA) [3], especially in paradigms such as services and cloud computing. This binding allows us to translate the performance of SAS into a more intuitive monetary way, e.g., instead of saying the SAS's response time is 2s in average, we are able to state the SAS creates a total of $54 profit (or debt) for the owner. The real money that the SAS carries (either as profit or debt) determines its economic health.

While majority of SAS research has focused on the runtime ***planning*** phase of the SAS that determines *what and how to adapt* (e.g., rule-based [7], search-based [11][29][12] or control theoretic planners [32]), there is little research that explicitly tackles the challenge of *when and whether to adapt* the SAS, i.e., how to design

Just one piece of a much bigger puzzle...



Trivial low hanging fruits: “General” TD in SAS

- Gravitating around the generic question: *Do TD types showcase peculiarities in SAS?*
- Code TD:
  - *Are some rule violations more recurrent in SAS?*
  - *Which SAS components are more affected by code TD?*
  - *What are the most recurrent causes of complex code in SAS?*
- Self-admitted TD (SATD) in SAS:
  - *What is the recurrence of SATD items in SAS?*
  - *To what extent are state of the art SATD tools effective in SAS?*

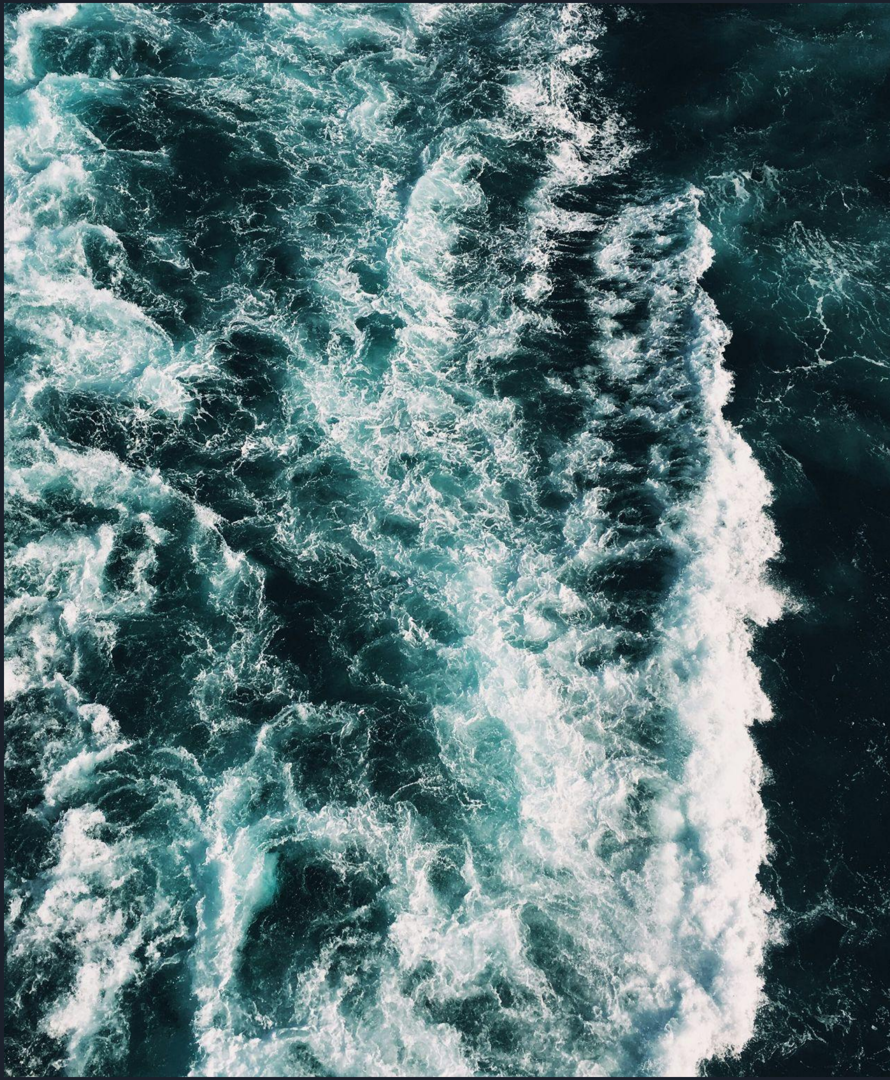
Trivial low hanging fruits: “General” TD in SAS

- Architecture TD:
    - *Which architectural smells are more recurrent is SAS?*
    - *Are there common technology lock-ins in SAS?*

- Infrastructure TD:
    - *Do some recurrent DevOps processes require manual intervention in SAS?*
    - *Are some deprecated technologies still widely used in SAS?*
- ...

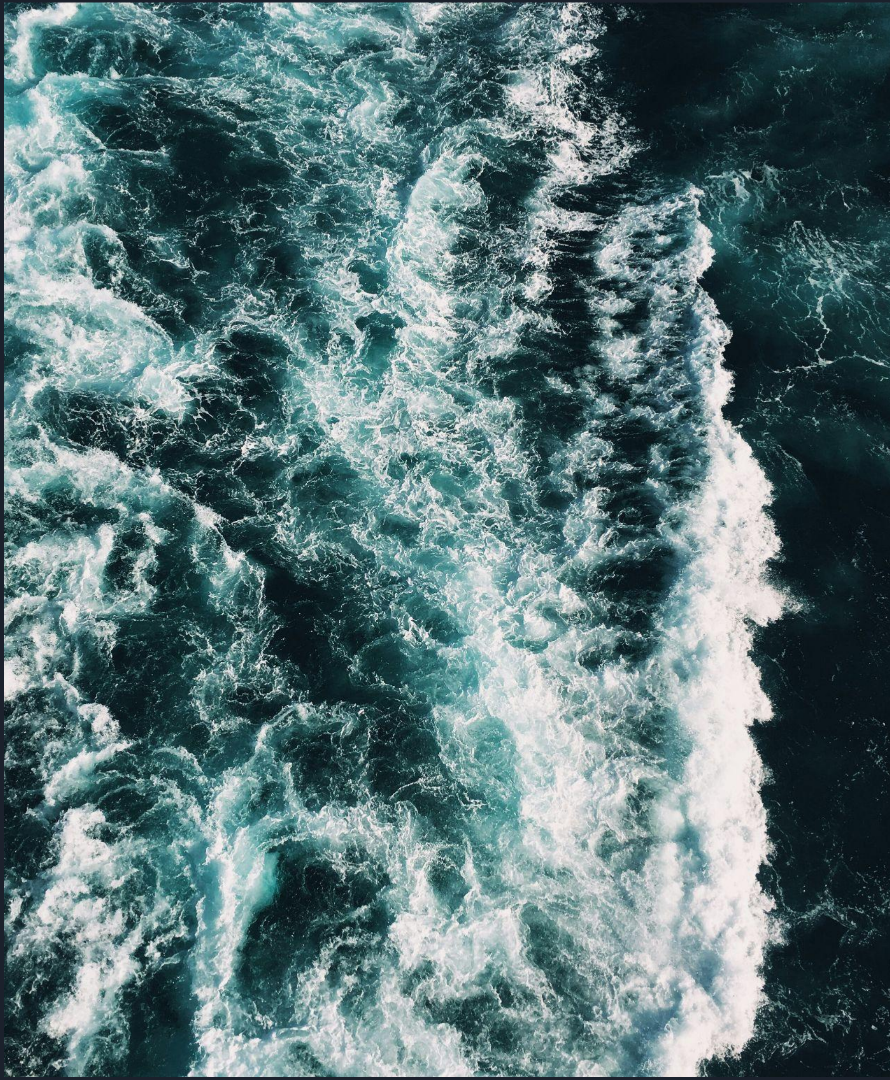
Going beyond: TD *of* SAS

- Understand, identify, monitor, and manage TD *specific to* SAS
- First step: Characterizing TD of SAS
  - **Interest:** Penalty due to inability to react to the changing environment
  - **Principal:** Cost of adaptation… + ?
- Starts from personal knowledge, intuition, and systematic qualitative data collection



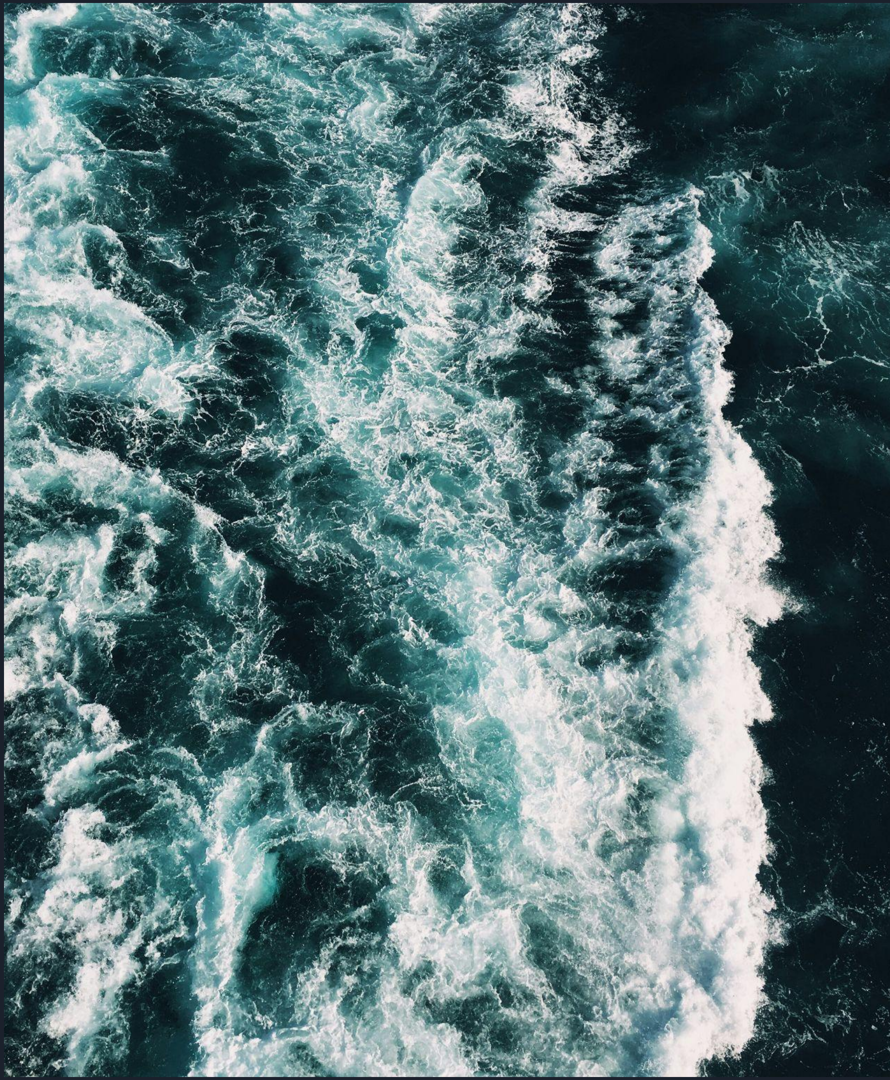
Going beyond: TD *of* SAS

- TD specific to SAS *environment*:
  - Ill-understood or rushed environment definition
  - Coarse environmental condition monitoring
  - Outdated environment modeling
  - ....



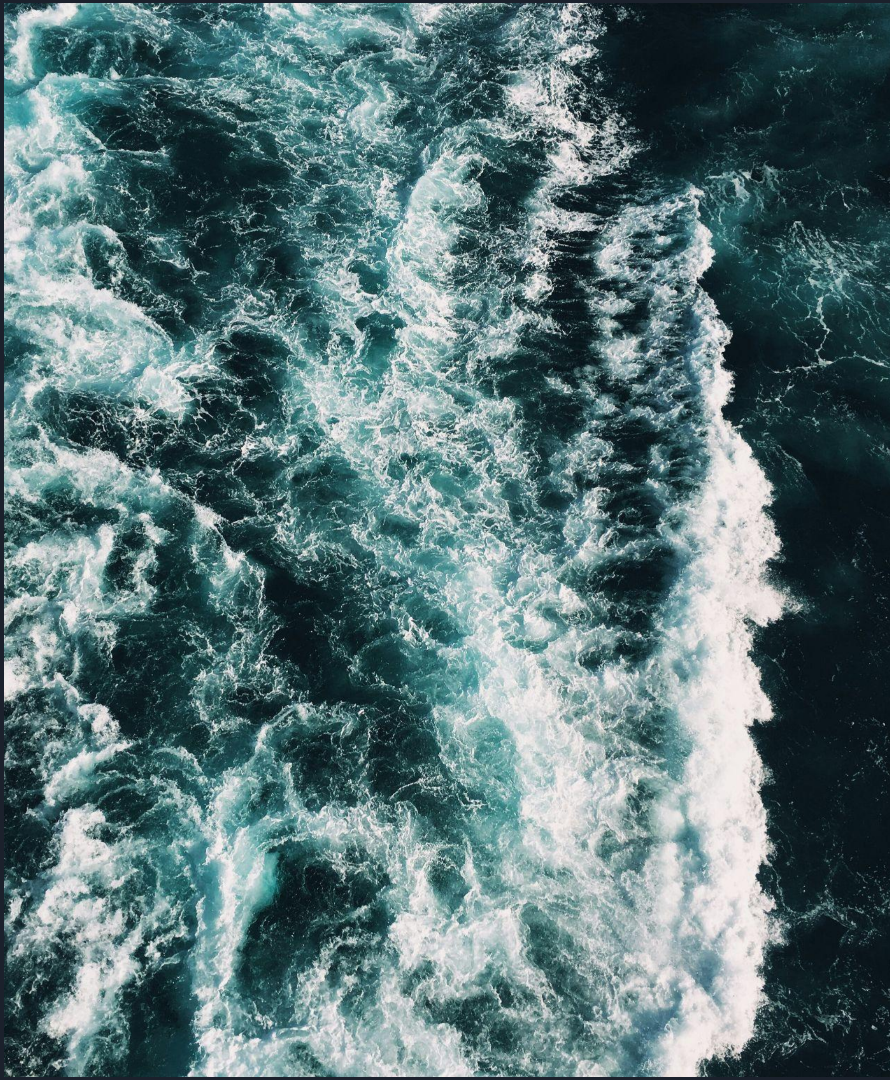
Going beyond: TD *of* SAS

- TD specific to SAS *adaptation*:
  - Rushed adaptation policy
  - Over adapting for the sake of simplicity
  - Choosing ill-suited adaptation strategy (periodic vs event-driven adaptation)
  - Simplistic event prediction
  - ....



Going beyond: TD *of* SAS

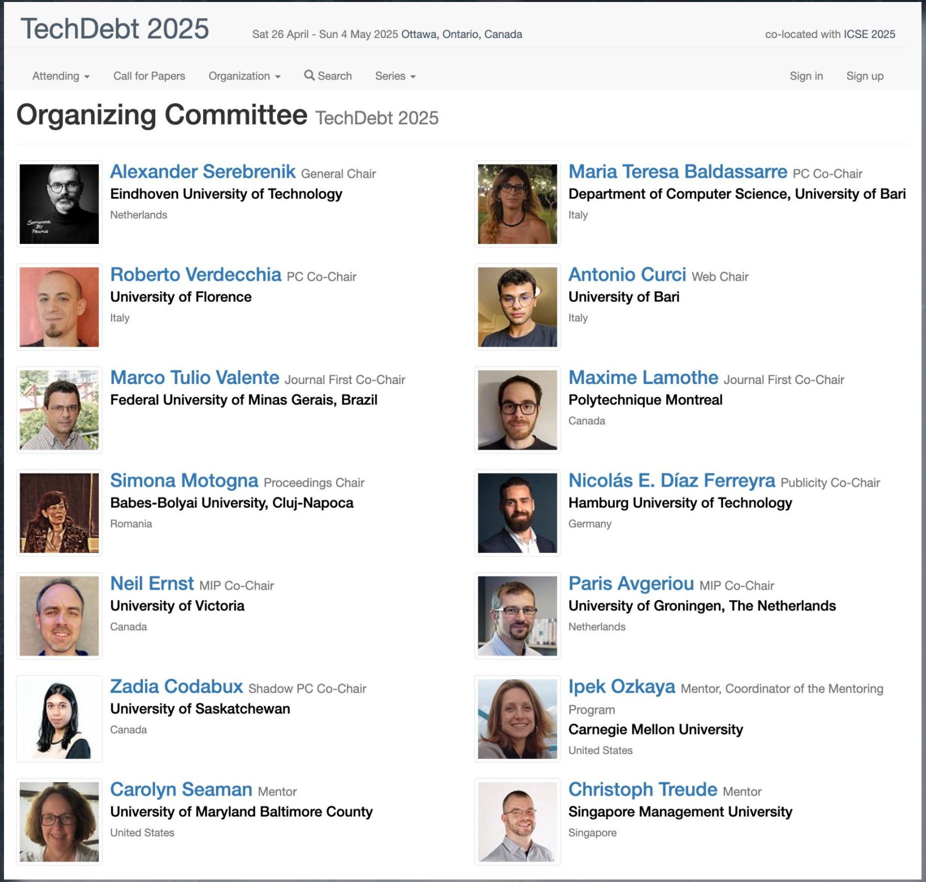
- TD specific to *Service Agreement Levels and KPIs of SAS*:
    - Misaligned KPIs and SLAs
    - Over-optimizing for SLA compliance
    - Overemphasis on easily measurable KPIs
    - Overly granular KPIs
    - Ignoring KPI interdependencies leading to conflicting adaptations
    - Inadequate KPI evolution w.r.t. changing business goals
    - ....



How is technical debt shaping current self-adaptive systems, and how can we manage it?

Consider submitting to TechDebt’25 🙂

- **Flagship conference** on Technical Debt
- **Co-located** with **ICSE**
- **Proceedings paper types:**
  - **Research/experience** (up to 10 pages)
  - **Short** (up to 5 pages)
- Deadline **November 15**


TechDebt 2025 Sat 26 April - Sun 4 May 2025 Ottawa, Ontario, Canada co-located with ICSE 2025

Attending Call for Papers Organization Search Series Sign in Sign up

Organizing Committee TechDebt 2025

Alexander Serebrenik General Chair
Eindhoven University of Technology
Netherlands

Maria Teresa Baldassarre PC Co-Chair
Department of Computer Science, University of Bari
Italy

Roberto Verdecchia PC Co-Chair
University of Florence
Italy

Antonio Curci Web Chair
University of Bari
Italy

Marco Tulio Valente Journal First Co-Chair
Federal University of Minas Gerais, Brazil

Maxime Lamothe Journal First Co-Chair
Polytechnique Montreal
Canada

Simona Motogna Proceedings Chair
Babes-Bolyai University, Cluj-Napoca
Romania

Nicolás E. Díaz Ferreyra Publicity Co-Chair
Hamburg University of Technology
Germany

Neil Ernst MIP Co-Chair
University of Victoria
Canada

Paris Avgeriou MIP Co-Chair
University of Groningen, The Netherlands
Netherlands

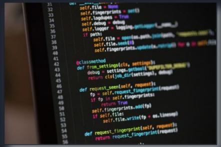
Zadia Codabux Shadow PC Co-Chair
University of Saskatchewan
Canada

Ipek Ozkaya Mentor, Coordinator of the Mentoring Program
Carnegie Mellon University
United States

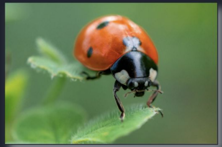
Carolyn Seaman Mentor
University of Maryland Baltimore County
United States

Christoph Treude Mentor
Singapore Management University
Singapore

Technical debt types



Code TD: complex code, code violations, dead code, …



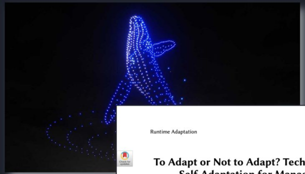
Test TD: low coverage, flaky tests, underperformance tests, …

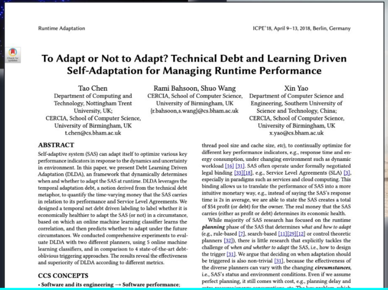
Architecture TD: architectural smells, technology lock-in, stuck on POF, ...

Requirements TD, e.g., ill-defined requirements, simplistic context definition…

Requirements TD, e.g., ill-defined requirements, simplistic context definition…

Technical debt is context specific





Runtime Adaptation — ICPE'18, April 9–13, 2018, Berlin, Germany

To Adapt or Not to Adapt? Technical Debt and Learning Driven Self-Adaptation for Managing Runtime Performance

Tao Chen
Department of Computing and Technology, Nottingham Trent University, UK;
CERCIA, School of Computer Science, University of Birmingham, UK
t.chen@cs.bham.ac.uk

Rami Bahsoon, Shuo Wang
CERCIA, School of Computer Science, University of Birmingham, UK
{r.bahsoon,s.wang}@cs.bham.ac.uk

Xin Yao
Department of Computer Science and Engineering, Southern University of Science and Technology, China;
CERCIA, School of Computer Science, University of Birmingham, UK
x.yao@cs.bham.ac.uk

ABSTRACT

Self-adaptive system (SAS) can adapt itself to optimize various key performance indicators in response to the dynamics and uncertainty in environment. In this paper, we present Debt Learning Driven Adaptation (DLDA), an framework that dynamically determines when and whether to adapt the SAS at runtime. DLDA leverages the temporal adaptation debt, a notion derived from the technical debt metaphor, to quantify the time-varying money that the SAS carries in relation to its performance and Service Level Agreements. We designed a temporal net debt driven labeling to label whether it is economically healthier to adapt the SAS (or not) in a circumstance, based on which an online machine learning classifier learns the correlation, and then predicts whether to adapt under the future circumstances. We conducted comprehensive experiments to evaluate DLDA with two different planners, using 5 online machine learning classifiers, and in comparison to 4 state-of-the-art debt-oblivious triggering approaches. The results reveal the effectiveness and superiority of DLDA according to different metrics.

CCS CONCEPTS

• **Software and its engineering → Software performance**;

thread pool size and cache size, *etc*), to continually optimize for different key performance indicators, e.g., response time and energy consumption, under changing environment such as dynamic workload [16] [31]. SAS often operate under formally negotiated legal binding [33][18], e.g., Service Level Agreements (SLA) [3], especially in paradigms such as services and cloud computing. This binding allows us to translate the performance of SAS into a more intuitive monetary way, e.g., instead of saying the SAS's response time is 2s in average, we are able to state the SAS creates a total of $54 profit (or debt) for the owner. The real money that the SAS carries (either as profit or debt) determines its economic health.

While majority of SAS research has focused on the runtime ***planning*** phase of the SAS that determines *what and how to adapt* (e.g., rule-based [7], search-based [11][29][12] or control theoretic planners [32]), there is little research that explicitly tackles the challenge of *when and whether to adapt* the SAS, i.e., how to design the trigger [31]. We argue that deciding on when adaptation should be triggered is also non-trivial [31], because the effectiveness of the diverse planners can vary with the changing ***circumstances***, i.e., SAS's status and environment conditions. Even if we assume perfect planning, it still comes with cost, e.g., planning delay and

Trivial low hanging fruits: "General" TD in SAS

- Based on the generic question:
  Do TD types showcase peculiarities in SAS?
- Code TD:
  - *Are some rule violations more recurrent in SAS?*
  - *Which SAS components are more affected by code TD?*
  - *What are the most recurrent causes of complex code in SAS?*
- Self-admitted TD (SATD) in SAS:
  - *What is the recurrence of SATD items in SAS?*
  - *To what extent are state of the art SATD tools effective in SAS?*

Going beyond: TD *of* SAS

- TD specific to *Service Agreement Levels and KPIs of SAS:*
  - Misaligned KPIs and SLAs
  - Over-optimizing for SLA Compliance
  - Overemphasis on easily measurable KPIs
  - Overly Granular KPIs
  - Ignoring KPI Interdependencies leading to conflicting adaptations
  - Inadequate KPI evolution w.r.t. changing business goals
  - ….