Technical Debt: A Research Roadmap

Report on the Eighth Workshop on Managing Technical Debt (MTD 2016)

Clemente Izurieta
Montana State University
Bozeman, MT, US
clemente.izurieta@montana.edu

Ipek Ozkaya
Software Engineering Institute
Pittsburgh, PA, US
ozkaya@sei.cmu.edu

Carolyn Seaman
University of Maryland, Baltimore County
Baltimore, MD, US
cseaman@umbc.edu

Will Snipes
ABB Corporate Research
Raleigh, NC, US
will.snipes@us.abb.com

DOI: 10.1145/3041765.3041774
http://doi.acm.org/10.1145/3041765.3041774

ABSTRACT

We report here on the Eighth International Workshop on Managing Technical Debt, collocated with the International Conference on Software Maintenance and Evolution (ICSME 2016). The technical debt research community continues to expand through collaborations of industry, tool vendors, and academia. The major themes of discussion this year indicate convergence on a common definition on technical debt and its elements which drive the maturation of a research roadmap, demonstrating that managing technical debt is a mainstream topic in software engineering research bringing empirical analysis, data science, software design and architecture analysis and automation among other challenges together.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *restructuring.* Metrics – *complexity measures, product metrics, process metrics.* Software Architectures.

General Terms

Management, Measurement, Design, Economics, Standardization.

Keywords

Technical debt, software economics, software quality, software evolution, software analytics.

1. INTRODUCTION

Researchers have met regularly since 2010, in the workshop series on Managing Technical Debt (MTD), to further study and better define the concept and its applicability to software development. In addition, research focusing on technical debt has started to be part of the work discussed in the main tracks of major software engineering conferences. In 2016 there was also a Dagstuhl seminar [1] held focusing on advancing research on managing technical debt. This eighth workshop therefore had the advantage of building on the outcomes of the Dagstuhl seminar, in addition to the recent research progress in the field. Previous workshops had made progress by creating an initial landscape for scoping technical debt [2], establishing key principles of what constitutes technical debt and what is not [3], establishing how different lines of research in software engineering form the basis of technical debt research and starting the beginnings of a research roadmap [4].

This eighth edition; therefore, shared the results of the Dagstuhl seminar and how different lines of software engineering research can contribute to forward progress.

2. CONTRIBUTIONS

The workshop began with a reflection on how far technical debt research has come through these workshops, and a description of the major milestones we have achieved as a community. Our timeline demonstrates the success of MTD research and the growing series of events surrounding this research area. Our introduction and welcome was followed by a keynote given by Firas Glaiel, a member of the software engineering technical staff at Raytheon Technologies. He presented "50 Years of Technical Debt with Rising Interest Rates," an exposé of the challenges faced by Raytheon around detecting and measuring technical debt. Raytheon's main customer is the Department of Defense, thus their experience in building small and large systems that have seen lifespans of 30 to 40 years, is compelling in helping us understand how technical debt, and the decisions that led to it (many of which had not been documented) have accumulated over the years.

The workshop continued with two paper presentation sessions from authors of accepted papers:

- Wolfgang Trumler and Frances Paulisch. *How "Specification by Example" and Test-driven Development Help to Avoid Technical Debt* [5]
- Areti Ampatzoglou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Paris Avgeriou, Pekka Abrahamsson, Antonio Martini, Uwe Zdun and Kari Systa. *"The Perception of Technical Debt in the Embedded Systems Domain: An Industrial Case Study"* [6]
- Mashel Albarak and Rami Bahsoon. *"Database Design Debts through Examining Schema Evolution"* [7]
- Derek Reimanis and Clemente Izurieta. *"Towards Assessing the Technical Debt of Undesired Software Behaviors in Design Patterns"* [8]
- Francesca Arcelli Fontana, Riccardo Roveda and Marco Zanoni. *"Technical Debt Indexes provided by tools: a preliminary discussion"* [9]
- Andriy Shapochka and Borys Omelayenko. *"Practical Technical Debt Discovery by Matching Patterns in Assessment Graph"* [10]
- Shirin Akbarinasaji and Ayse Bener. *"Adjusting the Balance Sheet by Appending Technical Debt"* [11]

After the paper presentation sessions, the attendees spent the last portion of the workshop focusing on i) the technical debt definition, ii) a discussion of a roadmap (from practice to research), and iii) steps to move forward with a research agenda. This session was led by Dr. Carolyn Seaman of the University of Maryland Baltimore County.

3. DISCUSSION

The MTD workshop series has traditionally reserved the last portion of the meeting to lively discussions that engage all attendees. During the first part of this session we discussed the new 16K definition of technical debt [4]:

"In software-intensive systems, technical debt is a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability."

To address the discussion of a roadmap and ways to move forward with a research agenda, Dr. Seaman led an exercise using *sticky notes* to capture attendee's concerns and ideas. The goal was to focus on three themes that helped with continuity from prior discussions from the Dagstuhl meeting. Specifically, these three themes are related to three broad research activity areas originally stated in [12] as:

1. The Core: Defining, understanding, and operationalizing the concept of value with respect to technical debt,
2. The Essential Context: Understanding phenomena that fall outside the core definition of technical debt, and that have an essential relationship with how technical debt plays out in practice, and
3. The Necessary Infrastructure: Building the shared infrastructure that facilitates all our research activities.

3.1 The Core

The specific aspects of this theme include the investigation of opportunity cost -- a term used to describe aspects of value not captured by technical debt principal and interest, for example, the availability of time sensitive resources, and understanding how to model technical debt over time. The group's sticky notes were classified into four distinct groups. The major concerns in each category include:

3.1.1 Technical Debt Vocabulary

- A consistent vocabulary reference clarifying technical debt and its related concepts is needed.
- The definition of "time" in software is not linear and is relevant to understanding technical debt. Can we find an approach to model it?
- The distinction between potential debts versus actual debt needs to be clarified. This becomes especially relevant at specific decision points in the software development lifecycle.
- Given the current Dagstuhl definition of technical debt, should technical debt only include code, design, and architecture debt?

3.1.2 Tools

- Provide developers with a tool or method to rate the difficulty of incorporating new functionality. The purpose of this functionality is to aid in finding hot spots and to help provide an indication where changes become harder to effect in some parts of the code base.
- Provide tools that hint developers and designers to technical debt on the architectural level
- Provide an analysis framework to help identify and decide on opportunities for taking on new debts
- How can we value good practices like automated testing in order to reduce technical debt? Can we provide this feedback in the form of a new tool?
- Can technical debt tell us to declare bankruptcy and start over?

3.1.3 Metrics

- The interdependence of technical debt indicators, such as issues, flaws, and events do have structural dependencies (correlational or causal). These dependencies need to be modeled to avoid biases in estimations. Reference common scores are needed to assess relevance of metrics.
- Does technical debt matter if project the life span is <1 year, <5 years? Technical debt measurement needs to account for relative age of a product
- Understanding what practitioners find useful unit of measure can help, such as surveying practitioners in the field to identify the most useful units of measure for technical debt; monetary, effort, etc.
- What are the units of measurement for value, cost, and interest?
- Can version control information be used to quantify the severity of technical debt?
- How much does it cost to track technical debt explicitly?
- Can machine-learning approaches inform actual costs estimates of potential costs?
- Is there a liability beyond principal and interest?
- Can we use large industry code bases to inform relative costs of maintaining high vs. low debt code?
- How can understanding intentional technical debt help assess value and establish software economics practices related to technical debt.

3.1.4 Dissemination

- We need to effectively communicate technical debt core concepts and values to both developers and managers
- We need to identify categories that characterize companies, perhaps by products they sell, that describe how each technical debt item measurement is considered and make this information available for its easy adoption
- We need to collect a list of the decision factors used in practice for deciding if/when to pay off technical debt

3.2 The Essential Context

The specific aspects of this theme include the identification of important contextual factors that may affect technical debt (i.e., business context) and the understanding of other peripheral definitions of artifacts that could also be described as other types of debt, for example, social and organizational aspects or impact of infrastructure and tooling. The sticky notes contributed by attendees helped with the emergence of four groups:

3.2.1 Contextual Metrics

- Do the same metrics apply for different systems sizes?
- How can existing risk management practices help technical debt management, in particular when appraising technical debt associated risk?
- Technical debt measurements need to incorporate uncertainty and error factors as a part of the final technical debt calculation.
- Metrics should use relative or percentage values rather than absolute values to enable comparisons guidance to a team.
- We need to develop methods for continual measurement of technical debt tracking. How does debt change with each build and/or commit?
- When software systems contain artifacts of different levels of abstractions (architecture, design models, code) we need to investigate how the technical debt measured at each level relates to the other levels
- What is the relationship between technical debt and external components when so much of code and architecture depends on 3rd party (COTS/OSS) components? Can there be a

common understanding how to value technical debt for a COTS/OSS component?
- From which contextual artifacts should benchmarks be extracted?

3.2.2 *Contextual Process*
- How do particular software engineering practices influence technical debt management, for example, automation, model-driven development, traceability of requirements to code, Devops, and test-driven development?
- What kind of development methodologies help developers address support or removal of technical debt?
- How do development methods (e.g., XP, Kanban, Scrum), team structure and people aspects impact the level of debt incurred or its management?
- Is it possible to identify a relationship between social debt and technical debt? If so, how strong is the relationship and does it hold up in other organizations?

3.2.3 *Analysis*
- We need to identify and catalogue the most common causes of technical debt
- Can machine learning assist with the discovery of context?
- Can technical debt context be discovered from existing development artifacts, issues trackers, architecture, documentation and others?
- What changes when technical debt assessment is goal-driven, for example, technical debt with respect to security, or performance?
- How can data mining of information from configuration management systems and other tools that are already used today help with technical debt analysis?
- What is the impact of technical debt in situations where code is expensive to update, for example, embedded, or regulatory?
- How can we evaluate potential debt in architecture via system/UML model analysis?

3.2.4 *Environment*
- Does the type of industry, for example, real time, embedded, finance, and/or gaming, impact technical debt differently?
- How can social and organizational (social debt) aspects be used as a way to predict technical debt or help reduce it?
- We need to understand how aging COTS/FOSS components affect technical debt.

3.3 The Necessary Infrastructure
The aspects of the third theme include activities that will help the community to share experimental data sets and study designs, the creation of benchmarks, and the development of techniques that can help with evaluation and prediction of technical debt, for example, injection techniques of synthetic debt that should yield predictable results. This is necessary to help characterize forms of debt, its cost, and to help calibrate tools. Two groups emerged from the organization of the sticky notes that point to the following activities:

3.3.1 *Technical Debt Sources*
- We need to leverage multiple sources of information (e.g. source code, bug reports, commits, user reports) to get a better assessment of technical debt --code, models, automated testing, commit history, tracking/bug, and analysis of results.
- A need to collect specific architecture examples from OSS.
- We need to establish a number of exemplary systems for which we have some validated information about technical debt and representative artifacts that characterize technical debt in different but consistent forms.
- We need to provide curated examples of technical debt that was avoided, explaining what was done right, and how technical debt is analyzed
- We need to collect and make available case studies beyond OSS to build on operational and useful technical debt models
- We need to provide guidelines that help define what a minimal data set (of artifacts) for an organization to assess its technical debt on a project looks like

3.3.2 *Platforms and Infrastructure*
- A need to establish a starting point benchmark consensus for thresholds of metrics
- A need to investigate tools and techniques to assign risk to technical debt
- We need to ensure that published research includes detailed methodology, data and guidance so that studies can be replicated
- A need to establish an open repository for datasets and/or establish guidance for how existing open access data sources can be utilized to move technical debt research forward

4. CONCLUSION
Identifying, measuring, reducing and monitoring technical debt is a real challenge for the software industry that is here to stay [13]. The research community working on creating theories and practices towards addressing these activities of technical debt management has made progress in creating a common definition, an initial research agenda, case studies and experiments with existing tools. Automated tooling and consistent measurement of technical debt continues to be a core industry need and research challenge that brings multiple software engineering activities together. The discussions during this year's workshop provided input for several fruitful small steps that can be taken towards that goal, as this report aimed to summarize.

The research community around understanding and managing technical debt has been growing steadily. In 2017, the 9th International Workshop on Managing Technical Debt will be held on May 22, 2017, collocated with XP 2017, the 18th International Conference in Agile Software Development in Cologne, Germany (https://www.sei.cmu.edu/community/td2017/).

A key question the community has also been addressing is how to best bring together key ideas, people and collaboration opportunities together to make effective and efficient progress. While these workshops have served the community well towards this goal the community has been growing steadily and there is a need to target the research and industry challenges more effectively collectively. In 2018 we will hold the first working conference focusing on technical debt, collocated with the International Conference on Software Engineering in Gothenburg, Sweden.

5. ACKNOWLEDGMENTS
Our thanks to the organizers of ICSME 2016, and ABB Corporate Research. We would also like to thank all authors, session chairs, presenters, and attendees. Their contributions were significant in helping move forward MTD's research agenda.

6. DISCLAIMER
The views and conclusions contained in this document are solely those of the individual authors(s) and should not be interpreted as representing official policies, either expressed or implied, of the Software Engineering Institute, Carnegie Mellon University, the U.S. Air Force, the U.S. Department of Defense, or the U.S. Government.

7. REFERENCES

[1] Dagstuhl Blog. https://mtd2016dagstuhl

[2] Philippe Kruchten, Robert L. Nord, Ipek Ozkaya: Technical Debt: From Metaphor to Theory and Practice. IEEE Software 29(6): *18-21 (2012)*

[3] Philippe Kruchten, Robert L. Nord, Ipek Ozkaya, Davide Falessi:Technical debt: towards a crisper definition report on the 4th international workshop on managing technical debt. ACM SIGSOFT Software Engineering Notes 38(5): 51-54 (2013)

[4] Managing Technical Debt in Software Engineering, Dagstuhl Reports, Vol. 6, Issue 4, April 17-22, 2016. http://www.dagstuhl.de/16162

[5] Wolfgang Trumler and Frances Paulisch. 2016. How "Specification by Example" and Test-driven Development Help to Avoid Technical Debt. *International Workshop on Managing Technical Debt (Raleigh, North Carolina, USA, Oct 4 2016)*

[6] Areti Ampatzoglou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Paris Avgeriou, Pekka Abrahamsson, Antonio Martini, Uwe Zdun and Kari Systa. 2016. "The Perception of Technical Debt in the Embedded Systems Domain: An Industrial Case Study" *International Workshop on Managing Technical Debt (Raleigh, North Carolina, USA, Oct 4 2016)*

[7] Mashel Albarak and Rami Bahsoon. 2016."Database Design Debts through Examining Schema Evolution" *International Workshop on Managing Technical Debt (Raleigh, North Carolina, USA, Oct 4 2016)*

[8] Derek Reimanis and Clemente Izurieta. 2016. "Towards Assessing the Technical Debt of Undesired Software Behaviors in Design Patterns" *International Workshop on Managing Technical Debt (Raleigh, North Carolina, USA, Oct 4 2016)*

[9] Francesca Arcelli Fontana, Riccardo Roveda and Marco Zanoni. 2016. "Technical Debt Indexes provided by tools: a preliminary discussion" *International Workshop on Managing Technical Debt (Raleigh, North Carolina, USA, Oct 4 2016)*

[10] Andriy Shapochka and Borys Omelayenko. 2016. "Practical Technical Debt Discovery by Matching Patterns in Assessment Graph" *International Workshop on Managing Technical Debt (Raleigh, North Carolina, USA, Oct 4 2016)*

[11] Shirin Akbarinasaji and Ayse Bener. 2016. "Adjusting the Balance Sheet by Appending Technical Debt" *International Workshop on Managing Technical Debt (Raleigh, North Carolina, USA, Oct 4 2016)*

[12] Clemente Izurieta, Ipek Ozkaya, Carolyn Seaman, Philippe Kruchten, Robert Nord, Will Snipes and Paris Avgeriou. 2016. "Perspectives on Managing Technical Debt: A Transition Point and Roadmap from Dagstuhl," *International Workshop on Technical Debt Analytics (TDA) (Hamilton, New Zealand, Dec 6 2016)*

[13] Paris Avgeriou, Philippe Kruchten, Robert L. Nord, Ipek Ozkaya, Carolyn B. Seaman: Reducing Friction in Software Development. IEEE Software 33(1): 66-73 (2016)