Perspectives on Managing Technical Debt

A Transition Point and Roadmap from Dagstuhl

Clemente Izurieta[1], Ipek Ozkaya[2], Carolyn Seaman[3], Philippe Kruchten[4], Robert Nord[2], Will Snipes[5], Paris Avgeriou[6]

[1]clemente.izurieta@montana.edu, Montana State University, Bozeman, MT, USA

[2]{ozkaya, rn}@sei.cmu.edu, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA

[3]cseaman@umbc.edu, University of Maryland Baltimore County, Maryland, MD, USA

[4]pbk@ece.ubc.ca, University of British Columbia, BC, Canada

[5]will.snipes@us.abb.com, ABB Corporate Research, Raleigh, NC, USA

[6]paris@cs.rug.nl, University of Groningen, Groningen, Netherlands

***Abstract*—Thirty-three practitioners, researchers, students, and tool vendors gathered in Dagstuhl, Germany, for five days in April 2016 to discuss the state of managing technical debt in software engineering. Participants reflected on the significant advances that the Managing Technical Debt (MTD) community has made since its inception in 2010; reached a consensus on a definition, called the Dagstuhl 16K technical debt definition; and discussed avenues for future progress in the area. This paper provides a brief history, summarizes current research, and offers a roadmap and a vision that describe the areas of research where significant challenges remain.**

***Keywords*—technical debt; software quality; software decay; software economics; software project management**

I. Introduction

While other software engineering disciplines—such as software sustainability, maintenance and evolution, refactoring, software quality, and empirical software engineering—have produced results relevant to managing technical debt, none of them alone suffice to model, manage, and communicate the different facets of the design trade-off problems involved in managing technical debt. Although the technical debt metaphor can be attributed to Cunningham [1], a community consensus on a pithy and focused definition has been a barrier for research progress that could address the most pressing immediate needs of the software engineering community. The technical debt metaphor describes a situation in which developers accept quality compromises in the current release to meet a deadline (e.g., delivering a release on time). A subsequent release that has been compromised will incur a higher cost in the form of higher maintenance efforts.

To date, the technical debt metaphor has served as a strong communication and reference mechanism, but the community now understands that technical debt is also a software development artifact that is incurred (mostly) unintentionally and discovered during later stages of software development. Moreover, the community also recognizes that the key research challenges ahead cannot be addressed by simply repurposing code quality and maintainability analysis as technical debt analytics. The Dagstuhl Seminar 16162 (Dagstuhl 16K) definition of technical debt focuses on design and implementation artifacts that affect maintainability and evolvability of software. This definition also prompted the community to address the problem of classifying artifacts in the periphery of the definition. Examples of the latter include social, documentation, process, and infrastructure debt. We thus present a conceptual model that allows for extension and context representation of various artifacts.

II. Background

The Management of Technical Debt (MTD) community has formally existed since 2010. Figure 1 depicts the timeline of prior events and illustrates new meetings that represent an increased level of activity. The outcome of these efforts has been more than 200 research papers written by research groups across the globe, systematic literature studies organizing the space and demonstrating gaps, and special issues in practitioner and research journals such as *IEEE Software* and the *Journal of Systems and Software*. Possibly the most welcomed and challenging outcome has been an ever-increasing involvement of the practitioner community. As a result, many tool vendors have started adding or repurposing features to support technical debt analysis. Many organizations are also looking into developing their own internal best practices for managing technical debt, and they need help.

Table 1 illustrates the topics on which technical debt research has focused since 2006. We clearly see a sharp distinction between artifacts that are easier to measure, such as code, and those that are not, such as people. It also shows which topics have received more and less research.

III. The Dagstuhl Format

Dagstuhl brought together researchers, practitioners, students, and tool vendors from academia and industry who are interested in the theoretical foundations of technical debt and how to manage it (e.g., techniques for measurement, analysis, and prevention). The organizers created a blog where attendees posted positions and started discussions to facilitate seeding of ideas prior to the seminar. Organizers grouped discussions and blog entries into relevant themes that included creating a common definition and conceptual model of technical debt, measurement and analysis of technical debt, management of technical debt, and a research roadmap for managing technical debt. No long talks were featured. Each day had three types of sessions. There was a plenary session for "lightning talks," in which each presenter had 10 minutes for presentation and questions on each day except for the last day of the seminar.

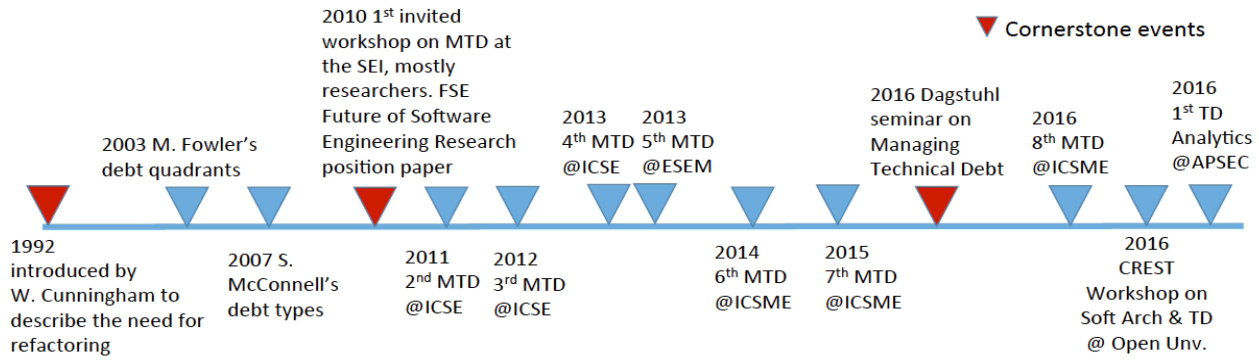
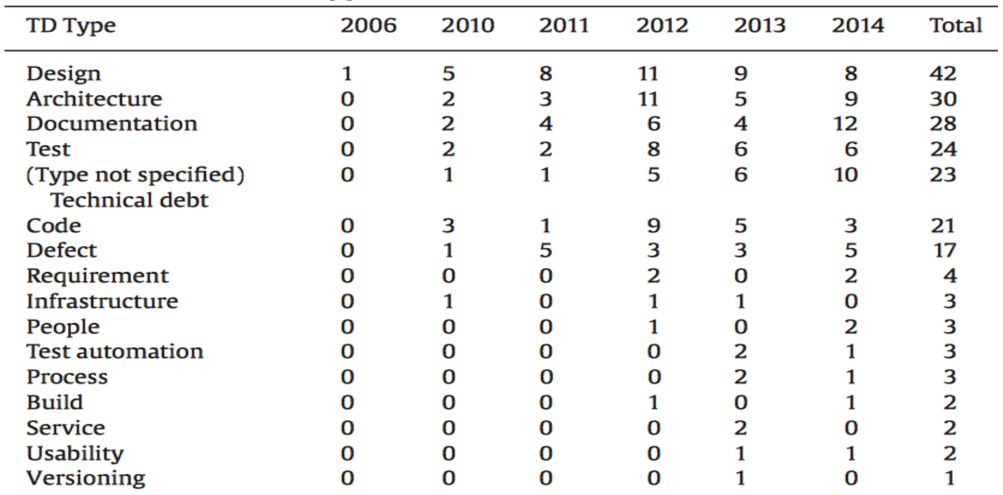

Cornerstone events
2010 1st invited workshop on MTD at the SEI, mostly researchers. FSE Future of Software Engineering Research position paper
2003 M. Fowler's debt quadrants
2013 4th MTD @ICSE
2013 5th MTD @ESEM
2016 Dagstuhl seminar on Managing Technical Debt
2016 8th MTD @ICSME
2016 1st TD Analytics @APSEC
1992 introduced by W. Cunningham to describe the need for refactoring
2007 S. McConnell's debt types
2011 2nd MTD @ICSE
2012 3rd MTD @ICSE
2014 6th MTD @ICSME
2015 7th MTD @ICSME
2016 CREST Workshop on Soft Arch & TD @ Open Unv.

Fig. 1. Technical debt community events [3]

Table 1. Where is research focused? [4]



TD Type	2006	2010	2011	2012	2013	2014	Total
Design	1	5	8	11	9	8	42
Architecture	0	2	3	11	5	9	30
Documentation	0	2	4	6	4	12	28
Test	0	2	2	8	6	6	24
(Type not specified) Technical debt	0	1	1	5	6	10	23
Code	0	3	1	9	5	3	21
Defect	0	1	5	3	3	5	17
Requirement	0	0	0	2	0	2	4
Infrastructure	0	1	0	1	1	0	3
People	0	0	0	1	0	2	3
Test automation	0	0	0	0	2	1	3
Process	0	0	0	0	2	1	3
Build	0	0	0	1	0	1	2
Service	0	0	0	0	2	0	2
Usability	0	0	0	0	1	1	2
Versioning	0	0	0	0	1	0	1

IV. Technical Debt

The significant outcomes of the seminar include a definition, a conceptual model, and a list of challenges that we face moving forward on the research agenda and transition prospects for managing technical debt. The definition and model serve as starting points for the community to build on and improve.

The Dagstuhl 16K definition presents an expansion over past definitions by taking into account the concerns heard from prior technical debt events and the thinking that has occurred over the years. Specifically, this definition elevates the concepts of evolvability and maintainability as the primary foci of technical debt research, combines design and implementation constructs, and highlights the context-specific trade-offs that need to be made in an expedient manner.

A. Definition of Technical Debt

Attendees converged on the following (*Dagstuhl 16K*) definition [2][5] for technical debt:

"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."

B. Conceptual Model and Related Activities of Technical Debt

Another outcome of the seminar was the recognition that, similar to other complex software engineering artifacts, technical debt is best described through multiple viewpoints. Concepts related to technical debt should be discussed based on two related viewpoints:

a) the viewpoint describing the properties, artifacts, and elements related to technical debt items (see Fig. 2)

b) the viewpoint articulating the management- and process-related activities to perform or the different states that debt may go through

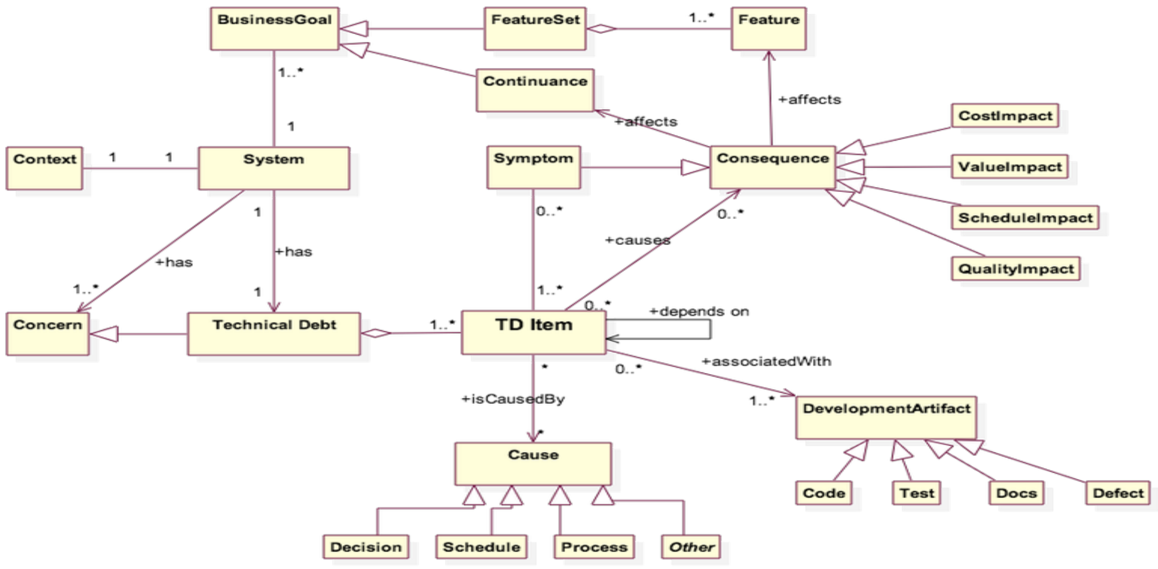


Fig. 2. Contextual figure of technical debt [2][5]

Figure 2 shows the conceptual model in the form of a UML class diagram, which focuses on the first viewpoint and helped the group converge on key concepts. The technical debt associated with a software-intensive system is composed of a set of technical debt (TD) items, and this technical debt is one of many concerns associated with a system. TD items have both causes and consequences. The cause of technical debt can be a process, a decision, an action (or lack thereof), or an event that triggers the existence of that TD item, such as schedule pressure, unavailability of a key person, or lack of information about a technical feature. The consequences of a TD item are many: technical debt can affect the value of the system, the costs of future changes, the schedule, and system quality. The business objectives of the sponsoring organization developing or maintaining the software system are affected in several ways: through delays, loss of quality for some features of the system, and difficulties in maintaining the system operations (continuance). A TD item is associated with one or more concrete, tangible artifacts of the software development process, primarily the code, but also to some extent the documentation, known defects, and tests associated with the system.

To keep with the financial metaphor, the cost impact of technical debt can be seen as composed of principal and interest. The principal is the cost savings gained by taking some initial approach or shortcut in development (the initial principal, often the initial benefit) or the cost that it would now take to develop a different or better solution (the current principal). The interest is comprised of costs that add up as time passes. There is *recurring interest:* additional cost incurred by the project in the presence of technical debt, due to reduced velocity (or productivity), induced defects, and loss of quality (maintainability is affected). And there is *accruing interest:* the additional cost of developing new software depending on not-quite-right code (evolvability is affected).

This view summarizing the elements related to technical debt, however, does not capture the activities that need to be conducted to manage technical debt or the states that debt may go through. An activity-focused view would map out research topics to be studied such as identifying, visualizing, assessing, and making decisions about technical debt. The phenomena all along the causal chain of causes and consequences are also important to investigate.

C. *Technical Debt Management*

Managing technical debt includes recognizing, analyzing, monitoring, and measuring it. Today many organizations do not have established practices to manage technical debt, and project managers and developers alike are asking for methods and tools to help them strategically plan, track, and pay down technical debt. We identified two broad high-priority challenges:

1) Developing effective tooling (academia and industry) to assist with assessing technical debt: A number of studies have examined the relationship between software code quality and technical debt. This work has applied detection of "code smells" (low internal code quality), coupling and cohesion, and dependency analysis to identify technical debt. However, empirical examples collected from industry all point out that the most significant technical debt is caused by design trade-offs, which are not detectable by measuring code quality. For example, an architectural decision encountered early in the design stage is the selection of a Visitor pattern vs. inheritance-based designs. Either design selection may be appropriate in the current context and would not yield smells; however, later evolutionary steps may reveal different maintenance problems, depending on the choice. Furthermore, several published case studies demonstrate that assessing technical debt appropriately requires combining several analysis techniques together.

2) Establishing an empirical basis and data science for technical debt: Well-defined benchmarks (with uncertainty levels) provide a basis for evaluating new approaches and ideas. They are also an essential first step toward creating an empirical basis on which work in this area can grow more effectively. Effective and well-accepted benchmarks allow researchers to validate their work and tailor empirical studies to be synergistic. Technical debt's evolving definition and its sensitivity to context have inhibited the development of benchmarks so far. An ideal benchmark for technical debt research would consist of a code base, architectural models (perhaps with several versions), and known TD items. New approaches could be run against these artifacts to see how well the approaches reveal TD items. Industry needs guidance for how and what data to collect and what artifacts they can make available to enable progress in understanding, measuring, and managing technical debt.

V. Research Roadmap and Vision

The Dagstuhl participants spent some time envisioning what the world would be like if technical debt research were as successful as we could ever hope it to be. The resulting vision is summarized in the following points:

- Technical debt will be managed as well as we now manage defects, vulnerabilities, and new features.
- We have a way to translate developer concerns to manager concerns—a basis for making decisions about allocating time for reducing technical debt.
- Technical debt will be mostly incurred intentionally.
- Projects that manage technical debt are more efficient, effective, and sustainable than projects that don't.
- There is support for up-front and continuous architectural work (vs. emergent architecture) and evidence that it helps avoid and manage technical debt.
- Tools support all aspects of technical debt management, and all stakeholders adopt them and use them.
- Technical debt-aware development (practices and tools) is an accepted way of producing software.

This vision set the stage for the beginnings of a research roadmap to guide future research to establish a cohesive body of knowledge about how to manage technical debt. The research roadmap consists of three major parts:

1) The Core: Defining, understanding, and operationalizing the concept of **value** with respect to technical debt. Specific activities include

- investigating the role of opportunity cost to measure the differences in value between a decision to implement new features that incur technical debt or make infrastructure improvements that avoid technical debt while foregoing the features
- understanding the factors, beyond principal and interest, that go into making decisions about incurring, paying off, and managing technical debt
- understanding how to model technical debt phenomena over time, which is not linear in software development

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. Specific activities include

- identifying the important context factors (e.g., code volatility, business context, development personnel) that affect the evaluation of technical debt
- understanding the relationship of other types of debt (e.g., social, infrastructure) as causes or consequences of technical debt
- exploring the role of development methodologies to manage technical debt

3) The Necessary Infrastructure: Building the shared infrastructure that facilitates all our research activities. Specific activities include

- sharing experimental data sets and study designs
- creating benchmarks in an effort to standardize tools and measures
- developing techniques to inject different forms of technical debt into data sets in order to evaluate, predict, and validate techniques

Conclusion

Technical debt is an active field of research with a growing community, as evidenced by the success of meetings and increased research output such as papers, commercial tools, and new projects. However, significant challenges remain to meet effective tooling demands, to establish an empirical basis, and to pinpoint artifacts that serve as inputs to measurement and analysis and most importantly to be useful in practice. The research roadmap is an evolving document and activity that requires active involvement from the greater community of academics and practitioners alike. The hope is that it will continue to be refined and instantiated at gatherings of researchers and engineers interested in future research in technical debt management.

Acknowledgment

Copyright 2016 Carnegie Mellon University. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT. [Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution. DM-0004135.

We thank Tamara Marshall-Keim for her expert input.

References

[1] W. Cunningham, "The WyCash portfolio management system," SIGPLAN OOPS Mess., Vol. 4, No. 2, pp. 29-30, Dec 1992.

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

[3] 8th International Workshop on Managing Technical Debt (MTD), Raleigh, NC, October 4, 2016.

[4] NSR Alves et al., Information and Software Technology 2016, Vol. 70, pp. 100-121.

[5] Dagstuhl Blog. https://mtd2016dagstuhl