Technical Debt at the Crossroads of Research and Practice
Report on the Fifth International Workshop on Managing Technical Debt

Davide Falessi
Fraunhofer Center for Experimental Software Engineering
College Park, MD, USA
dfalessi@fc-md.umd.edu

Philippe Kruchten
Electrical & Computer Engineering
University of British Columbia
Vancouver, Canada
pbk@ece.ubc.ca

Robert L. Nord, Ipek Ozkaya
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA, USA
{rn, ozkaya}@sei.cmu.edu

DOI: 10.1145/2579281.2579311
http://doi.acm.org/10.1145/2579281.2579311

ABSTRACT
Increasingly, software developers and managers use the metaphor of technical debt to communicate key trade-offs related to release and quality issues. We report here on the Fifth International Workshop on Managing Technical Debt, collocated with the Seventh International Symposium on Empirical Software Engineering and Measurement (ESEM 2013). The workshop participants reiterated the usefulness of the metaphor, shared emerging practices used in software development organizations, and emphasized the need for more research and better means for sharing emerging practices and results.

Keywords
Technical debt; software economics; software quality; software evolution; state of the practice.

1. INTRODUCTION
The technical debt metaphor introduced by Ward Cunningham in 1992 [1] has been further studied to better define the concept and its applicability to software development. Steve McConnell defines technical debt as “a design or construction approach that’s expedient in the short term, but that creates a technical context in which the same work will cost more to do later than it would cost to do now” [2]. McConnell further differentiates between two types of debt: Type 1, which is unintentional and nonstrategic, and Type 2, which is optimizing for the present and strategic for the short term or long term [3]. This distinction generates discussion, and often confusion, about what should be considered debt. For example, participants of the Second International Workshop on Managing Technical Debt debated questions such as whether to treat defects as technical debt or whether a lack of documentation constitutes technical debt [4]. The third workshop on technical debt aimed to address this confusion by delineating a technical debt landscape [5].

The topic of technical debt is complex and includes multiple aspects of software development. Understanding and managing technical debt seems possible only by combining solutions from different software engineering areas, including qualitative studies, software metrics, prediction, and release planning. During this Fifth International Workshop on Managing Technical Debt, we observed emerging practical approaches from industry that provide real benefits to managing technical debt. But there are several open problems, including the absence of a reliable metric to measure technical debt and the inability of available tools to describe the interest to be paid back.

From an empirical perspective, technical debt encapsulates some subtle aspects of software development and provides a context-dependent way of thinking about software quality across life-cycle phases, in a way that is amenable to quantitative analysis and hence objective observations. Thus, technical debt provides a potentially powerful combination of the maintainability, evolvability, quality, cost-effectiveness, and resource management aspects of software development for guiding empirical research. For example, technical debt encapsulates the incentive to measure not only instances of rework but also the opportunity costs should that rework not be performed. In addition, the concept of debt effectively transmits the results of this research to practitioners because they can recognize the existence and rational management of trade-offs.

Technical debt can be studied from different perspectives. However, during the discussions in the workshops, a unifying perspective has been emerging of technical debt as the invisible results of past decisions about software that affect its future. The effect can be negative if debt exists in the form of poorly managed risks, but if properly managed debt can be seen in a positive light as adding value in the form of deferred investment opportunities. The results of the fourth workshop on technical debt allowed us to make the definition of technical debt crisper by highlighting the following considerations. Technical debt [6]

- reifies an abstract concept
- is not simply bad quality
- can be introduced by a shift in context
- is not defects
- is not lack of process
- is not the new features not yet implemented
- implies both principal and interest
- depends on the future
- cannot be directly measured
- should not be completely eliminated
- should not be treated in isolation
- can be a wise investment

Large organizations have explicitly introduced technical debt management in their software development process as something to identify, value, and consider while planning iterations and releases. One such example is Cisco in Ireland [7].

Industry’s increasing interest in and the emergence of organization-specific practices can be seen as early indications that industry needs clearly defined practices for managing technical debt to deal with issues such as evolution, strategic resource management, and bridging the stakeholder communication gap. From our interactions with practitioners dealing with technical debt, we have noticed that organizations that embrace technical debt as part of their iteration-planning practices achieve success as a result of the following actions:

- making technical debt visible
- differentiating strategic structural technical debt from technical debt that emerges from low code quality
- using the elicited technical debt as a means for bridging the gap between the business and technical sides of the organization
- integrating technical debt into planning
- associating technical debt with future risk to identify a payback strategy

This fifth workshop consisted of invited presentations, a panel, and discussion sessions among the participants. The presentations, available here [8], included

Technical debt is about the future status of a system, especially the uncertainty in assessing the impact and size of the interest payments in software development. Managing technical debt involves predicting the future of a system.

2.5 Release planning
Release planning concerns deciding what new features or changes to implement during each release of a system [23]. As shown in previous studies [7], developers should balance the effort available in a given release between providing value to customers (i.e., more functionality) and removing technical debt. Thus, there is opportunity to apply methods from release planning to managing the health of a project and preventing too much debt from accumulating.

2.6 Architecture knowledge management and design trade-offs
The relationships among making decisions about architecture design, knowledge management, and design trade-offs are critical. Often, the decision to take on debt and the payback strategies are directly related to such design trade-offs [24]. Foundational work in architecture such as architecture patterns, architecturally significant requirements, and architecture evolution has unexplored relationships to taking on
technical debt, monitoring it, and evolving the system to pay down the debt. These concepts provide the foundation for tools and techniques that can improve how we manage technical debt.

3. FUTURE DIRECTIONS
More research is needed to quantify technical debt, produce repeatable results, and understand its relationship to software development.

Quantifying the principal and interest of technical debt to support decision making by managers and developers:

- Principal: Code assessment tools such as SonarQube [http://www.sonarqube.org/] and Cast Software’s Application Intelligence Platform [http://www.castsoftware.com/products/the-application-intelligence-platform] attach dollar figures to the status of a project. The amount indicates the cost necessary to modify the system to pay down the principal of its technical debt. This approach has two main drawbacks: 1) a perfect system with no debt is not a feasible target [25], and 2) scenarios exist in which the debt of a project exceeds its profit although the project still provides positive revenues and shutting it down would not make sense.
- Interest: Interest is not quantified in the same terms as the principal, so it is hard to trade off principal and interest.
- Decision making: These issues challenge the use of technical debt in practice. A project manager cannot be convinced of the value of managing technical debt if the dollar figure is incongruous with other measures. Tools provide many false positives. Developers can be demotivated by these red flags (e.g., data formatting mistakes), or they can take these as negative evaluations of their work.

Testing the hypothesis and sharing results that other researchers can repeat:

- Progress on managing technical debt takes advantage of existing work in code quality analysis and software measurement. However, we still need research that tests the basic hypotheses, such as whether modules that have low quality indicators also have real debt. This requires both articulating relevant hypotheses and running experiments with relevant data.

4. ACKNOWLEDGMENTS
We are grateful to the many participants in the technical debt workshops over the last four years, the contributors to the IEEE Software Special Issue on Technical Debt, and our consulting customers for their contributions. We extend our thanks to all those who have participated in the organization of this fifth workshop.

5. DISCLAIMER
The views and conclusions contained in this document are solely those of the individual author(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.

6. REFERENCES