Technical Debt in Software Development: from Metaphor to Theory
Report on the
Third International Workshop on Managing Technical Debt

Philippe Kruchten[1], Robert L. Nord[2], Ipek Ozkaya[2], and Joost Visser[3]
[1] University of British Columbia, Canada
pbk@ece.ubc.ca
[2] Software Engineering Institute, Carnegie Mellon University, USA
rn@sei.cmu.edu, ozkaya@sei.cmu.edu
[3]Software Improvement Group, Netherlands
j.visser@sig.eu

DOI: 10.1145/2347696.2347698
http://doi.acm.org/10.1145/2347696.2347698

Abstract

The technical debt metaphor is gaining significant traction in the software development community as a way to understand and communicate issues of intrinsic quality, value, and cost. This is a report on a third workshop on managing technical debt, which took place as part of the 34rd International Conference on Software Engineering (ICSE 2012). The goal of this third workshop was to discuss managing technical debt as a part of the research agenda for the software engineering field, in particular focusing on eliciting and visualizing debt, and creating payback strategies.

Keywords: technical debt, software economics, software quality

Introduction

Software developers and corporate managers frequently disagree about important decisions regarding how to invest scarce resources in development projects, especially for internal quality aspects that are crucial to system sustainability, but are largely invisible to management and customers, and do not generate short-term revenue. These aspects include code and design quality and documentation. Engineers and developers often advocate for investments in these areas, but executives question their value and frequently decline to approve them, to the long-term detriment of software projects. The situation is exacerbated in projects that must balance short deadlines with long-term sustainability.

The technical debt metaphor is gaining significant traction in the software development community, as a way to understand and communicate issues regarding intrinsic quality, value, and cost. Ward Cunningham first coined the metaphor in his 1992 Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) experience report in defense of relentless refactoring as a means of managing debt 1.

Technical debt is based on the idea that developers sometimes accept compromises in a system in one dimension (e.g., modularity) to meet an urgent demand in some other dimension (e.g., a deadline). Such compromises incur a debt on which interest must be paid and which should be repaid at some point for the long-term health of the project.

There is a key difference between debt that results from employing bad engineering practices and debt that is incurred through intentional decision-making in pursuit of a strategic goal 2. While technical debt is an appealing metaphor, theoretical foundations for its identification and management are lacking. In addition, while the term was originally coined in reference to coding practices, today the metaphor is applied more broadly across the project life cycle and may include practices of refactoring 3, test-driven development 4, iteration management 567, software architecture 89, and software craftsmanship 10.

The concept of technical debt can provide a basis on which the various stakeholders can reason about the best course of action for the evolution of a software product. As reflected by the composition of our program committee that includes practitioners, consultants, and researchers, this area has significant relevance to practicing software engineers and software engineering.

A first workshop on technical debt was held at the Software Engineering Institute in Pittsburgh on June 2 to 3, 2010. Its outcomes were published as a research position paper 11 summarizing the open research questions in the area.

The goal of the second workshop in 2011 was to come up with a more in-depth understanding of technical debt, its definition(s), characteristics, its different forms. The discussions of the second workshop proved that there is an increasing need to formulate a clear research agenda that is well-aligned with the industry challenges 12.

The goal of this third workshop was to discuss managing technical debt as a part of the research agenda for the software engineering field, in particular focusing on eliciting and visualizing debt, and creating payback strategies. One objective related to this goal was to understand the processes that lead to technical debt and its indicators, such as degrading system quality and inability to maintain code. A second objective was to understand how to handle technical debt by examining payback strategies and investigating the type of tooling that may be required to assist software developers and development managers to assess its cost. The discussions of the third workshop proved that there is an increasing need to formulate a clear research agenda that is well-aligned with the industry challenges.

The Workshop

The software engineering community is in the process of building the research agenda around managing technical debt. The purpose of these initial workshops is to bring forward work in progress and ideas from the entire community to collectively vet their validity for the future.

The workshop was structured to facilitate a dialog between two particular groups: 1) software engineers who need to elicit, communicate, and manage technical debt pertaining to different facets of their projects; and 2) researchers who examine different aspects of technical debt, with particular interest in applying their research in practice and collecting empirical evidence related to their research as it applies to technical debt.

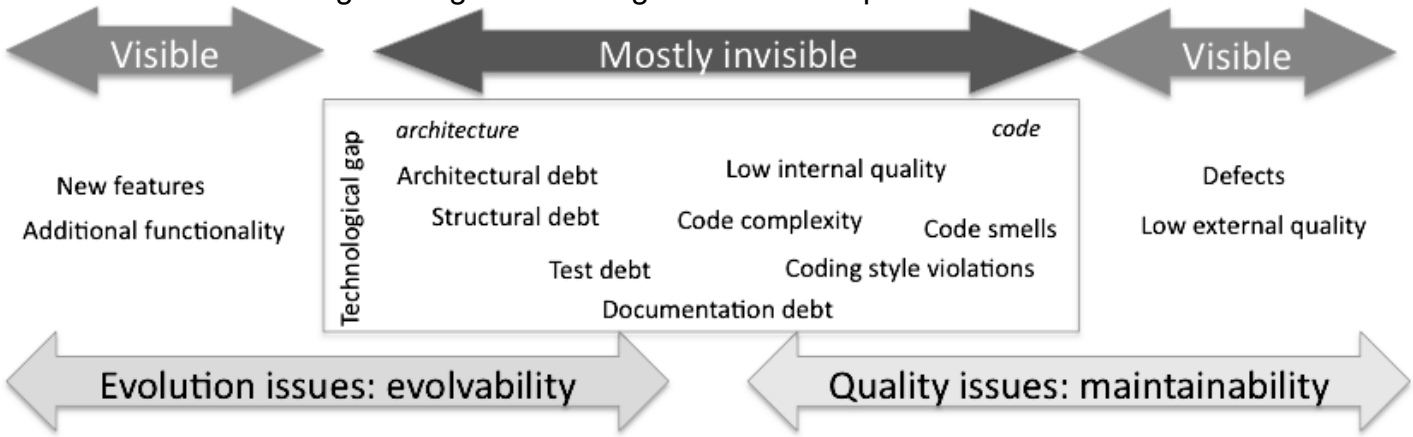

Figure 1: the technical debt landscape

Managing technical debt is a broad concern of software engineering that blends research and practice. This can be seen from the program and those involved in the workshop program selection process. The workshop had six sessions, each dedicated to a specific subject. We had 12 paper presentations grouped into four topics and two guided discussions 12.

- Industry challenges for the research community
- Landscape and other perspectives of technical debt in the software development lifecycle
- Discussion on the topic: Can we formulate the technical debt landscape?
- Eliciting and visualizing debt
- Research perspective on code and defects
- Discussion on the topic: What kinds of tools and techniques do we envision to help practitioners tackle technical debt?

The accepted submissions cover a range of topics such as: estimating the size and cost of debt, eliciting and visualizing debt, the technical debt landscape ranging from technical debt in software ecosystems to requirements, design and build, and the relationship between code defects and debt. For the complete set of workshop papers, see the ICSE 2012 proceedings at the IEEE Digital Library 14.

Here is a summary of these sessions, highlighting new insights that emerged.

First Steps From Metaphor to Theory

From the original description by Cunningham ("not quite right code" which we postpone making it right 1), various people have used the concept of technical "debt" to describe many other kinds of debts or "ills" of software development, encompassing broadly anything that stands in the way of deploying, selling or evolving a software system, anything that adds to the friction software development suffers from: test debt, people debt, architectural debt. As a result, the concept of debt gets somewhat diluted. Is a visible bug or defect technical debt? Is a new requirement, a new function or feature not yet implemented requirement debt?

Once we identify tools, such as static analyzers to assist us in the identification of technical debt, there is a danger of equating technical debt with "whatever the tool can detect." This approach leads to leaving aside large amounts of technical debt undetectable by tools: structural or architectural debt, or technological gaps.

Gaps in technology are of particular interest since the debt incurred is not the result of making a wrong choice, originally, but rather the result of the evolution of the context, merely the passing of time, so that the choice is "not quite right" in retrospect. Technical debt in this case is due to external events: technological obsolescence, change of environment, rapid commercial success, advent of new and better technologies, and so on.

To make some progress, we need to go beyond debt as a "rhetorical concept" 11, we need a better definition of what constitutes technical debt, and some perspective or viewpoints that allow us to reason across a wide range of technical debt.

Fig. 1 shows a possible organization of a *landscape* of technical debt, or rather of software improvement from a given state. We can distinguish visible elements such as potential new functionality and defects, and the invisible elements (or rather, visible only to the software developers). We can see that on the left we are dealing mainly with software evolution or difficulty of evolution, whereas on the right we are dealing with quality issues, both internal and external quality. We propose to limit the term "debt" to the invisible part, that is, the elements in the box.

Once we scope the concept of technical debt, the next step is to move from a useful metaphor, to a theoretical framework that would allow us to reason about it.

GUTSI[3]: the Grand Unified Theory of Software Improvement

A simple model to tackle technical debt is to consider a software development endeavor as a *sequence of changes*, or improvements. At a given point in time, the past set of changes is what defines the current state of the software. Some of these changes are the events that have triggered any of the current debt.

The main issue at hand is how to decide about *future* changes: what evolution do we want to see the software system undergo, and in which sequence? This evolution is in most cases constrained by *cost*: the resources that we can apply to making these changes, and most likely driven by *value*, as seen by the external stakeholders.

The decision making process about which sequence of changes we want to apply could be the main reconciling point across the whole landscape shown above, and since it is related to balancing cost and value, maybe economic or financial models could become the unifying concept;

- Net Present Value (NPV) for a product, from the finance world,
- Total Cost of Ownership (TCO) for an IT system, popularized in 1987 by the Gartner group
- Opportunity Cost
- Real Option Analysis (or valuation) (ROA)

During the workshop, of these four economic concepts, it seemed that Net Present Value would be the most promising: better formalized than opportunity cost, simpler and less proprietary than TCO, while ROA can be seen as a probabilistic extension to NPV.

[3] This sounded like Schwyzerdütsch, and we were in Zürich, after all; plus a wink to GUTSE 15.

Technical debt should not be treated in isolation from new functionality or defects, even if we chose not to include them in the definition of "debt". The challenge is expressing them all in terms of sequences of changes, associated with a cost and a value (over time). These changes are not independent. Their interdependencies play a big role, as M. Denne and J. Cleland-Huang have shown, in particular visible features dependent on less visible architectural aspects 1617.

Summary

The main future directions that were discussed are

- What should the research agenda look like? It should include:
  - Characterizing debt.
  - If defining technical debt is too difficult, can we say what is not?
  - Models to show where technical debt slows development and where it speeds it up and where the breaking point exists such that it is no longer efficient to carry technical debt.
  - Debt transfers and externalities (costs someone incurs and someone else pays for).
- A collection of examples of technical debt—having a catalog of examples from various stakeholder points of view could help us develop a better taxonomy. The collection could include:
  - More studies grounding technical debt in industry with case studies of genuine application of concepts to real situation (looking to invest to reduce the chaos of systems with expectation to produce profit).
  - Dataset for case studies large and significant enough to be available to all people; guidance on what to collect for the dataset.
  - Reasoning about commercial-off-the-shelf (COTS) products and services in the cloud, technical debt in ecosystems.
- Relating to mathematical theory of decisions such as:
  - Decision making under uncertainty in a multi-attribute world; depending on the situation, different valuation approaches makes sense.
  - Decision support (not just static analysis but other analyses that feed into decision support).
  - Dynamic decision making (real options is one technique; when, if ever, do you pay down debt; when is the optimal time to exercise an option to make an investment).
- While technical debt has a strong negative connotation, it can also be seen in a more positive light as a tactical investment in a project, something to gain a temporary advantage to later be repaid or not. Studies could include:
  - Present value study – retrospective of efforts made to reduce maintenance costs through refactoring.

Acknowledgments

We extend our thanks to all those who have participated in the organization of this workshop, particularly to the program committee members:

- Eric Bouwers, Technical University Delft, Netherlands
- Yuangfang Cai, Drexel University, USA
- Rafael Capilla, Universidad Rey Juan Carlos, Spain
- Jeromy Carriere, eBay, USA
- Bill Curtis, CAST, USA
- Hakan Erdogmus, Kalemun Research, Canada
- David Garlan, Carnegie Mellon University, USA
- Israel Gat, Cutter Consortium, USA
- Matthew Heusser, Socialtext, USA
- Jim Highsmith, ThoughtWorks, USA
- Rick Kazman, University of Hawaii and the Software Engineering Institute, USA
- Erin Lim, University of British Columbia, Canada
- Alan MacCormack, MIT, USA
- Don O'Connell, Boeing, USA
- Raghu Sangwan, Penn State University, USA
- Carolyn Seaman, University of Maryland Baltimore County, USA
- Kevin Sullivan, University of Virginia, USA
- Peri Tarr, IBM, USA
- Ted Theodoropoulus, Acrowire, USA

Disclaimer

References

1. Cunningham, W. 1992. The WyCash Portfolio Management System. *OOPSLA' 92 Experience Report*.
2. McConnell, S. 2007. Technical Debt. 10x Software Development. Available from: http://www.construx.com/Page.aspx?cid=2801
3. Fowler, M. 1999. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional.
4. Erdogmus, H., Morisio, M., and Torchiano, M. 2005. On the Effectiveness of the Test-First Approach to Programming. IEEE Trans. Softw. Eng. 31, 3 (Mar. 2005), 226-237.
5. Cohn, M. 2006. Agile Estimation and Planning, Prentice Hall.
6. Highsmith, J. 2009. Agile Project Management, 2. Addison Wesley.
7. Sutherland, J. 2005. Future of Scrum: Parallel Pipelining of Sprints in Complex Projects. Proceedings of the Agile 2005 Conference, IEEE CS, pp. 90-102.
8. Brown, N., Nord, R., Ozkaya, I. 2010. Enabling Agility through Architecture, Crosstalk, Nov/Dec 2010.
9. InfoQ: What Color is your Backlog? Interview with Philippe Kruchten, May 02, 2010. Available from: http://www.infoq.com/news/2010/05/what-color-backlog
10. Martin, Robert C. 2008. Clean Code: A Handbook of Agile Software Craftsmanship. Addison Wesley.
11. Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., MacCormack, A., Nord, R., Ozkaya, I., Sangwan, R., Seaman, C., Sullivan, K., Zazworka, N., 2010. Managing Technical Debt in Software-Reliant Systems, 2010 FSE/SDP Workshop on the Future of Software Engineering Research, ACM. doi: 10.1145/1882362.1882373
12. Second International Workshop on Managing Technical Debt http://www.sei.cmu.edu/community/td2011/
13. I. Ozkaya, P. Kruchten, R. Nord, and N. Brown, 2011. Managing technical debt in software development ACM Software Engineering Notes. 36, 5. 33-35.
14. Third International Workshop on Managing Technical Debt http://www.sei.cmu.edu/community/td2012/
15. P. Johnson, and M. Ekstedt. 2005. The grand unified theory of software engineering. Industriella informations- och styrsystem, KTH.
16. M. Denne, and J. Cleland-Huang. 2004. Software by Numbers: Low-Risk, High-Return Development. Prentice Hall.
17. M. Denne, and J. Cleland-Huang, 2004. The Incremental Funding Method: Data-Driven Software Development IEEE Software. 21, 3. (May/Jun), 39-47.