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

- Robert Eisenberg, Software Engineer Senior Staff, Lockheed Martin Corporation. *Management of Technical Debt: A Lockheed Martin Experience Report.*

- Todd Fritsche, Chief Architect, Siemens Healthcare Health Services. *Technical Debt: Identification, Payment, and Restructuring*.
- Murray Cantor, Distinguished Engineer, IBM Rational Software. *Technical Liability: Extending the Technical Debt Metaphor*.
- Olivier Gaudin, founder and CEO, SonarSource. *Take Control of Your Technical Debt.*
- Guenther Ruhe, Industrial Research Chair in Software Engineering, University of Calgary. *Product Release Planning in Consideration of Technical Debt.*
- Carolyn Seaman, Associate Professor at UMBC, Research Fellow at Fraunhofer USA. *Technical Debt: At the Intersection of Decades of Empirical Software Engineering Research*.

The presentations and discussions among the participants of the workshop reiterated that after decades of empirical software engineering research and practice, the study of managing technical debt is at a turning point [9]. In her opening talk, Carolyn Seaman stated that "it is only a little hyperbolic to call this a watershed moment for empirical [software engineering] study, where many areas of progress are coming to a head at the same time" [10].

In Section 2, we describe these different software engineering areas that contribute to the research and practices in managing technical debt that were discussed during the workshop. Section 3 summarizes future research directions.

2. THE CROSSROADS

2.1 Software aging and decay

Technical debt resonates with maintenance and evolution challenges when it needs to be repaid, especially with refactoring and re-architecting. Lehman observes that for systems to remain useful they must change, and that change will increase their complexity, leading to software decay if refactoring is not done as needed [11]. Parnas calls this phenomenon "software aging," reflecting the failure of a product owner to modify software to meet changing needs [12]. Lehman's observations about system evolution still apply to projects that follow agile software development approaches [13]. While Lehman's laws of evolution provide insight into the inevitability of and need for re-architecting (and the high potential of debt that can accumulate), work in this area has not addressed abstraction between code and architecture beyond the application of pattern-based approaches [14], which a better understanding of architectural technical debt can address.

Technical debt is in some way a restatement of ideas established in the software evolution and maintenance literature: systems age and they need to keep evolving to continue to meet their business goals. The theories within software evolution and system aging apply in understanding what debt is and how to manage it.

2.2 Qualitative methods

Qualitative research methods were designed, mostly by social scientists [15], to study the complexities of human behavior (e.g., motivation, communication, and understanding). Other disciplines have developed qualitative research methods, and researchers commonly use them to handle the complexity of issues involving human behavior [16].

In general, qualitative methods are preferable to quantitative ones to study phenomena for which reliable metrics (leading to quantitative data) do not yet exist [17]. Moreover, qualitative methods are particularly effective in capturing context factors [18]. To date, technical debt does not have reliable metrics [19] that software engineers and managers can use in decision making. A key challenge in creating a quantitative measurement environment for measuring technical debt is understanding the definitions and acceptable ranges of technical debt across contexts. However, this challenge also presents an opportunity to investigate using qualitative methods to understand and manage technical debt.

Application of qualitative methods in software engineering has reached a level of maturity. Thus, it is time to apply qualitative studies to manage technical debt.

2.3 Software metrics

Software metrics started in the early 1970s by measuring code complexity of software systems [20]. Two main problems in using software metrics are their reliability and their cost. Reliability is a problem because metrics can measure a different phenomenon than the one of interest, or analysts could make mistakes during data collection. Using metrics to guide development requires investing in appropriate tool support as well as in processes that can seamlessly integrate the results into decision making for software development, both of which require resources that development teams may not have.

However, progress in software metrics is rapidly advancing. Both researchers and tool vendors, including groups within large organizations such as Microsoft and Rational Software, have developed many tools. The software measurement and analysis research communities (e.g., Mining Software Repositories and Predictive Models in Software Engineering) are making solid progress in creating a reliable underlying theory to integrate data collection, analysis, and visualization into the development process. However, because we don't yet have reliable metrics for technical debt [19], we are limited to showing trends in other quality metrics. For example, we might assume that as complexity increases, more debt accumulates in the system. This approach has severe limitations because it assumes a linear relationship between metrics for code quality and the principal and interest of technical debt.

Bringing the software measurement and analysis and technical debt research communities together can accelerate progress. Questions such as what kinds of anomalies constitute debt, whether there is a relationship between code quality and high interest rates on debt, and whether there are certain classes of code quality issues that contribute to debt are all open questions that the advancement in tools and software metrics can start addressing with rigorous research methods.

2.4 Prediction

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.

Prediction models have been studied and adopted in software engineering, especially models for estimating effort [21] and bugs [22]. However, computing the interest of technical debt in a system requires making estimates about the future and includes uncertainty. Approaches that overlook uncertainty could provide unrealistic results. Predicting the negative impact of technical debt as interest payments and coming up with strategies to reduce the interest accordingly can take advantage of research on predictive models.

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.

Expanding the application of the technical debt concept to other areas in software development:

- The definition of technical debt that is most accepted describes it as a measurement of the intrinsic quality of the software. This description also has a direct relationship to its roots in Cunningham's use of technical debt to describe a system's quality. However, software development organizations, especially large ones, see benefit in communicating such intrinsic quality issues in other software development artifacts that contribute to the development of a system, in particular requirements and test artifacts. The value and usefulness of expanding the concept to understanding the quality and trade-offs within such artifacts require further research.

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

6. REFERENCES

[1] W. Cunningham. The WyCash Portfolio Management System. In *OOPSLA' 92 Experience Report*. Vancouver, 1992.

[2] S. McConnell. *Managing Technical Debt* [Webinar], Sep. 2011. Available from http://www.youtube.com/watch?v=lEKvzEyNtbk

[3] S. McConnell. *Technical Debt*. 10x Software Development, 2007. Available from http://www.construx.com/Page.aspx?cid=2801

[4] I. Ozkaya, P. Kruchten, R. Nord, and N. Brown. Managing technical debt in software development. *ACM Softw. Eng. Notes* 36, 5 (2011), 33-35.

[5] P. Kruchten, R. L. Nord, and I. Ozkaya. Technical debt: from metaphor to theory and practice. *IEEE Software* 29, 6 (2012), 18-21.

[6] P. Kruchten, R. L. Nord, I. Ozkaya, and D. Falessi. Technical debt: towards a crisper definition; report on the 4th International Workshop on Managing Technical Debt. *SIGSOFT Softw. Eng. Notes* 38, 5 (2013), 51-54.

[7] K. Power. Understanding the impact of technical debt on the capacity and velocity of teams and organizations: viewing team and organization capacity as a portfolio of real options. In *4th International Workshop on Managing Technical Debt (MTD), IEEE CS*, 2013, 28-31.

[8] Fifth International Workshop on Managing Technical Debt, Oct. 9, 2013. Available from http://www.sei.cmu.edu/community/td2013esem/program

[9] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazworka. Managing technical debt in software-reliant systems. In *Future of Software Engineering Research (FoSER 2010) Workshop*, part of FSE 2010. Santa Fe, NM. ACM, 2010, 47-52.

[10] F. Shull, D. Falessi, C. Seaman, M. Diep, and L. Layman. Technical debt: showing the way for better transfer of empirical results. Forthcoming in *Future of Software Engineering*, published in honor of the 60th birthday of Prof. Dr. H. Dieter Rombach, 2013.

[11] M. M. Lehman. *Program Evolution: Processes of Software Change*. Academic Press Professional, San Diego, 1985.

[12] D. L. Parnas. Software aging. In *Proceedings of the 16th International Conference on Software Engineering*. Los Alamitos, CA, 1994, 279-287.

[13] R. Sindhgatta, N. C. Narendra, and B. Sengupta. Software evolution in agile development: a case study. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*. ACM, New York, 2010, 105-114.

[14] C. J. Neill and P. A. Laplante. Paying down design debt with strategic refactoring. *Computer* 39, 12 (2006), 131-134.

[15] S. J. Taylor and R. Bogdan. *Introduction to Qualitative Research Methods*. New York: John Wiley & Sons, 1984.

[16] C. B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Trans. Softw. Eng.* 25, 4 (1999), 557-572.

[17] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic, 2000.

[18] E. Lim, N. Taksande, and C. B. Seaman. A balancing act: what software practitioners have to say about technical debt. *IEEE Software* 29, 6 (2012), 22-27.

[19] R. L. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas. In search of a metric for managing architectural technical debt. *WICSA/ECSA*, 2012, 91-100.

[20] T. J. McCabe. A complexity measure. *IEEE Trans. Softw. Eng.* 2, 4 (1976), 308-320.

[21] M. Jorgensen and M. Shepperd. A systematic review of software development cost estimation studies. *IEEE Trans. Softw. Eng.* 33, 1 (Jan. 2007), 33-53.

[22] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Trans. Softw. Eng.* 38, 6 (Nov. 2012), 1276-1304.

[23] G. Ruhe. *Product Release Planning: Methods, Tools and Applications*. Taylor & Francis, 2010.

[24] R. L. Nord, I. Ozkaya, and R. S. Sangwan. Making architecture visible to improve flow management in lean software development. *IEEE Software*, Special Issue on Lean Software Development, 29, 5 (Sep./Oct. 2012), 33-39.

[25] D. Falessi, M. A. Shaw, F. Shull, K. Mullen, and M. Stein. Practical considerations, challenges, and requirements of tool-support for managing technical debt. In *4th International Workshop on Managing Technical Debt (MTD), IEEE CS*, 2013, 16-19.