Technical Debt: Towards a Crisper Definition

Report on the 4th International Workshop on Managing Technical Debt

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

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

DOI: 10.1145/2507288.2507326
http://doi.acm.org/10.1145/2507288.2507326

ABSTRACT

As the pace of software delivery increases and technology rapidly changes, organizations seek guidance on how to insure the sustainability of their software development effort. Over the past four years running the workshops on Managing Technical Debt, we have seen increased interest from the software industry to understanding and managing technical debt. A better understanding of the concept of technical debt, and how to approach it, both from a theoretical and a practical perspective is necessary to advance its state of the art and practice. In this paper, we highlight the current confusion in industry on the definition of technical debt, their contributions that have led to a deeper understanding of this concept and the limits of the metaphor, the criteria to discriminate what is technical debt and not, and areas of further investigation.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *restructuring.*

General Terms

Management, Measurement, Design, Economics.

Keywords

technical debt; software economics; software quality; software evolution; state of the practice.

1. INTRODUCTION

Technical debt is a metaphor introduced by Ward Cunningham in1992 [3] to help us think about a problem that is crippling many software endeavors. In his metaphor, doing things the "quick and dirty" way sets us up with a technical debt, which is similar to a financial debt. Like a financial debt, the technical debt incurs interest payments, which come in the form of the extra effort that we have to dedicate in future development because of this quick and dirty design choice. We can choose to continue paying the interest, or we can pay down the principal by refactoring the quick and dirty design into a better design. Although there is a cost to pay down the principal, we gain by reduced interest payments in the future as noted by Fowler [7].

The metaphor saw little use for many years, but suddenly around 2000 and curiously in parallel with the advent of agile methods (though this may be just coincidental), it gained increased attention, especially in the blogosphere and in many software gurus' sermons. The phrase started to be used to label all kinds of software ills. Since 2010 it became the subject of more scrutiny and researchers have attempted to better understand the concept, define it, measure it, assess its impact, and propose tools and methods to manage it [2].

A more recent definition of technical debt which seems to aptly convey the current consensus is given by Steve McConnell [19]: *"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 (including increased cost over time)."*

During the third workshop on managing technical debt in 2012, academics and industry practitioners drew a map of the "technical debt landscape"; see Fig. 1 [12][13]. Aligned with McConnell's taxonomy [18], the landscape covers at the left end intentional technical debt (architectural debt), technical debt due to a change in context (technological gap), and at the right end technical debt of a smaller granularity, mostly low internal code quality. On the left of the figure, technical debt affects mostly the evolvability of the software system, while on the right it affects mainly its maintainability.

Further reflections and studies on technical debt led to a deeper understanding of the concept and the limits of the metaphor, criteria to discriminate what is technical debt and not, and areas of further investigation, and the technical debt landscape became even crisper during a subsequent workshop held in May 2013 [14].

From an empirical perspective, technical debt encapsulates some delicate aspects of software development and provides a context-dependent way of thinking about software quality across lifecycle phases, in a way that is amenable to quantitative analysis and hence objective observations. Thus, technical debt provides a useful framework for guiding empirical research (e.g., the incentive to measure not only instances of rework but the opportunity costs should that rework *not* be performed) and effectively transmitting the results of this research to practitioners by recognizing the existence and rational management of tradeoffs [29].

While the phrase is used a lot now in industry, we wonder if it is a useful concept that can lead to improvements, or at least a better understanding, of the way we develop software. It is probably a useful *rhetorical* concept, as pointed out by many [2], to start a dialogue between business people and technical people about schedule pressure, architecture, poor internal quality, and release planning. Now we want to understand if it is possible to go beyond the rhetoric and get to the point where it leads to actionable, objective, concrete facts, data, techniques or even tools.

One of the issues is that the metaphor has been applied loosely to a vast range of issues or concepts, and that the metaphor, like any good metaphor, has its limits [15]. We should understand what these limits are: technical debt in software development is not literally a financial debt, like a mortgage. *"…there is a plethora of attention-grabbing pronouncements in cyberspace that have not been evaluated before they were published, often reflecting the authors' guesses and experience on the subject of Technical Debt."* [30].

This being said, the metaphor proved to be useful, and large organizations have explicitly introduced it in some form or another in their software development process, as something to identify, value, and take into consideration while planning iterations and releases; one such example is Cisco in Ireland [25]. There is also increasing anecdotal evidence that we have gathered that organizations have started to create working groups for defining a technical debt practice, or have started tagging particular tasks as technical debt, for example in their backlogs to draw better and more quantifiable attention them.

The challenges faced by our industry are to decide whether technical debt as a practice has common, repeatable and clear boundaries, and how to concretely manage it.

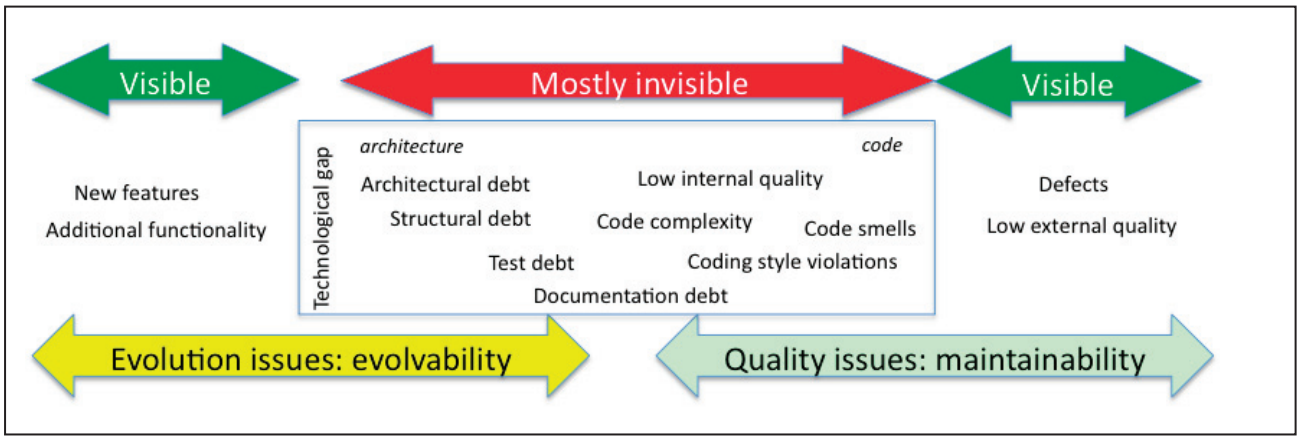

Figure 1. Technical debt landscape

2. TAKE-AWAYS FROM INDUSTRY FOR INDUSTRY

We now have considerable material from small and large industries about technical debt, how they perceive it and manage it. This includes companies such as IBM, Cisco, Siemens, Google, Lockheed-Martin, and so on. We have reports from companies who have a different interest at stake, selling tools or services pertaining to technical debt: Cast Software, Software Improvement Group, ThoughtWorks, Cutter Consortium, and so on. We also draw from industry contributions that we unfortunately cannot cite in this paper, drawing from reports that are not publicly available.

The increasing industry interest and emergence of organization specific practices can be seen as an early indication that industry needs a clearly defined practical managing-technical-debt practice to deal with issues such as evolution, strategic resource management and bridging the stakeholder communication gap. While often the impact of technical debt is seen as the inevitable consequences of "death by a thousand cuts" understanding the key contributors and deciding whether managing them as debt would help is an important first step. From our interactions with practitioners dealing with technical debt, we notice that organizations that have embraced 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,
- integrating technical debt into planning,
- associating technical debt with future risk to identify a pay-back strategy.

In the remainder of this section we develop the above-mentioned concepts.

A. Technical Debt Reifies an Abstract Concept

Technical debt is indeed a useful rhetorical concept to foster dialogue between business and technical actors [2]. This has been confirmed again and again by studies of practitioners [10][11][17]. On one hand technical people do not appreciate the value of shorter time to market, quick delivery, rapid tactical changes of direction; but on the other hand business people do not always realize the impact of earlier design decisions made and the cost incurred downstream. By identifying concrete items of technical debt, looking at their impact, evaluating the life-cycle costs associated with them, and introducing mechanisms for expressing technical debt and estimating its impact, the pains of software evolution can be made more real and technical debt reduction can be planned together with other items such as new features, defects, and architectural elements.

B. Technical Debt is not Simply Bad Quality

Original definitions of technical debt led us to think it is simply bad code quality, using negative terms: 'quick-and-dirty', 'short-cuts', 'bad design choices', 'death by a thousand cuts' and so on. As shown in Fig. 1, low internal code quality is effectively a kind of technical debt, maybe the prevalent kind. Tools including static code analyzers assist in identifying problems with low internal quality and related issues with documentation and testing. However, as pointed out by McConnell [18] and Fowler [7] there are also deliberate, "intentional", strategic decisions at the level of the structure or architecture of the system, or the choice of technologies, that are done for an immediate gain, usually reducing time-to-market, which are also technical debt, and not at all bad code quality [21][32]. You may decide not to take into account multiple languages for user-interface, and defer this choice to a later time, when the original market's need have been satisfied. This does not mean that your code is of bad quality.

C. Technical Debt can be Introduced by a Shift in Context

As noted by Letouzey [16] technical debt at the structural level can also be introduced by a shift in context. Specifically, the system is used in other circumstances than originally envisaged, or new technologies have emerged, invalidating what was deemed originally a good design decision. The debt incurred isn't the result of a bad decision, but rather the results of the context's evolution. As a metaphor presented by Judith Bishop [1], if you are a good citizen and pay on time the unexpected can happen to change the environment that incurs a debt. The shift in context could traverse supply chains and cross organizational boundaries [11][20][22].

D. Defects are not Technical Debt

As a corollary to the points above, low external quality, that is, visible to the user in the form of defects, bugs, and so on, should not be considered as technical debt, as this would dilute the term too much. Speaking of defects as "quality debt" [25] may be confusing. Most defects have an immediate, current impact on the value of the product, whereas technical debt will be only felt in the future in the form of additional costs. Thus, time is an important factor to qualify technical debt; we'll revisit this later in section H.

E. Lack of process is not Technical Debt

Due to time and resource constraints, not all required software life-cycle activities may be completed on time, for example, running all of the test

suites or not documenting the complete architecture. Tagging such actions as "process debt," results in the inability to articulate the impact of the design approach on the quality and the maintainability of the system. As such, any unfinished and postponed work as espoused by some [26], such as not completing a process task, is not technical debt. Technical debt is increasingly embraced by industry because it identifies direct system characteristics and should be confined as such. As an example, the increasingly complex build tree creates unnecessary runtime resource consumption, hence the need to periodically maintain to avoid the buildup of debt [23].

F. New Features not yet Implemented are not Technical Debt

New features visible to the user in the form of additional functionality should not be considered as technical debt, as this would also dilute the term. However, the argument can be made that a debt is incurred when sufficient requirements analysis is not conducted making it possible to miss intent leading to rework [5].

G. Technical Debt Implies both Principal and Interests

A design choice that has no permanent consequence on future cost of changes, that is, incurs no form of interest payment, probably should not be labeled as technical debt, but just as an alternative choice. The presence of some form of interest, either constant (productivity is affected constantly [25][31]), or in the form of a balloon payment (additional cost to retrofit an alternate solution), should be an important criterion for deciding if a design approach is in debt as stressed by McConnell [19].

Consider the example of a feature X that can be implemented by a simple strategy A or a more complex and costly strategy B. You choose A, and A leads to no additional costs in the future, until it can be simply replaced by B if and when B is needed. Then the choice of A may not be considered as a technical debt, simply as a wise alternative.

H. Technical Debt Depends on the Future

Many authors have noted that the financial metaphor has some limitations; unlike financial debt, it is possible to walk away from your debt [2]. Klaus Schmid re-iterated that something can only be qualified as technical debt when we know what will happen in the future [27]. If the system stops evolving, or if a part of the system is very stable and unaffected by future changes, then there is no reason to repay the debt (fix the code, refactor the design), and therefore this is just a kind of latent debt. This point puts a different perspective on approaches to measure technical debt and put a monetary value to it [4][16][8]. Only when the future is known can technical debt be given an absolute value in terms of effort (to repay).

Schmid distinguishes potential from effective technical debt [28]. What many tools will identify is the total potential technical debt (mostly of the low internal quality type); the total effective technical debt of a given system is smaller and depends on the evolution strategy: what part of the systems will need to evolve to go forward, and are impacted by earlier decisions. It is possible to take a probabilistic approach to this evolution, a technique similar to that used in risk management, to lead to an estimate of a probable effective technical debt.

I. Technical Debt cannot be Directly Measured

This is a direct consequence of the previous point. However, it is possible to give estimations of various sub-types of technical debt, expressed in effort, or to take a probabilistic approach, similar to the techniques used in risk management. This approach is hard to apply at the structural debt level (or technological gap), because you cannot assign a probability to something that you do not even suspect exists.

J. Technical Debt should not be Completely Eliminated

As a consequence of the points above, it is illusory or even wrong to attempt to eliminate all technical debt. It would be a waste of time and effort to clean up code that is correct (defect free) but badly organized, if no one will ever change it. Therefore although static analysis tools can detect all forms of bad smells, the designers must use judgment to decide which of these must be repaid. And on-going development is likely to continuously introduce more technical debt in a system.

Some authors, such as Gat and his colleagues of the Cutter Consortium [8] or Nugroho [24] have defined technical debt as the cost to improve software quality to an ideal level, but not only is this ideal level hard to define, we've seen above that while all is potential debt, not all is effective debt.

K. Technical Debt should not be treated in Isolation

Technical debt should not be treated in isolation from adding new features, fixing defects, or completing unfinished software development life-cycle activities even though they are not included in the definition. The challenge is to express software development activities in a unified perspective.

L. Technical Debt can be a Wise Investment

There is a positive aspect to technical debt, when one looks at it from the perspective of an investment [9]. Although large intentional structural or architectural debt was not what Cunningham had in mind when he proposed the metaphor, we know that these deliberate debts can considerably speed up time-to-market, allow an organization to put its code in the hands of its end-users earlier, get feedback, and evolve it. For startup ventures, it is key to preserve capital in early stages. The major issue is to clearly identify the corresponding debt, and plan on its repayment. Such debt must be part of the release planning strategy, at the same level as defects, or new features. Failure to do so is what leads some software development effort to situations where all development is crippled.

3. CONCLUSIONS AND AREAS OF INVESTIGATION

From our interactions with industry practitioners over the last four years, we have now a better understanding of the concept of technical debt, and how to approach it, both from a theoretical and a practical perspective. A unifying perspective is emerging of technical debt as the invisible results of past decisions about software that affect its future. The affect can be negative in the form of poorly managed risks but if properly managed can be seen in a positive light to add value in the form of deferred investment opportunities.

At the same time, even if we slowly converge to a better, crisper definition of technical debt that does not include all software development ills, and as we better understand the boundaries of the metaphor, there are numerous additional challenges ahead. At the top of people's lists are how to measure technical debt and how to include value in the technical debt landscape. Further research questions, aimed to improve the way technical debt will be managed by practitioners, include:

- Is quantification of technical debt possible or will well-grounded sound rules of thumb or frameworks suffice?
- Can progress be made measuring relative improvements (a system of particular type) rather than absolute values?
- How do you know when technical debt is a big problem, when is it on you or about to happen, what are the leading indicators?
- When does software quality become a technical debt issue?
- Business and technical actors may be taking different payback strategies, what are the remediation strategies for debt repayments taking into account both perspectives?
- Can technical debt be used to look at projects as a whole and determine how to focus from a risk perspective?

As evidence accumulates that organizations are developing practices and using research results, there is growing interest in identifying baseline practices and determining what organizations are doing differently to address problems in their context. Finally, there is interest in building on the results of initial interviews to construct a survey as a mechanism to collect and anonymize stories as well as sharing data sets to enable replication of existing results and validation of new studies.

4. ACKNOWLEDGMENTS

We are grateful to the many participants in the technical debt workshops organized over the last four years, and the contributors to the IEEE Software special issue on Technical Debt, as well as our consulting customers for their contributions.

5. DISCLAIMER

The views and conclusions contained in this document are solely those of the individual creator(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

[1] J. Bishop. Managing Technical Debt Panel, International Conference on Software Engineering, 2013.

[2] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. B. Seaman, K. Sullivan, and N. Zazworka, "Managing Technical Debt in Software-Intensive Systems," in Future of Software Engineering Research (FoSER 2010) Workshop, part of FSE 2010. Santa Fe, New Mexico, USA: ACM, 2010, pp. 47-52

[3] W. Cunningham, "The WyCash Portfolio Management System" in OOPSLA'92. Vancouver, 1992.

[4] B. Curtis, J. Sappidi, and A. Szynkarski, "Estimating the Principal of an Application's Technical Debt," IEEE Software, vol. 29(6), 2012.

[5] N. A. Ernst, "On the role of requirements in understanding and managing technical debt," in Managing Technical Debt (MTD), 2012 Third International Workshop on, 2012, pp. 61-64.

[6] 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 Proceedings of the 4th Workshop on Managing Technical Debt, at ICSE 2013, P. Kruchten, I. Ozkaya, and R. Nord, Eds., IEEE, 2013, pp. 16-19.

[7] M. Fowler, "Technical Debt," 2009 http://martinfowler.com/bliki/TechnicalDebt.html

[8] I. Gat, Ed. How to settle your technical debt--a manager's guide. Arlington, MA: Cutter Consortium, 2010.

[9] Y. Guo and C. Seaman, "A portfolio approach to technical debt management," in Proceedings of the 2nd Workshop on Managing Technical Debt, MTD '11. New York, NY, USA: ACM, 2011, pp. 31--34.

[10] C. Izurieta, A. Vetro, N. Zazworka, Y. Cai, C. Seaman, and F. Shull, "Organizing the technical debt landscape," in Managing Technical Debt (MTD), 2012 Third International Workshop on, 2012, pp. 23-26.

[11] T. Klinger, P. Tarr, P. Wagstrom, and C. Williams, "An enterprise perspective on technical debt," in Proceedings of the 2nd Workshop on Managing Technical Debt, MTD '11. New York, NY, USA: ACM, 2011, pp. 35--38.

[12] P. Kruchten, R.L. Nord, I. Ozkaya, "Technical Debt: From Metaphor to Theory and Practice," IEEE Software, vol. 29(6), pp. 18-21, 2012.

[13] P. Kruchten, R. Nord, I. Ozkaya, and J. Visser, "Technical Debt in Software Development: from Metaphor to Theory--Report on the Third International Workshop on Managing Technical Debt, held at ICSE 2012 " ACM SIGSOFT Software Engineering Notes, vol. 37(5), pp. 36-38, 2012.

[14] P. Kruchten, R.L. Nord, I. Ozkaya, "4th international workshop on managing technical debt (MTD 2013)," in Proceedings of the 2013 International Conference on Software Engineering, 2013.

[15] G. Lakoff and M. Johnson, Metaphors we live by. Chicago: The University of Chicago Press, 1980.

[16] J.-L., Letouzey, "The SQALE method for evaluating Technical Debt," in Managing Technical Debt (MTD), 2012 Third International Workshop on , pp.31-36, June 2012.

[17] E. Lim, N. Taksande, and C. B. Seaman, "A Balancing Act: What Software Practitioners Have to Say about Technical Debt," IEEE Software, vol. 29(6), 2012..

[18] S. McConnell, "Technical debt," 2007. http://blogs.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx

[19] S. McConnell, "Managing technical debt (slides)," in Workshop on Managing Technical Debt (part of ICSE 2013): IEEE, 2013.

[20] J. D. McGregor, J. Y. Monteith, and J. Zhang, "Technical debt aggregation in ecosystems," in Managing Technical Debt (MTD), 2012 Third International Workshop on, 2012, pp. 27-30.

[21] R. Mo, J. Garcia, Y. Cai, and N. Medvidovic, "Mapping Architectural Decay Instances to Dependency Models," in Proceedings of the 4th Workshop on Managing Technical Debt, at ICSE 2013, P. Kruchten, I. Ozkaya, and R. Nord, Eds., IEEE, 2013, pp. 39--46.

[22] J. Y. Monteith and J. D. McGregor, "Exploring Software Supply Chains From a Technical Debt Perspective," in Proceedings of the 4th Workshop on Managing Technical Debt, at ICSE 2013, P. Kruchten, I. Ozkaya, and R. Nord, Eds., IEEE, 2013, pp. 32--38.

[23] J. D. Morgenthaler, M. Gridnev, R. Sauciuc, and S. Bhansali, "Searching for build debt: Experiences managing technical debt at Google," in Managing Technical Debt (MTD), 2012 Third International Workshop on, 2012, pp. 1-6.

[24] A. Nugroho, J. Visser, and T. Kuipers, "An Empirical Model of Technical Debt and Interest," in Second international workshop on managing technical debt (part of ICSE 2011), I. Ozkaya, P. Kruchten, R. Nord, and N. Brown, Eds. Honolulu, HI, USA: ACM, 2011, pp. 1-8.

[25] 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 Proceedings of the 4th Workshop on Managing Technical Debt, at ICSE 2013, P. Kruchten, I. Ozkaya, and R. Nord, Eds., IEEE, 2013, pp. 28--31.

[26] J. Rothman, "An Incremental Technique to Pay Off Testing Technical Debt," 2006. http://www.jrothman.com/2006/01/an-incremental-technique-to-pay-off-testing-technical-debt/

[27] K. Schmid, "On the Limits of the Technical Debt Metaphor," in Proceedings of the 4th Workshop on Managing Technical Debt, at ICSE 2013, P. Kruchten, I. Ozkaya, and R. Nord, Eds., IEEE, 2013, pp. 63-66.

[28] K. Schmid, "A Formal Approach to Technical Debt Decision Making," in Proceedings of the Conference on Quality of Software Architecture QoSA'2013, Vancouver, 2013, ACM.

[29] C. A. Siebra, G. S. Tonin, F. Q. B. da Silva, R. G. Oliveira, L. C. Antonio, R. C. G. Miranda, and A. L. M. Santos, "Managing technical debt in practice: An industrial report," in Empirical Software Engineering and Measurement (ESEM), 2012 ACM-IEEE International Symposium on, 2012, pp. 247-250.

[30] R. O. Spinola, N. Zazworka, A. Vetrò, C. B. Seaman, and F. Shull, "Investigating Technical Debt Folklore: Shedding Some Light on Technical Debt Opinion," in Proceedings of the 4th Workshop on Managing Technical Debt, at ICSE 2013, P. Kruchten, I. Ozkaya, and R. Nord, Eds., IEEE, 2013.

[31] D.J. Sturtevant, "System Design and the Cost of Architectural Complexity," Massachusetts Institute of Technology, February 2013.

[32] P. Wang, J. Yang, L. Tan, R. Kroeger, and J. D. Morgenthaler, "Generating Precise Dependencies for Large Software," in Proceedings of the 4th Workshop on Managing Technical Debt, at ICSE 2013, P. Kruchten, I. Ozkaya, and R. Nord, Eds., IEEE, 2013, pp. 47--50.