An Enterprise Perspective on Technical Debt

Tim Klinger, Peri Tarr, Patrick Wagstrom, Clay Williams
IBM Thomas J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532 USA
{tklinger, tarr, pwagstro, clayw}@us.ibm.com

ABSTRACT

Technical debt is a term that has been used to describe the increased cost of changing or maintaining a system due to expedient shortcuts taken during its development. Much of the research on technical debt has focused on decisions made by project architects and individual developers who choose to trade off short-term gain for a longer-term cost. However, in the context of enterprise software development, such a model may be too narrow. We explore the premise that technical debt within the enterprise should be viewed as a tool similar to financial leverage, allowing the organization to incur debt to pursue options that it couldn't otherwise afford. We test this premise by interviewing a set of experienced architects to understand how decisions to acquire technical debt are made within an enterprise, and to what extent the acquisition of technical debt provides leverage. We find that in many cases, the decision to acquire technical debt is not made by technical architects, but rather by non-technical stakeholders who cause the project to acquire new technical debt or discover existing technical debt that wasn't previously visible. We conclude with some preliminary observations and recommendations for organizations to better manage technical debt in the presence of some enterprise-scale circumstances.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management—life cycle, productivity

General Terms
Human Factors , Management

Keywords
Extended Stakeholders, Leverage, Technical Debt

1. INTRODUCTION

Technical debt has been framed previously [2][3] as a tradeoff between implementing some piece of software in a robust and mature way (the "right" way) and taking a shortcut which may provide short term benefits, but which has long term effects that may impede evolution and maintainability. Such a proposition frames technical debt in a primarily negative light by assuming that there is some fixed "right" way to proceed, as well as by emphasizing the cost of making the expedient choice over the benefit that may accrue because of that choice.

But is the problem or the tradeoff really either this straightforward or this localized, particularly in enterprises that deliver software products for a living? From our vantage point as researchers in a large, software-producing enterprise, we have seen a variety of things that make us suspect that the sources of technical debt, the motivation for incurring it, and the management of it, are considerably more complex than simple tradeoffs made by technical architects. While technical architects bear responsibility for the technical features of the code, numerous additional stakeholders are active in ensuring the product is a success. These individuals cover diverse technical and non-technical roles such as product testing, brand strategy, legal, marketing, and more [5]. Furthermore, in an enterprise context, decisions about products or projects are rarely made in a vacuum. Instead, the portfolio of development activities is managed as a collection, meaning that decisions for one product to take on debt may be made in order to realize an important opportunity for the portfolio as a whole.

When considering the larger set of project stakeholders and the perspective of the portfolio of software investments, acquiring technical debt may offer strategic benefits to an enterprise. In this case, the debt is analogous to using leverage for long-term investment. For example, the ability to meet a critical customer delivery date may necessitate technical shortcuts, but meeting the delivery date may ensure that the product thrives and grows in the marketplace, something that may be in jeopardy if the delivery date is missed. After delivery of the product, the enterprise may have to repay some of the technical debt, or, similar to a financial option, they may not be required to repay the debt if the product or enterprise evolves in a different direction.

To begin exploring this perspective, we conducted a limited ethnographic study involving four technical architects at IBM. Although the sample size was small and the architects had different backgrounds and responsibilities, we heard remarkably similar perspectives from them, and we believe the results provide some interesting and useful preliminary insights into how enterprise organizations view, evaluate, and leverage technical debt. We present our interviews and general findings Section 2, then describe recommendations for enterprises and software engineering researchers alike in Section 3.

2. INTERVIEWS

To understand the problem of technical debt, we conducted a series of interviews with technical architects inside IBM. The interviews were designed to elicit general responses about technical debt, and also to have the project architects hone in on a smaller number of specific instances when their projects had incurred technical debt. For these specific instances, we drilled deeper to address the nature of the debt incurred, the reason for incurring the debt, the individuals responsible for incurring the debt, and the ramifications of the debt.

2.1 Interview Subjects and Protocol

Although we have a small number of subjects in this preliminary work, we have attempted to draw a sample from a wide range of projects within the company. The four interview subjects were as follows:

Subject A: An architect for a relatively new product that fits into an existing product line at IBM. This product was required to integrate with existing and evolving technologies.

Subject B: An experienced architect with exposure to many different product lines. This architect was charged with creating and managing a team to shepherd a recently acquired, but mature software product toward integration with existing tools. Subject B was an IBM employee when the product was acquired.

Subject C: An architect for an established product acquired by IBM (different from the product managed by Subject B). The architect worked with the product before it was acquired and successfully led an effort to integrate the acquired product with existing IBM products.

Subject D: A very experienced architect who has worked on a variety of extremely large-scale systems both inside and outside of IBM.

Interviews were conducted by phone by members of our team with one researcher designated as the lead interviewer, and 2-3 additional researchers serving to take notes and ask additional questions. All interviews, with the exception of Subject B, were recorded and extensive notes were taken to supplement the recordings. Interviews were approximately 1 hour in length.

The interviews were conducted in a semi-structured manner. After preliminary background information was shared, interviewees were asked to recall specific instances when their project incurred technical debt. They were then asked a series of questions about the nature of the technical debt and the context surrounding the technical debt, such as which stakeholders were involved in making the decision to incur the debt, what benefits were obtained by incurring the technical debt, who received those benefits, how (or if) the decision to incur the debt was recorded, and if any drivers for the decision-making process were quantified. For the cases where the debt had been repaid, the interviewees were asked about the decision to repay the debt, while those that had not repaid the debt were asked about the prospect of repaying the debt, the concrete costs of continuing to hold the debt, and who "paid" for it.

2.2 Interview Summaries

Subject A was faced with a choice between designing a product to deliver a required capability by supporting an older, existing API that would soon be obsolete, or by supporting a newer API that was likely to be the choice for future releases of the product. Either API would be suitable for Subject A's current needs, but only the newer API would also suit his future needs. The subject described feeling constrained to support the older API by two factors. First, supporting the new API required cooperation from a partner team on which they were dependent. The release manager from the partner team decided that investing in the new API at that time was lower priority than other of the partner team's goals, such as meeting the advertised ship date. Second, a number of other existing products with which this product needed to integrate were still using the old API.

In this case, the choice by the partner team not to collaborate on supporting new API induced technical debt for Subject A's team. Although the issues were debated, and Subject A noted that an important cost of this decision could be customer dissatisfaction, Subject A felt that there were barriers to effectively estimating and communicating the nature and magnitude of the technical debt in terms that made sense to the non-technical stakeholders, who might have acted to induce a different decision (e.g., by changing the partner team's ship date).

Subject B was brought on to work on a new release of an existing product that IBM recently acquired. Prior to the acquisition, this product's customer-specific requirements had caused some other requirements to be prioritized fairly low. After the acquisition by IBM, however, many of these lower-priority requirements became considerably more important, as the product was positioned in a larger software portfolio. For example, IBM's global presence and wide distribution of software caused requirements regarding accessibility, globalization, and performance to gain prominence. At the same time, the team also had commitments for new features. In this case, the acquisition induced technical debt in the product because of the increased priority of architecturally significant requirements for which the product had not been designed.

Subject C provided several additional examples of the need to understand the hidden costs of technical debt. After his company was acquired by IBM, he had responsibility for helping to integrate his product into IBM's portfolio. In particular, his product was complementary to an existing IBM product, and management decided that his product should support a connection to the existing product. Subject C agreed with his IBM product colleagues that a particular, but as-yet unimplemented, integration approach would be most appropriate and most architecturally consistent with the portfolio. However, the team supporting the existing product had firm commitments to deliver features to their customer base and did not have the resources to accommodate both the development of those features as well as the new APIs needed to support the new integration approach. The decision was made to forgo the development of the new approach and to use an existing integration mechanism that was suboptimal but "good enough."

Subject C needed to ensure that his product's architecture would support the post-acquisition requirements, while the existing product team needed to deliver features which they had promised. The choice to integrate using the old API did serve the needs of the existing product's management, but it induced technical debt for both the technical team and the acquired product since both teams will eventually have to abandon the stop-gap integration and reimplement their code to the new API.

Our final interview was with Subject D, who discussed a variety of projects across industry sectors both inside and outside of IBM. In one example, he described a widely used non-IBM desktop product that accrued a significant amount of technical debt while simultaneously becoming very successful over a series of releases. Product managers realized that the existing code base was increasingly impairing their ability to continue to deliver new features. A team was put together to rewrite the product from scratch to try to pay down technical debt. Unfortunately, the team was unable to anticipate some of the ways their customers had been using the software, or how many customers had come to rely on the peculiarities of the product's original architecture and features. Without a clear understanding of these hidden customer requirements it was impossible for the team to remediate all of the product's debt. This is a case where the cost of paying down the debt wasn't limited to the cost of the technical activities within the team. It also involved a financial risk of losing customers who might abandon the product after it ceased to function in the way they required. In essence, the success of the product induced an unacceptably high cost for the team trying to pay off the technical debt by replacing the product.

In the end, the company decided not to rewrite the product. This illustrates a "challenge of the collective" in managing technical debt: making this type of debt visible and assessing it is a difficult task because it involves a deep understanding of both the technical artifacts and the ecosystems that develop around those artifacts.

Another example from Subject D is that of a successful web-based startup company. This company routinely accrues technical debt because developers operate in relative isolation, without a unifying architecture and set of practices. As a result, their product contains many code clones, which represent a form of technical debt that increase costs because of the need to maintain these redundancies. However, the company makes sufficient profit with a relatively small code base that they are not perturbed by the debt. The small code base also means that they can periodically rewrite the entire system to pay off their technical debt.

In this case, the company apparently made the assessment that the cost of acquiring and then paying off this debt repeatedly is relatively minor compared to the profit the firm is generating. Whether this assessment is right or wrong from a business perspective, it points to the fact that technical debt is only one factor among many in managing a successful business. Decisions concerning that debt need to be viewed in the larger enterprise context and weighed against the costs and benefits of remediating that debt.

2.3 Common Findings across Subjects

Induced and unintentional debt are challenges. Although all four subjects discussed the reality of technical debt that is intentionally incurred [4] by product architects, they also highlighted two additional sources of debt as potentially more debilitating. One is debt that is induced by other stakeholders in the project or across the portfolio. This includes the imposed requirement to meet a specific release date, even if quality or other architectural properties suffer, as well as cascaded impact from decisions made on other projects on which a given project depends. This cascading effect may happen along interfaces between development groups or even temporally across the ecosystems that come to depend on the decisions from one release to another. The second category involves unintentional debt that the architects and other stakeholders did not actively incur, but which was caused by situations such as acquisition, new alignment requirements, or changes in the market ecosystem. In general, this category results from the imposition of new, unanticipated, and architecturally significant requirements. In the view of the technical architects we interviewed, the non-intentional debt typically was much more problematic than the intentional debt.

Decisions are managed in an ad hoc manner. In all four interviews, we learned that the explicit management and tracking of debt-inducing decisions was often informal and ad hoc. When faced with a choice between two technical options, the decision was often made without any degree of formalization or attempt to quantify the impact of the decisions. While some thought was given toward the potential cost of paying back the debt within technical teams, such explorations at the enterprise level appear to be exceptional rather than routine. Most details regarding decisions were only available through "tribal memory," which is an unreliable source of historical information due to forgetfulness and the attrition of team members.

Stakeholders with different types of concerns lack effective ways to communicate and reason about technical debt. In the enterprise context, decisions that incurred technical debt were often made by stakeholders who did not fully comprehend the ramifications of that debt, especially when the debt was induced for other stakeholders. In particular, there was a prominent communication gap between stakeholders whose primary concerns were financial or customer related and the technical stakeholders. The technical architects indicated that there was neither a channel nor a common vocabulary to express the costs of incurring a technical debt to non-technical stakeholders.

Managing the communication and negotiation for decisions involving these multiple types of stakeholders may be difficult without a careful (if not necessarily quantitative) assessment of a product's technical debt. Such an assessment is complicated by the fact that technical debt depends heavily on many dynamic and challenging factors that may change over time, including customer requirements, dependencies between products and teams, ecosystem changes, and mergers and acquisitions.

3. CONCLUSIONS AND FUTURE WORK

This paper presents results and insights from a set of preliminary interviews with product architects in a software delivery enterprise. The interview structure was developed to understand the process of reasoning about, incurring and managing technical debt. Our concerns included whether or not debt is viewed as leverage for other actions; if decisions to take on debt are explicit or implicit; where such decisions originate in enterprises; how visible the ramifications of the debt are; and how decisions are made to pay down debt.

Even though our study was small, we believe that it has shed light on some important aspects of technical debt for both enterprises and software engineering researchers. On the broad level for enterprises we make the following three observations regarding technical debt:

Collectives matter, for both products and people: Every successful enterprise, including those that deliver software, makes decisions that optimize benefits across their portfolio of investments. These decisions may cascade to have negative impact on individual projects. In other words, for the sake of optimizing globally, some projects may suffer locally. However, enterprises often lack the global view and communication channels necessary to properly optimize the complexities of technical debt. Therefore, this results in many locally optimized decisions that may not correspond to the global optimum for the enterprise.

Another collective effect is that portfolios of products in the same domain can suffer from architectural issues regarding where capabilities belong. This challenge frequently arises when the customer sets for products in a portfolio are partially disjoint, leading to redundancy of capabilities. This can be further aggravated by acquisitions.

Additionally, every product and portfolio has many stakeholders, and these stakeholders often have competing goals and "win" conditions [1]. Decisions are often made that best optimize across this diversity of "win" conditions, but these appear to be suboptimal from a purely technical perspective. Moreover, since these stakeholders have the ability to affect a variety of decisions made about a project, they may have the ability to influence not only whether or not a project acquires technical debt, but also the ability of the project to pay back the technical debt.

The ability to assess debt matters: Decisions regarding technical debt were rarely, if ever, quantified, but quantification is a critical step before such decisions can be properly monetized in the enterprise context. However, quantification will be very difficult to do, as technical debt is not absolute—it is relative to a set of goals, requirements, stakeholders, and sometimes, the ecosystem in which the product lives. If any of these things change, the debt may change as well. For example, a decision to defer support for a particular standard in an API may incur a technical debt, but if the standard is superseded subsequently, this debt disappears. Yet in our set of interviews, skilled technical architects were able to reason cogently about issues they did not quantify. However, because there was no quantitative analysis, there was no data to record and pass to other team members for future decision analysis and review. Tracking architectural and other decisions is a necessary step to being able to assess technical debt and to trace from business decisions to their architectural implications to understand impact of change.

Bridges across enterprise gaps matter: Diverse software-delivery enterprises often have significant "gaps" between business, operational, and technical stakeholders [5]. As a result of these "gaps," potential problems may go undetected until they wreak havoc. Our interviews found that these gaps were also relevant to technical debt. The individuals choosing to incur technical debt were often different from those responsible for servicing the debt. Organizational processes developed to carry information across enterprise gaps weren't designed with technical debt as a concern. The end result of this is that technical architects felt they didn't have a feasible method to communicate and provide feedback to the non-technical stakeholders who were causing the debt to be incurred. Furthermore, the lack of common methods to quantify and/or monetize debt makes it difficult to ensure that all parties fully understand the ramifications of the decision to acquire debt. To ensure that organizations are able to thrive and properly manage their technical debt, there needs to be not only a common approach to communicate about the technical debt, but also mechanisms in the process to provide feedback on decisions to incur or pay off technical debt.

In addition to the observations relevant to software engineering enterprises, our study has yielded valuable information for software engineering researchers. We highlight four issues for further study.

First, because enterprise technical debt occurs in the context of a larger portfolio, there is an opportunity to apply concepts from investment leverage and / or options theory to the analysis of technical debt. Second, because of the diverse stakeholders who need to be involved in technical debt management and the lack of decision support for this activity, there is an opportunity to apply concepts from decision science to the process of managing technical debt. Third, as noted above, technical debt is often invisible or impressionistic, especially to non-technical stakeholders. Finding ways to quantify debt in ways meaningful to specific stakeholders is essential. We believe that relative or bucketed measurements (e.g. "debt points" or debt ratings) are more likely to be feasible in the complex enterprise environment than are absolute measures of technical debt. Finally, we described the importance of the occurrence of unintentional debt. This usually arises due to dynamic forces beyond the control of the technical team, and often beyond the scope of the enterprise. We plan to further characterize this form of debt and explore mechanisms to help organizations detect and deal with it.

We see three phases of future work to move forward on these preliminary observations. First, we need to conduct more interviews across a broader set of enterprise stakeholders to validate and expand upon these findings. The stakeholders should be carefully chosen to ensure we both sample across an array of projects and portfolios, but also sample different stakeholders involved with the same product. Next, we need to develop techniques to assess technical debt in principled ways, including approaches to converting or mapping it into perspectives that make sense to the variety of stakeholders who can be involved in incurring or paying down debt. Finally, we want to pilot these techniques in projects and portfolios and explore the changes that occur in the organization as a result of increased visibility and communication.

4. ACKNOWLEDGMENTS

We thank the interviewees for their willingness to participate in this research. Thanks also to Peter Santhanam and Mark Wegman for their helpful comments and suggestions.

5. REFERENCES

[1] Barry W. Boehm and Rony Ross. Theory-W Software Project Management Principles and Examples. *IEEE Transactions on Software Engineering* (July 1989).

[2] Nanette Brown, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim, Alan MacCormack, Robert Nord, Ipek Ozkaya, Raghvinder Sangwan, Carolyn Seaman, Kevin Sullivan, and Nico Zazworka. 2010. Managing technical debt in software-reliant systems. In *Proceedings of the FSE/SDP workshop on Future of software engineering research* (FoSER '10).

[3] Ward Cunningham. 1992. The WyCash portfolio management system. In *Addendum to the proceedings on Object-oriented programming systems, languages, and applications (Addendum)* (OOPSLA '92).

[4] Steve McConnell. 2007. *Technical Debt*. http://forums.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx

[5] Clay Williams, Patrick Wagstrom, Kate Ehrlich, Dick Gabriel, Tim Klinger, Jacquelyn Martino, and Peri Tarr. 2010. Supporting enterprise stakeholders in software projects. In *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering* (CHASE '10).