Restructuring and Refinancing Technical Debt

Raul Zablah
The Wharton School
University of Pennsylvania
Philadelphia PA 19104 USA
razablah@wharton.upenn.edu

Christian Murphy
Department of Computer and Information Science
University of Pennsylvania
Philadelphia PA 19104 USA
cdmurphy@seas.upenn.edu

***Abstract*—Given the increasing importance of software to society, the issue of technical debt is becoming more pervasive in software development. Its implications range from incurring small amounts of technical debt to speed up development - a positive - to stalling and making development no longer possible - a huge negative. In this paper, we present a framework that attempts to refine the understanding of technical debt by tracing more links to the financial metaphor, specifically focusing on the concepts of *restructuring* and *refinancing* technical debt. This paper looks at technical debt as a leverage product that is contingent upon the liquidity of the debtor. From this perspective, it is then possible to more effectively assess the incurment of technical debt and also to more effectively strategize the use of leverage in software development - accounting for the respective risks and benefits it provides.**

I. Introduction

Software development has become the foundation for modern society. The range of devices, applications, and services that are dependent on software systems has increased exponentially with time; with them, the number of individuals who utilize these systems and expect them to be reliable has also increased. Due to this dependence and high interconnectedness, any problems with the development of software will likely lead to significant problems - both operationally and financially for individuals and businesses alike. A 2013 study found that the annual cost of debugging and fixing software faults and errors had risen to approximately USD$312 billion, while the cost of designing and developing new software was also USD$312 billion [1]. Spending one dollar on fixing code for every dollar spent on the design and development of new software indicates that there is an imperative need for change within the industry, as well as an inherent need for reform and improvement in the field of software design and development.

Part of the challenge is that software is dynamic and changing by nature. As software is a realization of a set of requirements, it can be very difficult to define if there is no model that helps formalize the problem. The creation of software is in itself a way of modeling the real-world problem and then coming up with a solution for it; however, the model is invisible and intangible. Therefore, the design and development of software is inherently difficult and requires constant change [2].

Another important factor that demands consideration when designing, developing, and maintaining software is the existence of a trade-off between internal quality (e.g., software design) and external quality (e.g., functionality). This incurred trade-off is usually the source of issues later on. Given that the set of requirements will be dynamic throughout time, it is imperative to understand how this trade-off impacts the quality of the software being developed during the software development lifecycle.

The notion of "technical debt" [3] was introduced to provide an understanding of the potential cost of favoring external quality over internal quality such that later changes to the code would be made more difficult. This analogy is well understood in the software development industry, but still there are areas in which this metaphor falls short, and the relationship to financial debt is not fully realized [4]. In this paper, we present a framework that attempts to refine the understanding of technical debt by tracing more links to financial theory, specifically focusing on the concepts of *restructuring* and *refinancing* technical debt, and describe how these concepts are related to "liquidity" within a software development project. Although the framework itself is not yet complete, here we present our initial observations and discuss future directions for this work.

II. Terminology

The following terms are defined according to the context of this paper. These are the definitions with which our framework is built and how the terms are to be interpreted semantically.

- Internal quality: the totality of characteristics of the software product from an internal view, i.e. the code as human-readable text, including analyzability (understandability and readability), changeability, stability, and testability [5].
- External quality: the totality of characteristics of the software product from an external view, i.e. the code as it executes in an environment, including functionality, reliability, usability, efficiency, and portability [5].
- Debt restructuring: method used by companies with impending debt obligations to alter the terms of the debt agreements in order to prevent default [6].
- Debt refinancing: method used by companies or individuals to revise the payment schedule for the repayment of the debt, essentially replacing an older debt agreement with a new debt agreement that offers better terms [7].

978-1-4673-7378-4/15 © 2015 IEEE

III. Proposed Framework

The main objective of our framework is to refine the understanding of the technical debt metaphor. In particular, we suggest that technical debt is a type of debt that can be refinanced or restructured, depending on the debtor's liquidity.

To understand this claim, it is necessary to understand where the debt comes from. At a high-level, technical debt is the result of the trade-off between internal and external quality of software. This trade-off is the result of a conscious decision made between the two parties involved in the development of the software. The two parties are: The Client, who is the individual or entity who communicates the requirements for the development of the software; and the Developer, who is the individual or entity who is responsible for the development of the software the Client requests.

Unlike Fowler's Technical Debt Quadrants (Fig. 1 [8]), our framework suggests that the inadvertent dimension is *not* pertinent to the analysis and refinement of the technical debt metaphor. We consider the inception of technical debt a conscious decision made by one or both of the parties involved. This suggests that at least one of the parties is aware of the advantages of incurring technical debt to improve the efficiency of development cycles, therefore acknowledging that there exists the likely possibility that debt will be incurred during development. The framework considers that the inadvertent dimension is the result of a lack of knowledge and hence not relatable to the incurring of debt financially - incurring financial debt without being aware of it is usually the result of lack of knowledge regarding terms and conditions.

This suggests that the Developer is aware of development practices - and their respective limitations - and of the Clients' constraints and requirements. Even though understanding of these need not be perfect, the Developer acknowledges their existence. Therefore the Developer actively engages in the acquisition of the technical debt.

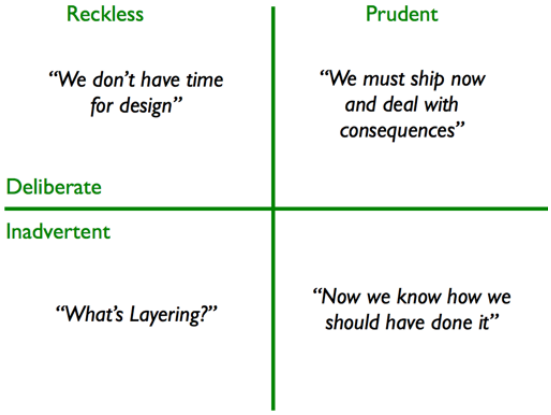

Fig. 1. Fowler's Technical Debt Quadrants. [8]

The Project Management Triangle [9] demonstrates the constraints of a software development project: resources, scope, and time. The resources for the software development case are the man-power and capital requirements in order to develop the software. This includes the developers, the necessary software and hardware platforms, and any other relevant requirements for the development of the software. The scope is the set of features, functionality, and performance of the desired software. Lastly, time is the determined time-frame attributed to the development of the software. Therefore, just as in the Project Management Triangle, it is the combined pull of these three constraints that give rise to technical debt - since there must exist some sacrifices in quality in order to satisfy the constraints [9].

The trade-off that results from the Project Management Triangle constraints is characterized by a trade-off between the internal and external quality of the software. Just as this trade-off is dynamic throughout time - given that the three constraints are present throughout the entire lifecycle of the software - so should be the solutions to this trade-off. The relationship between refactoring and technical debt is then characterized by a convex curve that compares the relationship between resources and amount of technical debt.[1] From this relationship it is easy to tell that there exists a point where the marginal benefit of the refactoring of the debt is outweighed by the marginal cost in terms of magnitude of resources utilized for it, just as stated by Brooks' Law. Hence, this implies that the optimal level of technical debt for the development of software is a non-negative value [4].

If technical debt is the resulting trade-off between internal and external quality of the software being developed, then it follows that the accumulation and sustenance of technical debt decreases the debtor's ability to meet its debt obligations - effectively reducing the debtor's liquidity. Just as with financial debt, the existence of technical debt in a software system implies an inherent risk of default due to lack of liquidity. In the case of software development, the default risk is the risk of not being able to meet any new requirement deadlines or project milestones because all efforts are devoted to paying off the interest over the principal of the debt (i.e., maintaining the software). Therefore, if a debtor is incapable of satisfying its short-term obligations due to lack of liquidity, then it follows that the debtor is in financial distress.

It is imperative to understand the difference between liquidity and solvency in the context of technical debt. Liquidity refers to the ability to deliver on short-term obligations regarding the project, while solvency refers to the ability to deliver on long-term obligations, which in this case, would be the overall project being developed. It is possible to be solvent and yet illiquid, which would imply being in financial distress as stated before. It is also possible to be liquid and insolvent, which would imply that the developer can meet short-term obligations but cannot meet the overarching long-term obligations. For the scope of this work, we are assumping that liquidity is what potentially leads to default and therefore demands a solution.

[1]The quantification of technical debt is beyond the scope of the framework, and hence we abstract the specifics and assume that all technical debt can be quantified. This assumption is addressed in Future Directions below.

Given the layout of the foundation of our framework, we now delve into the specifics of how technical debt is a type of debt that can be **restructured** or **refinanced** depending on the debtor's liquidity. We first delineate the difference between restructuring and refinancing. From the definitions of both, it follows that restructuring deals with modifications of the existing terms of the debt, whereas refinancing deals with the replacement of existing debt with a new debt obligation. While refinancing is an activity in which the debtor can proactively engage, since there is no liquidity distress, restructuring is an activity that the debtor considers only when there is liquidity distress and there is a large risk of not being able to meet the necessary obligations [10].

A. Restructuring Technical Debt

Restructuring, since it only occurs in times of distress, primarily seeks to directly affect these aspects of the debt obligation, in order to provide the debtor the ability to meet its obligations. Restructuring benefits the debtor, providing liquidity relief, but at the expense of the counter-party that provided the debt obligation. In the specific case of software development, the Developer benefits from a restructuring, at the expense of the Client. Restructuring includes:

Reduction of principal: this implies decreasing the functionality requirements of the software in order for the debtor to be able to meet expectations. For this to happen, the Client will have to renegotiate the terms of the development of the software with the Developer in order to determine which functionality aspects will have to be reconsidered for inclusion in the project. This is the last option the Client would consider for the restructuring of the debt, given that this implies the largest loss of value for the project.

Lower interest rates: interest rates are in the form of refactoring cycles needed to maintain the leveraged software. Therefore, lowering the interest rate of the debt is essentially equivalent to the extension of deadlines for immediate milestones or delaying new functionality requirements. This enables the debtor to meet the interest payments without defaulting on them. The Developer and the Client have to engage in dialogue to determine the full impact of the lower interest rate on the project, since this option will inevitably slow down development of the software. Nevertheless, this option provides both Client and Developer the most flexibility within the distressed environment, since functionality is not being sacrificed and the lower interest payments will be implemented only until the restructuring provides the Developer enough room to improve liquidity.

Any combination of these would be considered a restructuring and should be explored as possibilities by the debtor, depending on the type of liquidity distress - and the underlying causes behind it. It is important to note, however, that for the restructuring to take place it is necessary for the Developer and the Client to engage in negotiation. This could become a complicated process due to the inherent conflict of interest between the two parties. If the debtor is under distress, it is most likely that the largest compromises will have to be conceded by the Client. The most relevant and direct implication of this is the speed at which the software can be developed, as well as running the risk of incurring further debt within the restructuring itself. Also, the implication of a restructuring is that both parties have acknowledged the issues at hand, both parties are trying to improve the situation at hand, and both parties are not only accepting the situation but rather trying to relieve the debt burden in order to increase productivity.

B. Refinancing Technical Debt

Refinancing, since it is an activity the debtor proactively engages in *without* necessarily being under distress, primarily seeks to improve the terms of the existing debt obligations to reap some benefit for the debtor. Within the financial context, refinancing might take place for a multitude of reasons, with the most important being things like: taking advantage of a better interest rate; consolidating multiple debts into a single debt obligation; reducing the payment of interest per period; reducing the risk of the debt obligations; or freeing up cash-flows [11]. Through refinancing, the debtor is increasing its ability to manage its debt obligations. Applying that same logic to software development leads to the idea of refinancing of technical debt acting as a vehicle for efficient debt management. The debtor may engage in debt refinancing for the following reasons:

Taking advantage of a better interest rate: for instance, if the Developer has access to more resources to put into the development cycles. By doing this, the Developer is essentially increasing the number of refactoring cycles per development iteration, therefore improving the interest rate paid over the debt. The type of resource increase that the Developer utilizes can be of the nature of increase in the number of development hours put into the current cycle or an increase in the number of individuals involved in the development cycle. The important aspect of taking advantage of a better interest rate is that it is endogenous to the Developer. Also, through this, it is possible for the Developer to improve the existing relationship with the Client - since an improvement in the ability of the Developer to pay back debt is equivalent to an improvement of the overall project. It is also important to note that there is a limit to how many extra resources can be devoted to the refinancing, as mentioned in Brooks' Law.

Freeing up cash-flows: by improving the debtor's ability to manage the existing debt, it is possible to improve the liquidity of the debtor, therefore freeing up existing cash-flows for the project. In terms of software development, freeing up cash-flows is essentially the same as freeing up resources that could be devoted to further development cycles. The debtor can free up resources by improving the development techniques being employed, and by ensuring the existence of quality assurance and quality control practices within the development process. This aspect is characterized by the proactiveness it implies in terms of development strategies. It is in the debtor's best interest to engage in these proactive measures in order to

facilitate the management of the debt, while improving the overall project through it.

For technical debt refinancing, it is important to note that it is a proactive measure taken by the debtor to benefit and facilitate the development practices. Through these, it is possible for the Developer to improve the overall project - the refinancing activities are an excellent tool for the improvement of the debt management of the project.

IV. Future Directions

Our framework is very robust but is still being developed. Even though it delineates the basic layout of how to interpret technical debt as a type of debt that can be restructured or refinanced depending on the debtor's liquidity, it still needs to account for specific details for some of its aspects [4].

First, technical debt cannot currently be directly measured. Without a precise way of measuring the technical debt of a software system, the framework is hard to apply in practice. There are approximation methods, e.g. the quantification of resources (development hours, lines of code, etc.) put into combating technical debt, but without an accurate way of quantifying the technical debt, it is very difficult to start understanding the terms of the debt - its principal, interest payments, and maturity - and even more difficult to understand the degree up to which it affects the liquidity of the debtor.

Additionally, technical debt can be introduced by a shift in context. In our framework, technical debt is the result of a conscious decision made by the Developer. The major limitation with this assumption is that technical debt may be the result of a change in the overall environment, exogenous to any decision making process of the Developer. This presents a limitation because it is necessary to then determine how to evaluate this debt, more specifically, how to determine the terms of this new debt obligation. It is only through the understanding of the terms of the new debt that it will be possible for the Developer to determine how its liquidity is affected by it, and whether it is necessary to either restructure the debt or to engage in a refinancing of it.

These two limitations of the framework require future refinement. Even though the framework is robust enough for it to theoretically model technical debt with respect to liquidity, it works with abstractions of the quantification of debt and the degree of liquidity of the debtor.

V. Conclusion

Technical debt is a type of debt that can be restructured or refinanced, contingent upon the debtor's liquidity status. It is the product of the resulting trade-off between internal and external quality in software development. Nevertheless, it is not incurred by mistake: it is the result of a conscious decision made by the Developer. It is the result of the inherent tension between scope, resources, time - constraints that govern the quality of the product and are present in the development of any project [9].

Even though technical debt is perceived as a bad thing, it can be a wise investment to make [4]. It is a good short-run investment to speed up development, and it explains why the optimal level of technical debt within a software system is non-zero. Also, as with financial leverage, technical debt as a way of leverage is useful in the development cycle as long as its benefits are not overshadowed by the distress costs and other risks. This is why it is so imperative to understand the degree to which the technical debt affects the software system - since it is mainly reflected on the way the debtor deals with the debt. On the one hand, if the debtor understands that he is in distress, it is crucial for the debt to be restructured in order to prevent the development process from stopping. On the other hand, if the debtor understands he is not in distress, it is then possible to explore opportunities to improve the terms of the existing debt and if possible refinance the debt.

This framework mainly impacts project managers. This framework becomes a new toolset with which managers can assess their development practices and their existing software systems and make decisions to improve the cost and risk management of the projects. It empowers managers by allowing them to now strategize the way they incur debt and handle debt - important tools for anyone who is interested in capitalizing on the advantages of leverage. Once the framework is fully refined, it could be used to not only assess leveraged positions in software projects, but also help with the valuation of software products and software companies.

Acknowledgments

The authors would like to thank Yuanfang Cai for her guidance and suggestions.

References

[1] Cambridge University study states software bugs cost economy $312 billion per year. [Online]. Available: http://www.prweb.com/releases/2013/1/prweb10298185.htm

[2] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," in *Proceedings of the Conference on the Future of Software Engineering*. ACM, 2000, pp. 35–46.

[3] W. Cunningham, "The WyCash portfolio management system," in *Addendum to the proceedings on Object-oriented programming systems, languages, and applications*, 1992, pp. 29–32.

[4] 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," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 5, pp. 51–54, September 2013.

[5] H.-W. Jung, S.-G. Kim, and C.-S. Chung, "Measuring software product quality: A survey of ISO/IEC 9126," *IEEE Software*, vol. 21, no. 5, pp. 88–92, September/October 2004.

[6] Debt restructuring. [Online]. Available: http://www.investopedia.com/terms/d/debtrestructuring.asp

[7] Refinance. [Online]. Available: http://www.investopedia.com/terms/r/refinance.asp

[8] M. Fowler. Technical debt quadrant. [Online]. Available: http://martinfowler.com/bliki/TechnicalDebtQuadrant.html

[9] R. Atkinson, "Project management: cost, time and quality, two best guesses and a phenomenon, it's time to accept other success criteria," *International Journal of Project Management*, vol. 17, no. 6, pp. 337–342, December 1999.

[10] J. Warren. Debt restructuringis it a simple refinancing or troubled debt restructuring? [Online]. Available: https://www.stlouisfed.org/Publications/Central-Banker/Winter-2009/Debt-RestructuringIs-It-a-Simple-Refinancing-or-Troubled-Debt-Restructuring

[11] When would a corporation want to refinance its debt? [Online]. Available: http://www.investopedia.com/ask/answer/07/corporaterefinance.asp