Using Technical Debt Data in Decision Making: Potential Decision Approaches

Carolyn Seaman, Yuepu Guo
Information Systems Department
UMBC
Baltimore, MD, USA
{cseaman, yuepu.guo}@umbc.edu

Clemente Izurieta
Dept. of Computer Science
Montana State University
Bozeman, MT, USA
clemente.izurieta@cs.montana.edu

Yuanfang Cai
Dept. of Computer Science
Drexel University
Philadelphia, PA, USA
yfcai@cs.drexel.edu

Nico Zazworka, Forrest Shull
Fraunhofer CESE
College Park, MD, USA
{nzazworka, fshull}@fc-md.umd.edu

Antonio Vetrò
Automatics and Informatics Dept.
Politecnico di Torino
Torino, Italy
antonio.vetro@polito.it

***Abstract*—The management of technical debt ultimately requires decision making – about incurring, paying off, or deferring technical debt instances. This position paper discusses several existing approaches to complex decision making, and suggests that exploring their applicability to technical debt decision making would be a worthwhile subject for further research.**

***Keywords*-technical debt; decision making; cost-benefit analysis; portfolio management; Analytic Hierarchy Process; options**

I. INTRODUCTION

The technical debt metaphor describes a situation in which developers accept compromises in one dimension (e.g. maintainability) to meet an urgent demand in another dimension (e.g. delivering a release on time) [1]. These compromises incur a "debt" in the software that should be repaid to restore the health of the system in the future and to avoid "interest" in the form of decreasing maintainability.

The current state-of-the-practice in software development and maintenance involves difficult decisions about prioritizing tasks under tight resource constraints. These decisions often focus on maintenance changes that have immediate impact on the software's functional capabilities (i.e. enhancements and defects), and thus on the financial viability of the product. What often does not get incorporated into the decision making process are long term maintenance tradeoffs, such as technical debt. This is understandable, as technical debt, when introduced, is often less visible to decision makers, and does not have an immediate impact on functional capabilities and the customer. However, there are many examples of software development projects in which inattention to monitoring technical debt has resulted in unexpectedly large cost overruns [2], severe quality issues, inability to add new features without breaking existing features, and even the premature loss of a system. Therefore, technical debt information should be incorporated into decision making.

Including technical debt as a decision factor requires information about what technical debt items exist in the system, where they are, their magnitude and probable past and future impact. This information can come from technical debt identification techniques (e.g. source code analysis tools) applied to the maintained system. Other required information includes estimates of the costs and benefits of paying off the identified technical debt (i.e. the estimated principal and interest on each technical debt item). Gathering this information is non-trivial.

In this position paper, we focus on the next step in the process. Once this technical debt information is gathered, decision techniques and approaches are needed to support evaluating the tradeoffs between proposed enhancements, corrective maintenance, and the payment of technical debt items. Although many decision approaches have been proposed and widely used in various domains, few of them have been applied to technical debt management. In our previous work [3], we proposed a cost-benefit analysis approach to support decision making in release planning. Inspired by the debt metaphor, borrowing from the finance domain, we have also started to examine other investment decision approaches for prioritization of software maintenance tasks. In addition, we have also considered a classic approach from the decision science domain. Although these decision models approach the problem in different ways, they are similar in the sense of supporting decision making in a scientific manner. We believe application of these decision approaches would improve the quality of the decision making process, and would like to propose, as part of a technical debt research agenda, the investigation of the strengths and weaknesses of these approaches.

II. DECISION APPROACHES

From the perspective of technical debt management, the goal of identifying and measuring technical debt is to facilitate decision making. Thus, once instances of technical debt have been identified and coarse-grained estimates of their properties have been made, this information can then be used to help decide which technical debt items should be paid off next. In this section we propose four decision approaches that can be used to incorporate technical debt information into decisions about release planning; they are: Simple Cost-Benefit Analysis,

Analytic Hierarchical Process (AHP), Portfolio Management Model and Options.

A. Simple Cost-Benefit Analysis

In our previous work [2, 3] we proposed an initial technical debt management mechanism, which is a framework within which to develop technical debt theory. In the context of release planning decision support, it provides a baseline decision approach that can be used as a basis of comparison for the other three approaches. The simple cost-benefit analysis approach to technical debt management centers on a "technical debt list." The list contains technical debt "items," each of which represents a task that was left undone, but that runs a risk of causing future problems if not completed. General examples of technical debt items include modules that need refactoring, testing that needs to be done, documentation (including comments) that needs to be written or updated, architectural compliance issues, known latent defects that need to be removed, etc.

Each item includes a description of where in the system the debt item is and why that task needs to be done, and estimates of the principal and the interest. As with financial debt, the ***principal*** refers to the effort required to complete the task. The interest is composed of two parts. The ***interest probability*** is the probability that the debt, if not repaid, will make other work more expensive over a given period of time or a release. The ***interest amount*** is an estimate of the amount of extra work that will be needed if this debt item is not repaid. For example, the interest probability for a design debt item (e.g. a modularity violation found in the source code [4]) is the probability that that part of the system will have to be modified in future releases. Since the probability varies with different time frames, a time element must be attached to the probability. For example, a module may have a 60% probability of being changed over the next year, but a much lower probability of being changed in the next month. The interest amount would be the extra work required to modify the code containing the modularity violation, over and above what it would have cost to modify the same code without the violation.

When a technical debt item is created, all three metrics (principal, interest probability, and interest amount) are assigned values of high, medium, or low. These coarse-grained estimates are sufficient for tracking the technical debt items and making preliminary decisions. More detailed estimates are not made until later, when they are required to do fine-grained planning and when more information is available upon which to base the estimates. When more precise estimates are needed, estimation procedures are followed for each metric, based on the type of technical debt item.

In our previous work [5] we showed how to estimate principal and interest in a realistic software project using one existing technical debt detection technique: code smells (in particular god classes). In Fig. 1 we show a Cost/Benefit matrix for nine god classes in an industrial software application. Interest (y-axis) is estimated using historical data on effort to fix defects and maintain each of the god classes; principal (x-axis) is calculated by estimating the cost to refactor the god class. One strategy for paying off the debt associated with god classes is to choose classes starting with the upper left corner of the chart, and then moving down and to the right. The classes in the upper left corner (e.g. GodClass8 and GodClass7) are most likely to have a greater effect on future maintenance (high interest), but are relatively cheap to pay off (low principal).

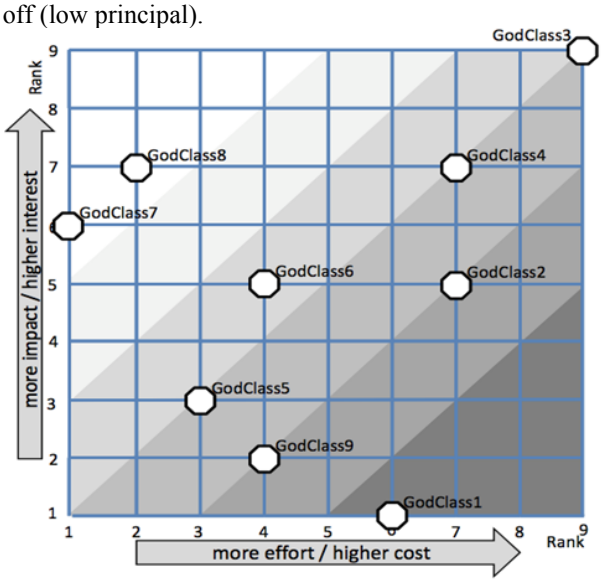

Figure 1. Cost/Benefit matrix for nine god classes in an industrial software application.

The reliability and usefulness of the decision-making support that this approach provides increase with the quantity and quality of the historical data available. However, any amount of data will provide more support for decision making than is currently available to project personnel. Thus, the approach is useful even with limited amounts of data, or even no data. Expert opinion, or a combination of opinion and limited data, can be substituted for any of the inputs to the approach that call for historical data. A benefit of the approach is that it is able to utilize software metrics data, to the extent that it is available, further motivating the collection of this data.

B. Analytic Hierarchy Process

The Analytic Hierarchy Process (AHP) originates from the decision sciences literature and is supported by a significant body of scholarly research and available tools. It has been shown to be effective both in multi-criteria and group decision making scenarios [6, 7]. It provides a simple method for structuring a problem, comparing alternatives with regard to specified criteria, and determining an overall ranking for each alternative. In group decision making, AHP also provides a structured method for combining each group member's judgments [6]. AHP structures the decision making process by constructing a hierarchy of decision criteria, and defining a method of pair-wise comparison under the covering

criteria (those at the leaves of the hierarchy of criteria) to calculate a priority vector representing the relative priority of all alternatives under consideration.

Applying AHP involves building a criteria hierarchy (which includes all criteria, quantitative and qualitative, objective and subjective, that are relevant to the decision being made), assigning weights and scales to the criteria, and finally performing a series of pair wise comparisons between the alternatives against the various criteria. Many of the pair wise comparisons between criteria that are objective and quantitative can be automated, thus reducing the burden on the human decision makers. When applying AHP to the technical debt management problem, the decision alternatives would be the various identified instances of technical debt currently existing in the system, and the outcome of the process would be a prioritized ranking of these items, indicating which should be paid off first. The criteria are related to notions of principal and interest, some of which are uncertain and subjective. Thus, in any approach to decision making about technical debt, some human intervention is required to provide information that cannot be reliably measured. We hypothesize that AHP will provide an efficient and effective mechanism for structuring and minimizing the human input into the process.

C. Portfolio Approach

In the finance domain, a portfolio refers to a bundle of assets held by an investor. The assets include stocks, bonds, real estate, bank accounts and other financial products on the market that can yield a return through investment. The portfolio is usually used as a risk reduction strategy by investors. Portfolio management is a decision making process that involves determining the types and amounts of assets that should be invested or divested and when. This is very similar to the process of technical debt management where software managers make decisions on when and what technical debt items should be paid or kept. Due to the involved uncertainty, the expected return and variance of the return are often used to evaluate the performance of assets. This evaluation can be performed on a qualitative basis if the future value of an asset is highly uncertain, or on a quantitative basis using mathematical models, if the information to derive the future value is available. The goal of portfolio management is to select the asset set that can maximize the return on investment or minimize the investment risk.

Since different types of assets such as stocks and real estate vary in liquidity and sensitivity to market changes, they usually exhibit different volatility and performance patterns. Even in a single asset category such as a stock market, the prices of different stocks may have different fluctuation ranges and change directions. Therefore, holding a portfolio of assets, i.e. diversifying the investment, is less risky than investing in a single asset. Based on the diversification principle, various portfolio models have been proposed. A portfolio model is a mathematical formulation of the diversification problem. Among these models, the Modern Portfolio Theory is an example of a mean-variance analysis model, which uses the expected return and the return variance to measure the performance of a portfolio [8].

Although this model is competent in finding the optimal portfolio, the model assumptions have raised challenges to its application. For example, such models assume that the assets' return conforms to a normal distribution, which may not be true in reality. Therefore this model has been extended by other researchers to allow other distributions of the asset return [9]. In addition, some assumptions are specific to the finance domain and thus the model cannot be used for technical debt management without tailoring. For example, assets in the finance domain are continuously divisible, while it's not generally possible to partially own a technical debt item. Therefore, a new constraint needs to be added to the model when it is applied to technical debt management [10].

D. Options

Paying off an instance of technical debt (e.g., by refactoring) can be seen as an investment decision, in that it incurs relatively certain short-term costs—a commitment of time, money and/or resources—in the pursuit of more uncertain longer-term benefits—in the form of maintenance cost-savings. This accounts for differences in the timing of paying off different technical debt items (i.e., when will the refactored part be used?) and their "risk" (i.e., how confident are we of the estimates?). Every module provides the right but not the symmetric obligation to be replaced with a better version to accommodate changes. The value of such an investment is in the form of options. In other words, investment in refactoring is analogous to purchasing the option that facilitates change to the software in the future, if the software has to be changed, but without immediate profits.

Previously, people have experimented with applying the Black-Scholes valuation formula [11] to value architectural investments (of which refactoring is a common example). The Black-Scholes formulation, however, is unfamiliar to managers and requires the estimation of key parameters that are difficult to determine in practice, such as a "replicating portfolio" to estimate the future value of an asset, which does not exist for software modularization. Baldwin and Clark's Net Option technique (NOV) [12, 13] exploits the same idea but uses statistical methods to calculate options values, without the hard-to-justify arbitrage assumption in the Black-Scholes formulation. The basic idea of NOV is that, the more likely to change a module is (which they call the *technical potential* of the module), the more valuable it is to make improvements to it (which they call conducting *experiments* on it); and the more improvements, the more likely it is to realize its greatest potential value. On the other hand, if a module has a lot of dependencies, or, is very complex, then it is more expensive to make such improvements. The "experiment" cost offsets the benefits of improving the module. The essence is, simply put, that independent modules with high technical potential have the highest option values. It is difficult to apply NOV

directly, however, because the key parameter, technical potential, is hard to quantify.

III. Discussion

All decision models discussed input estimates of principal and interest, but they differ in the level of additional input they require from the user. AHP, for example, requires the user to make pair wise comparisons. On the other hand, the portfolio model does not require user input beyond the principal and interest estimates. These models address the decision problem in different ways, and have their conditions of application, strengths and weaknesses for decision support. It remains unknown under what situations they work best for software maintenance decisions and whether combinations of these models will yield better results than any one in isolation. Most likely, different models perform better (in terms of expressability, user satisfaction, and efficiency) in different situations. It may even be possible that the overhead of applying a formal decision model is overkill for technical debt decision making, i.e. is not cost-effective no matter which approach is applied. In order to address all these questions, these approaches should be evaluated in real settings.

These evaluations could take any of several forms. We believe the first step should be to study each decision approach in isolation through in-depth case studies, in order to better understand the strengths and weaknesses of each one in the context of technical debt management. Such case studies could be more broadly designed, to study the management of technical debt in general (from identification, to measurement, to decision making). Such case studies would yield not only insight into the appropriateness of the decision approach being used, but also into what makes decision making about technical debt in particular different from other types of software maintenance-related decision making.

After understanding each decision approach in more detail (and possibly eliminating one or more as infeasible), studies could be conducted to compare the decision approaches side by side. Outcome variables to be used as a basis for the comparison could include ease of use, effort to use, the degree to which the results are trusted by developers and managers, understandability, and value added. Side by side direct comparisons of the approaches are best done in a laboratory setting, with fabricated technical debt information in fabricated, but realistic, scenarios. Industrial comparative studies would be more difficult, as it is usually not possible to repeat a decision scenario in two different ways without a learning effect. However, repeated case studies in industrial contexts would be useful to iteratively refine and tailor the approaches, as well as our understanding of which decision approaches work best in different contexts.

As part of a larger plan to build an end-to-end technical debt management toolkit, we have planned to conduct these types of studies. Our vision for the toolkit includes the ability to recommend different decision approaches, or combinations of approaches, appropriate to different contexts and development scenarios. .

Acknowledgment

The participation of Seaman, Guo, Zazworka, Vetró, and Shull in this work is supported by the US National Science Foundation, award #0916699.

References

[1] W. Cunningham, "The WyCash Portfolio Management System," in *Addendum to the proceedings on Object-oriented programming systems, languages, and applications*, 1992, pp. 29-30.

[2] Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. Q. B. Da Silva, A. L. M. Santos, and C. Siebra, "Tracking Technical Debt – An Exploratory Case Study," in *27th IEEE International Conference on Software Maintenance (ICSM' 11)*, Williamsburg, VA, USA, 2011, pp. 528-531.

[3] C. Seaman and Y. Guo, "Measuring and Monitoring Technical Debt," *Advances in Computers,* vol. 82, p. 22, 2011.

[4] W. S., C. Y., K. M., and D. M., "Detecting software modularity violations," presented at the 33th International Conference on Software Engineering, Honolulu, Hawaii, USA, 2011.

[5] N. Zazworka, C. Seaman, and F. Shull, "Prioritizing Design Debt Investment Opportunities," presented at the 2nd Workshop on Managing Technical Debt, Honolulu, Hawaii, USA, 2011.

[6] T. L. Saaty, *Decision making for leaders: The analytical hierarchy process for decision in a complex world*. Belmot, CA, USA: Lifetime Learning Publications, 1982.

[7] J. Buchanan and N. Kock, "Information overload: A decision making perspective," presented at the *International Conference on Multiple Criteria Decision Making (MCDM)*, Ankara, Turkey, 2001.

[8] H. Markowitz, "Portfolio Selection," *The Journal of Finance,* vol. 7, pp. 77-91, 1952.

[9] F. A. Sortino and L. N. Price, "Performance Measurement in a Downside Risk Framework," *The Journal of Investing,* vol. 3, pp. 59-64, 1994.

[10] Y. Guo and C. Seaman, "A Portfolio Approach to Technical Debt Management," presented at the 2nd Workshop on Managing Technical Debt, Honolulu, Hawaii, USA, 2011.

[11] K. Sullivan, P. Chalasani, S. Jha, and V. Sazawal, "Software Design as an Investment Activity: A Real Options Perspective in Real Options and Business Strategy: Applications to Decision Making," *Risk Books,* 1999.

[12] C. Baldwin and K. Clark, *The Power of Modularity* vol. 1: MIT Press, 2000.

[13] K. Sullivan, W. Griswold, Y. Cai, and B. Hallen, "The structure and value of modularity in software design," presented at the ESEC/SIGSOFT FSE, 2001.