Empir Software Eng
DOI 10.1007/s10664-014-9351-7

Exploring the costs of technical debt management – a case study

Yuepu Guo · Rodrigo Oliveira Spínola · Carolyn Seaman

Abstract Technical debt is a metaphor for delayed software maintenance tasks. Incurring technical debt may bring short-term benefits to a project, but such benefits are often achieved at the cost of extra work in future, analogous to paying interest on the debt. Currently technical debt is managed implicitly, if at all. However, on large systems, it is too easy to lose track of delayed tasks or to misunderstand their impact. Therefore, we have proposed a new approach to managing technical debt, which we believe to be helpful for software managers to make informed decisions. In this study we explored the costs of the new approach by tracking the technical debt management activities in an on-going software project. The results from the study provided insights into the impact of technical debt management on software projects. In particular, we found that there is a significant start-up cost when beginning to track and monitor technical debt, but the cost of ongoing management soon declines to very reasonable levels.

Keywords Technical debt · Decision making · Cost · Case study

1 Introduction

Software development teams often face the challenge of delivering software products under tight schedules while trying to keep the quality up to standard. To deal with time and resource constraints, software developers have to prioritize, focusing only on crucial requirements, and

Communicated by: Tony Gorschek

Y. Guo (✉) · C. Seaman
Department of Information Systems, University of Maryland Baltimore County, Baltimore, MD, USA
e-mail: yuepu.guo@umbc.edu

C. Seaman
e-mail: cseaman@umbc.edu

R. O. Spínola
Department of Systems and Computing, University of Salvador, Salvador, Bahia, Brazil
e-mail: rodrigo.spinola@pro.unifacs.br

R. O. Spínola
Fraunhofer Project Center for Software and System Engineering at Federal University of Bahia, Salvador, Bahia, Brazil

Published online: 30 November 2014

even take shortcuts. In such situations, long-term maintainability is often given little attention or completely ignored. As a result, low-quality artifacts emerge, which in turn adds more constraints on future maintenance tasks and makes modification more difficult, costly and unpredictable. This phenomenon is called "Technical Debt" (Cunningham 1992).

By incurring technical debt, software managers can trade off software quality against productivity. Maintenance time or cost is reduced in the short term, which is the main advantage of incurring technical debt. However, this advantage is achieved at the cost of extra work in the future, analogous to paying interest on a debt. In this sense, the term "technical debt" precisely characterizes the effect of delaying software maintenance tasks (or doing them quickly and less carefully) on software projects. Nonetheless, technical debt is not exactly the same as financial debt. The major difference lies in the fact that the interest associated with technical debt may or may not need to ever be paid off. For example, it is not necessary to refactor a software module that is overly complex if no further changes will be requested on the module in the future. It is this uncertainty that differentiates technical debt from financial debt and further complicates the issue, but it also provides an opportunity for software managers to leverage technical debt for their projects. Technical debt benefits a software project as long as it is handled before the bigger long term cost is realized. Therefore, software managers have to balance the costs and benefits of technical debt and make informed decisions on when and what technical debt should be paid off. In fact, incurring technical debt has been widely used as a strategy in software project management. Such a strategy can be effective as long as the long-term costs of the debt are managed.

Besides explicit strategic decisions, there are other situations in software maintenance that give rise to technical debt, e.g. carelessness, incompetence, or lack of proper process. All of these causes help explain why technical debt is so common in software projects. It has been reported that technical debt exists in most software projects in all industry segments, and the average cost to pay off this debt is $3.61 per line of code (CAST 2012). Because of the magnitude of technical debt and its impact on software projects, proper management of technical debt has become an important contributing factor to the success of software projects. It is agreed that technical debt can turn into a big problem unless it is properly handled, but it is currently managed in an implicit way, if at all. Decisions are largely based on a manager's experience, or even gut feeling, rather than hard data gathered through proper measurement. While some approaches do rely on software measurement data (Gaudin 2009, Schmid 2013), there is a shortage of evidence that the attributes being measured have a causal effect on maintenance outcomes like quality and productivity. Thus, decisions based on traditional software metrics (size, complexity, etc.) are suspect and only appear to be more rigorous than subjective approaches based on experience.

Hereby the technical debt problem in a software project is often underestimated and hence not given enough attention until it gets out of control. The results can be serious schedule slippage, budget overrun, and project failure. To overcome the disadvantages of current technical debt management, we have proposed a technical debt management approach (Seaman and Guo 2011), which promotes explicitly identifying, measuring and monitoring technical debt, to help software managers make informed decisions for their projects.

To evaluate our proposed approach, we must examine both the costs and benefits of managing technical debt. Because of the expected long time horizon in achieving the benefits of better technical debt management, it is useful from an empirical research perspective to explore the costs and benefits of the approach in separate studies. To explore the benefits of technical debt management, we have designed a set of retrospective studies that simulate the relative benefits of different decisions that would have been made in the presence of explicit technical debt management. These retrospective studies are ongoing and one has been reported elsewhere (Guo et al. 2011). To explore the costs of explicit technical debt management, in particular using the approach we have described in (Seaman and Guo 2011), we choose a case study design. Several such case studies are ongoing, and reporting the results of the first such completed case study is the goal of this article.

In the study described here, we explore the costs of the new approach by tracking the technical debt management activities in an ongoing software project. The research questions for this study are:

(1) What are the costs of managing technical debt using the proposed approach?
(2) How does technical debt information contribute to decision making?

In the following sections, we first review some important work from the related research areas from which we drew inspiration for the technical debt management approach and its evaluation. In section 3 we present the proposed approach, followed by a detailed description of the study design in Section 4. Then we give the results of this study in Section 5. Finally we conclude with a discussion of the study's limitations and future work.

2 Related Work

The term "technical debt" was coined by Cunningham in 1992, where he presented the metaphor of "going into debt" when a new release of a system is shipped (Cunningham 1992). He further pointed out the two sides of technical debt, that is, that a little debt can speed up software development in the short run, but spending extra time on "not-quite-right code" is like paying interest on that debt (Cunningham 1992). Technical debt originally referred to delayed or "quick and dirty" work in the design and implementation phases, resulting in immature code, but this metaphor has been extended to include any immature artifacts in the software development lifecycle.

Early work in this area focuses on establishing the foundation of technical debt research by conceptualizing the phenomenon and classifying technical debt. There are multiple dimensions by which technical debt can be classified. McConnell defined two categories of technical debt, unintentional and intentional debt (McConnell 2007). Fowler extended this classification and created the technical debt quadrants (Fowler 2009). The quadrants are formed by two dimensions – deliberate/inadvertent and reckless/prudent. This type of classification is helpful in finding the causes of technical debt, which lead to different identification approaches. Technical debt can also be classified from the perspective of the software lifecycle – design debt, testing debt, defect debt, documentation debt, etc. (Rothman 2006). Design debt refers to architectural violations or deficiencies in source code; testing debt refers to the tests that are skipped or not developed sufficiently; defect debt refers to the known defects that are not fixed yet; documentation debt refers to missing or out-of-date documentation (Seaman and Guo 2011). This type of classification sheds light on the possible sources and forms of technical debt, each of which may need different measures for identification, and approaches for management.

Originally, the technical debt metaphor was mostly used as a communication device, as shifting the dialog from a technical vocabulary to a financial vocabulary made discussions clearer and easier to understand for non-technical people (McConnell 2007). As the term has become popular, technical debt has also drawn attention from the software research community in recent years. One of the most fruitful areas of achievement in technical debt research is technical debt identification, which is the first step in technical debt management. Many approaches have been proposed or adapted for technical debt identification. Specific tools and techniques have also been developed based on these approaches. Identification of technical debt, especially design debt, often relies on software quality assessment methods. Since technical debt compromises software maintainability, the metrics for evaluating software maintainability can be useful for technical debt identification (Gaudin 2009). Further, some quality metrics have been used to find patterns that indicate poor programming practices and bad design choices. Such patterns, termed "code smells" (Fowler et al. 1999), are believed to cause maintainability problems over time, hence can be used as indicators of technical debt. To detect code smells, rules have been defined using metrics and thresholds, which can be found in the literature (Fowler et al. 1999). These rules facilitate code smell identification by automatic means. Indeed, several tools have been developed based on these rules to detect code smells (Emden and Moonen 2002). There are other tools, for example, Software Maps, that facilitate technical debt identification through visualizing internal software quality and risk (Bohnet and Döllner 2011, Wang et al. 2013). Similar to code smell detection but targeted to the architectural level, extended augmented constraint networks (EACN) have been used to transform software architectural models into dependency relations, which facilitates detection of software architectural decay (Mo et al. 2013). Tools for visualizing architectural decay have also been developed (Brondum and Zhu 2012).

Besides technical debt identification, approaches have been proposed to facilitate technical debt quantification. One approach to quantifying technical debt is to measure the decrease in software productivity during the software development process (Fowler 2003). Other approaches quantify technical debt by estimating the cost of fixing software quality issues to achieve an ideal quality level, as defined by the values of certain software metrics (Nugroho et al. 2011; Letouzey 2012). Technical debt can also be evaluated by measuring the healthiness of the software architecture, which is an important indicator of future work cost (Nord et al. 2012). Brown et al. have proposed a research framework to quantify software architecture healthiness using propagation cost and dependency analysis (Brown et al. 2011). Based on the metrics of code smells, Marinescu proposed to quantify design flaws, an indicator of technical debt, using influence, granularity and severity (Marinescu 2012). Although identification and quantification of technical debt are essential for technical debt management, they are not the focus of our work. Instead, our focus is decision making using technical debt information, which is the central issue in technical debt management.

Closer to our focus, strategies for paying off technical debt have also been proposed by software practitioners as well as researchers. One strategy is to try to determine the time point at which technical debt has to be paid (Fowler 2007). Other proposed strategies include paying the highest interest debt items first (Lester 2008), or paying technical debt items according to their predicted defect and change proneness (Zazworka et al. 2011). These strategies are less formal, based on personal experiences from tackling similar problems in related research areas, and they reflect early thoughts on the technical debt management problem. By contrast, some other approaches for technical debt decision making take a more formal approach. For example, Schmid proposed an approach to formalize the technical debt concept and decision making (Schmid 2013). In this approach, technical debt is modeled using implementation cost, rework cost and future evolution path. Thus technical debt decision making turns into an optimization problem. Although this is theoretically promising, the approach is not practical for real world application due to the hard requirements for finding an optimal solution, even with relaxed assumptions. Therefore, Schmid proposed a simplified approach that only considers evolution cost, refactoring cost and the probability that the predicted evolution path will be realized. In the sense of technical debt decision making, this simplified approach is very similar to the principal-interest-probability approach we use and hence supports its application in this study. Another example is Nord et al.'s work in which technical debt for two distinct delivery strategies, i.e. quick delivery vs. low rework cost, were evaluated using an architectural metric (Nord et al. 2012). With a concrete software project, they demonstrated the importance of architectural debt information in decision making related to product delivery. Agile development appears to be more prone to technical debt accumulation compared to traditional software development approaches, due to its delivery-oriented focus. To take advantage of agility while still maintaining a stable infrastructure for long term health, Bachmann et al. proposed a set of architectural tactics for system decomposition, architecture development and staffing, and proposed a way to make these tactics work together harmoniously (Bachmann et al. 2012). Although the tactics do not explicitly address the technical debt management issue, they offer another angle to view and investigate the technical debt problem.

Most of the approaches and strategies mentioned above focused on a particular aspect of the technical debt management problem, such as quality metrics for evaluating technical debt, or incorporating architectural information. But managing technical debt requires a holistic approach that can address all the main aspects of the problem of whether and how technical debt should be managed. Even within the scope they are targeted to, these approaches have limited power due to the stringent assumptions or application conditions. For example, Nugroho et al.'s approach heavily relies on historical project data, which are often lacking or unavailable (Nugroho et al. 2011). Moreover, none of the approaches in the literature have been tried and validated in practice, to the best of our knowledge. Therefore, our research objective lies not only in development of technical debt management approaches, but also in the empirical validation of these approaches.

Given the uncertainty involved in its re-payment, technical debt can be considered as a particular software risk. Therefore, managing technical debt could leverage approaches for software risk management. Since the 1970's, various techniques and approaches have been proposed to deal with software risks (Boehm 1991; Alter and Ginzberg 1978, Davis 1982, Charette 1989; Fairley 1994; Higuera and Haimes 1996). Among them the most famous is Boehm's approach (Boehm 1991), which established risk management as an important research field in software project management and laid the foundation for most of the work in this field (Kontio 2001). Risk management approaches or standards were also developed by other research institutes and organizations, e.g., ISO (ISO 2002), SEI (Alberts et al. 1996), U.S. Department of Defense (DoD) (DoD 2006) and NASA (Stamatelatos 2002). Although the definitions of the risk management process are presented in different ways, they are consistent with one another in terms of the basic components that the process should possess – risk management starts with risk identification, followed by risk analysis, risk control and finally evaluation of the risk management approaches. Inspired by risk management, we mean to follow the same process (identify, analyze, control and evaluate) to manage technical debt, although the study described in this paper, due to limitations of the study itself, does not include the evaluation step. Furthermore, some approaches used for risk analysis, especially those used for probability estimation and risk prioritization, e.g. risk matrix and risk exposure analysis, are also applicable to technical debt management (Boehm 1991; Sisti and Joseph 1994).

In technical debt management the decision of whether to keep or pay off debt also depends on the present and future cost of the debt, which can be measured by the effort required to pay off the debt. Therefore, cost estimation is essential for managing technical debt. Research on software effort estimation started in the 1960's. Since then various models and techniques have been proposed. Effort estimation approaches can be categorized as algorithmic methods, which use mathematical formulae to model software cost, or non-algorithmic methods (Leung and Fan 2002). Putnam's model (Putnam 1978) and COCOMO (Boehm 1981) are examples of algorithmic methods. Non-algorithmic methods include analogy-based estimation methods and expert estimation methods. As its name suggests, expert estimation methods involve consulting one or more experts and the estimates are based on their opinion and experience. Delphi (Dalkey and Helmer 1963), Top-down, Bottom-up and Parkinson's law (Parkinson 1957) belong to the expert estimation category. It should be noted that each approach has advantages as well as shortcomings. There is no single approach to software cost estimation that can fit all circumstances (Shepperd and Kadoda 2001). Therefore, selecting an appropriate cost estimation approach relies on knowledge about the "home ground" where the approaches perform best. In a general scenario of technical debt management, paying off debt involves implementing a maintenance task. In this sense selecting an approach for technical debt estimation is nothing more than choosing a cost estimation approach for software maintenance projects. However, in our research, the best choice is the estimation method currently used in the project even if it's not the most accurate one. The reason for the choice is that our study designs should avoid introducing new factors that may have effects on the evaluation object, and changing the cost estimation method during the study could have such effects.

3 Proposed Approach

To facilitate managing technical debt in an explicit way, we have proposed a simple management framework (Seaman and Guo 2011). The framework is meant to help organize the activities and information needed to manage technical debt in practice. It is designed to allow incorporation of human judgment in all phases. Thus the framework can easily evolve by incorporating results, understanding, specific techniques, and emerging theories.

3.1 Technical Debt Item

The proposed framework centers on a "technical debt list", as shown in Fig. 1. The list contains technical debt "items", each of which represents a delayed task that may cause maintenance problems in the future. Each item has a set of properties, which define what the technical debt is, where it is located, when it was detected, who is responsible, and estimates of the principal and the interest. The principal refers to the cost of eliminating the debt or the effort required to complete a task that is considered being postponed, i.e. a technical

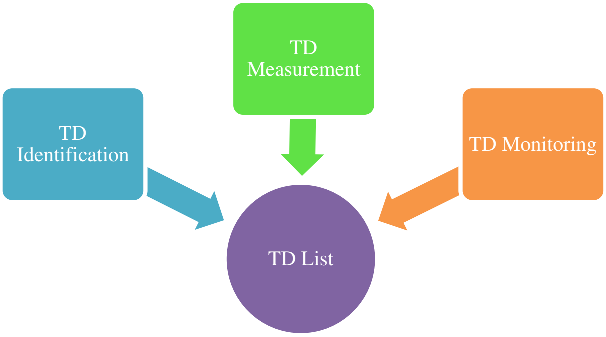


Fig. 1 Technical Debt Management Framework

debt candidate. The interest is composed of two parts. The "interest probability" refers to the likelihood that an incurred debt item turns to a problem by making other work more expensive over a given period of time or a release. The "interest amount" represents the amount of extra work that may be needed in the future if this debt item remains.

Initially, the principal, interest amount and interest probability of a technical debt item will be estimated using a high-medium-low scale. Although this is a very coarse-grained estimation, it is sufficient in the initial stages for tracking technical debt items and making preliminary decisions (Seaman and Guo 2011).

3.2 Technical Debt Management Process

The process of managing technical debt consists of three general activities: TD Identification, TD Measurement, and TD Monitoring, which revolve around a technical debt list, the central component of the framework. The process starts with identifying the technical debt items to construct the technical debt list. As outlined in section 2, a number of approaches to identifying technical debt have been proposed and examined in the literature (Gaudin 2009, Wang et al. 2013; Mo et al. 2013), most of them focusing in a particular type of debt. The general management process we propose here does not assume any particular technical debt identification approach, but only assumes that the output of the identification activities is in the form of a technical debt list. The procedures for identifying technical debt in our case study are described in section 4.3.

After technical debt identification, the next step is to measure the debt items on the list by estimating the principal, interest amount and interest probability. Again, many of the technical debt identification approaches proposed in the literature also have associated proposals for quantifying the type of debt being identified (Brown et al. 2011; Nord et al. 2012). However, many don't, and even those that do recognize that quantification is a very hard problem, and fraught with a high degree of uncertainty. Our high-level management framework and process simply assumes that some quantification has been done with the items on the technical debt list, without assuming anything about the level of detail or accuracy of those estimates. Obviously, more detail and more accuracy will, in most cases, yield better results in terms of the effectiveness of the management process, as with any aspect of software management, but we assume that any information about the scale of the principal and interest will be useful in managing it. To temper the expectations about the quantification of the items on the technical debt list, the management approach assumes that the initial estimates for the principal and interest will use a coarse-grained scale, i.e. high, medium, and low. Such a coarse-grained characterization is sufficient for early decision-making and management activities. The estimates may be refined later, if needed, by using approaches specific to the type of technical debt item and how it was identified. The approach we used in the case study to estimate principal and interest is described in section 4.3.

Once they are quantified, the next major activity in the proposed technical debt management approach is to monitor debt items and make decisions on when and what debt items should be paid or deferred. Since existing debt items may be paid off in the software lifecycle and new debt items may be incurred as well, technical debt should be continuously monitored. The changes in technical debt must be reviewed and reflected in the technical debt list.

Section 2 outlined some approaches to technical debt decision making in the literature (Bachmann et al. 2012; Nord et al. 2012; Schmid 2013), none of which have been validated in practice. In section 3.3, we propose a simple approach based on risk management techniques, which we adapt in the case study.

3.3 Application in Release Planning

The goal of identifying and measuring technical debt is to facilitate decision making. Release planning is one of the scenarios in which the technical debt management approach can be applied to help software projects make optimal decisions. In release planning, a decision must be made regarding whether and what technical debt items should be paid off in the upcoming release. The following example illustrates how our approach is used for release planning.Consider a software project in which component X will be significantly changed in the next release. One of the decisions that has to be made during release planning is whether to pay down some debt on component X at the same time. If so, how much and which items should be paid?Assume that a technical debt list has been updated to reflect the current understanding of technical debt in the project. Each item on the debt list can be sorted by its value, i.e. the high, medium, and low estimates for principal and interest. The release planning process is illustrated in Fig. 2.In step (1), a subset of the TD list is extracted, focusing on items associated with component X, as they are the most likely to have immediate impact and may be easier to pay off while doing other work. Because the cost and impact of these items might be different than originally thought (due to the fact that component X is planned to be modified anyway in this release), in step (2) we re-evaluate the original high/medium/low estimates for these items. For instance, some items may have a greatly reduced principal if they overlap with the planned work on component X in the release.In step (3), we further restrict the subset of TD items we're considering by choosing high-impact items and doing numeric estimates only for those items. In this way, we reserve the time-consuming and difficult task of effort estimation for only those items most likely to be chosen for payoff in this release.Step (4) operates on this reduced subset, and compares the cost and benefit for each item in that subset. The cost of choosing this item to be paid off is equal to the principal, i.e. the amount of effort required to eliminate this debt item. The benefit of paying off this item is the future extra work that would be avoided by eliminating the debt now. This is represented by the expected interest on this item, calculated as the product of interest amount and interest probability. The result of

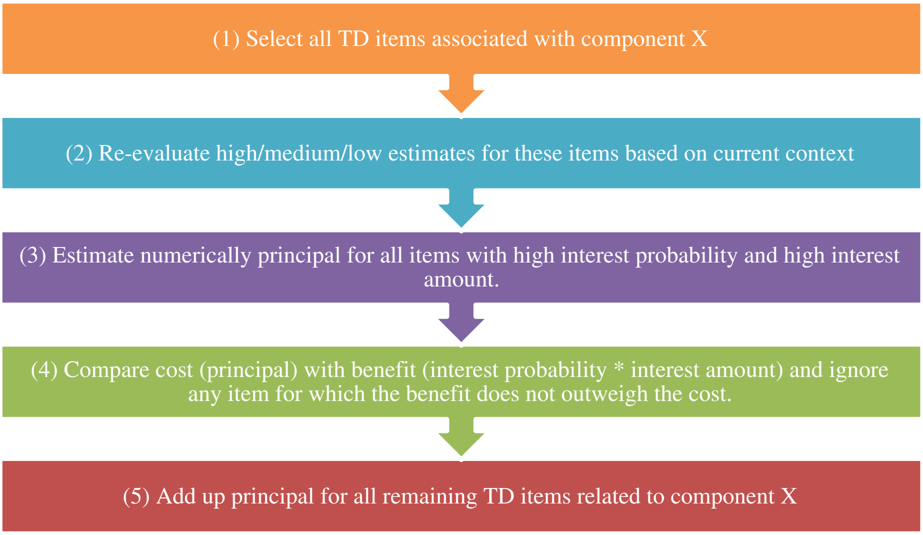

(1) Select all TD items associated with component X
(2) Re-evaluate high/medium/low estimates for these items based on current context
(3) Estimate numerically principal for all items with high interest probability and high interest amount.
(4) Compare cost (principal) with benefit (interest probability * interest amount) and ignore any item for which the benefit does not outweigh the cost.
(5) Add up principal for all remaining TD items related to component X

Fig. 2 Release Planning Process

step (4) is an even more reduced subset of items that not only are related to work currently planned for the upcoming release, but that also are likely to have high impact if not repaid, and for which the benefits of debt payment outweigh the costs.Finally, in step (5) we add up the estimated principal for the subset of items left after Step 4. This gives us a reasonable estimate of the cost of paying off the TD items that should be highest priority for paying off in this release. At this point, as part of the release planning process, we need to decide if this cost can be reasonably absorbed into the next release. If not, then we can use information about the interest (i.e. impact) related with these items to determine if the TD items should be prioritized over other tasks in the release (e.g. fixing bugs or new enhancements), and to justify the cost to management. In some cases, we might find that the cost of paying off the TD items can reasonably fit into the release schedule, and in fact there is room for further debt repayment. In this case, we repeat steps 3–5 with items with high interest probability and medium interest amount, and vice versa, then with medium for probability and interest, etc., until no more debt repayment can be absorbed by the release.

3.4 Known Limitations

In the proposed approach described above, we have made several simplifying assumptions that may not always hold in practice. For example, we assume that each technical debt item is independent of other items. Thus, estimation of TD interest can be done individually. This simplifies the prioritization mechanism for paying off these TD items. Another simplifying assumption is that all types of TD can be described as individual items, and that it is possible to capture all effects of that TD in the concepts of principal and interest. When these assumptions do not hold, the optimal decision for paying off technical debt may not be achieved. Therefore, we expect that this initial approach for TD management, as a starting point, will be refined, improved, and expanded through this and other studies. The objective of the case study reported here, then, is to provide support for the general approach we have outlined in this section, and provide input to its refinement.

4 Case Study Methodology

This study is part of a larger research effort that aims at technical debt theory construction through the study of how technical debt can be measured and monitored. The emerging theory addresses the relationship between the costs and benefits of explicitly managing technical debt in comparison with implicit management. To investigate these cost-benefit relationships, we designed two types of studies. The first type of study focused on the benefit side of the proposed technical debt management approach. In those studies (an example is Guo et al.'s study (Guo et al. 2011)) we are exploring the benefits retrospectively by applying the technical debt management approach (through simulation) to past releases of several ongoing software projects. The second type of study is targeted to understanding the costs of tracking technical debt by applying the approach to an ongoing software project through several release cycles in real time. The case study described here falls into the second type, that is, we use the selected software project to investigate the costs of explicit technical debt management.

4.1 Subject Project

The subject project was selected by convenience by the second author, who served as the company contact as well as one of the researchers conducting the case study. It was a software application from a Brazilian software company that provides enterprise-level software development, consulting and training services. The project consisted of a small database-driven web application for water vessel management. It included 25 K non-commented lines of code in 245 classes (of which 80 were domain classes). The application was written in Java and based on the MVC framework. The application included administration, resource allocation, ship management, and billing and operation modules, which were mapped to 95 use cases. The average development effort was 37 person-hours per use case. The project was developed with the following infrastructure: Eclipse IDE, Subversion (for code version control), and Trac (a bug and workflow tracking system).

There were 1 project leader, 1 technical leader and 7 developers working on the project. The project followed a Scrum-like development process to continuously integrate features and deliver working versions to the customer. Each sprint lasted about 1 week. The project began in late 2011. The first release of the application was approved by the customer in May 2012. Since then, a new customer release was delivered every sprint. The case study began with baseline data collection in March 2012. Afterwards, the project started managing technical debt explicitly using our proposed approach. We observed and collected data on the technical debt management process until the middle of November 2012. The case study ended in the middle of December, 2012 with the follow-up interview being completed.

4.2 Case Study Process

Due to the distance and language, the principal researcher (first author) did not have direct contact with the subject project. Therefore, this study involved a company contact, who was also part of the research team (the second author) and was in charge of translation, data collection (e.g., interviewing the project personnel), and communication between the project team and the research team. All communication with the project team was in Portuguese, while the research team communicated in English.

The case study was conducted with the following general steps (details in the following subsections):

(1) The company contact collected baseline data in terms of effort, cost and productivity of the project.
(2) We designed a simple spreadsheet to aid in managing the technical debt list and to collect data on its use. The list was kept under version control, along with other project documents.
(3) The company contact trained the project team on how to manage technical debt using the proposed approach and how to document and report information required by this study. Training materials were prepared by the research team in English and translated to Portuguese by the company contact.
(4) The project team identified technical debt items in their system to prepare the initial technical debt list.
(5) In the first release cycle, during which the project was tracking technical debt for the first time, the team members provided data to the research team regarding the amount of time they spent using the technical debt list and the problems and questions that came up.
(6) After the first release, the research team reviewed the technical debt list, the changes made by the project team and other data collected in the first release cycle. Then the research team asked questions about the project team's experience, the data they provided, and the process they followed. This began a back-and-forth discussion, which continued until all the questions were addressed, the research team was satisfied with

the types of data provided by the project team, and the project team was clear and comfortable with the proposed approach.

(7) The project team continued tracking the technical debt items in the successive three sprints using the approach. In the process, they documented the decisions made concerning technical debt, the principal and interest paid and/or avoided, and the costs (i.e. effort) of using the technical debt list. To make sure that all reported technical debt items were clearly consistent with the broad definition of technical debt, all data were first reviewed by the project leader and then sent to the research team for further review. There were numerous opportunities for clarification of any discrepancies or ambiguities.

(8) The company contact interviewed the project leader, who was in charge of release planning, about his decision making process and the effect of the approach on the project.

4.3 Data Collection

In step 1 of the case study process, we needed to collect baseline data about the effort, cost, and productivity of the project, which was to be used later for comparison. The baseline data we collected is summarized in Table 1. It gives an overall picture of the size and current productivity of the project. This data was gained via interview with the project leader (conducted by the company contact).

At the same time, the project team created the initial technical debt list. This activity began with a training on the concept of technical debt. This training was performed in Portuguese by the company contact and took about 30–40 min. After that, the team members had a chance to ask questions. The definition of technical debt presented to the team was: incomplete, immature, or inadequate artifact in the software development lifecycle which in turn adds more constraints on future maintenance tasks and makes modification more difficult, costly and unpredictable (Cunningham 1992). As mentioned in Section 2, there are multiple dimensions to categorize technical debt and several classifications have been proposed. Among these classifications, the one Rothman proposed (Rothman 2006) makes it clear that technical debt has different sources and forms, and hence may need different measures for identification, and approaches for management, which is the center of our larger research project. In addition, categorizing technical debt by development phase is natural to software developers. It would be easier for them to understand the technical debt concept and thus fulfill their tasks for this study more effectively. Therefore, we decided to use Rothman's classification as part of our technical debt training for the project team. Specifically, we explicitly defined the following types of technical debt:

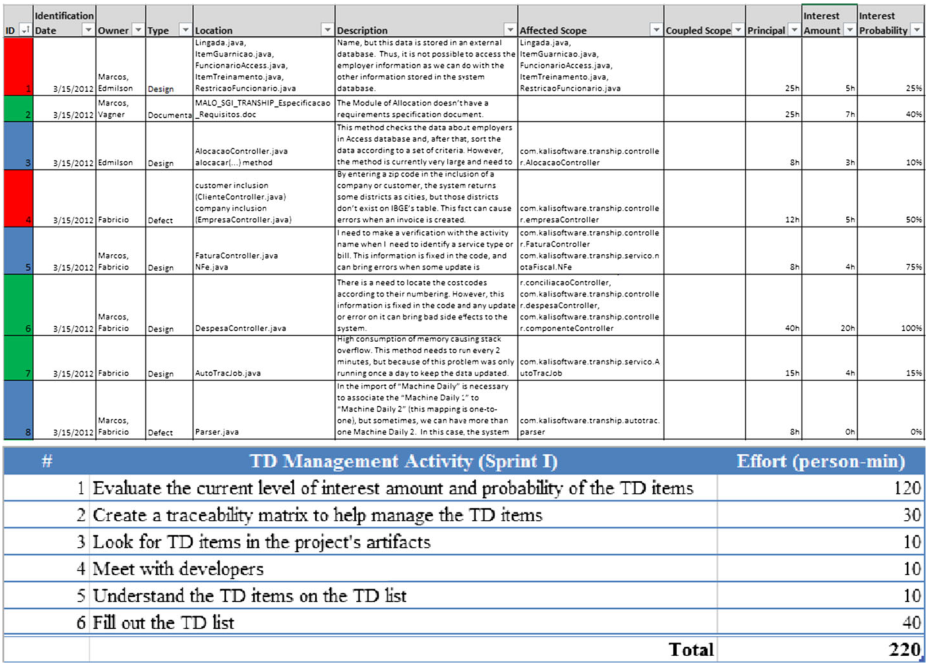
- Design debt: any kind of anomaly or imperfection that can be identified by examining source code and/or related documentation, that leads to decreased maintainability if not remedied;
- Testing debt: tests that were planned but not executed;

Table 1 Baseline Size and Productivity of the Project

No. of Software Modules	No. of Use Cases	Hours/Month	Average Person-hours/ Use Case	Average Person-hours/ sprint planning
8	95	120	37	5

- Documentation debt: documentation that is not kept up-to-date;
- Defect debt: known defects that are not yet fixed.

Finally, the development team was also trained on how to report technical debt items using the technical debt list (Seaman and Guo 2011). After that, the development team started to identify technical debt items. Basically, for this task, each member of the team reported technical debt items based on their role and activities on the project. Some of the developers looked manually for technical debt items in the parts of the source code with which they were working and familiar, others just recalled instances of incomplete, immature, or inadequate project artifacts that they had encountered. Some of the testers examined test reports or requirements specifications to identify debt items. No analysis tools were used. For each reported item, the team member assigned its type (design, testing, documentation, or defect).The team members were also asked to estimate rough values of principal, interest amount, and interest probability. To estimate these values, the development team was instructed during the training to use their own experience and expertise to make their best guess, and to use high, medium, and low for the initial estimates. After that, they were instructed to refine those values numerically, based on their own experience, during sprint planning. Thus, initially, all technical debt items had high, medium, and low estimates for principal and interest. Later, the team analyzed each technical debt item considered during sprint planning and estimated numerically these values. Information about each technical debt item was collected into a single spreadsheet (example shown in Fig. 3).Given the objective of this study, the cost of using the proposed technical debt management approach is the core category of data. The cost data includes costs of meetings, effort spent managing the technical debt list, and time spent gathering data to feed into technical debt management. These data were collected directly by the project leader during sprint planning over each of the 4 sprints included in the case study. For example, for each relevant meeting, the project leader recorded the number of people present and the length of the meeting, thus yielding the overall cost of the meeting. During the sprint planning meeting, the project leader asked team members to report time spent gathering data relevant to technical debt management. The project leader also recorded his own time spent managing the technical debt list and in other technical debt related activities.Data was also collected concerning the changes made to the technical debt list, including when a technical debt item was paid off or incurred, the updated estimates of the principal and interest, and the actual effort of paying off an item. Collecting this change data allowed us to monitor the implementation of the technical debt management approach. The change data was recorded in the same spreadsheet as the technical debt list, which was kept under version control.Another type of data that was collected was about the technical debt decisions made in the process. We were interested in the decision making process, and the ways in which it was influenced by technical debt information. Although this study focuses on the cost side of technical debt management, we also expected that information about decisions might reveal some of the short-term benefits that could be observed from using the approach, as well as problems and suggestions regarding the usefulness of the approach. The decisions themselves were recorded as changes to the technical debt list, and more in-depth information on decisions was collected through the final interview with the project leader.In short, we collected 6 categories of data: (1) baseline data of the project, (2) costs of using the technical debt management approach, (3) changes in the technical debt list (4) decisions on when and what technical debt items should be paid off, (5) benefits of using the technical debt management approach, and (6) problems and suggestions for the proposed technical debt management approach.Data (1) were collected as described above. Data (2)–(4) were collected from the project team when they implemented the approach. These Data were collected using the simple spreadsheet (shown in Fig. 3) we designed in step 2 of the case study process (section 4.2). Data (4)–(6) were collected through the final interview, after the project team finished tracking the technical debt items.The spreadsheet we used for data collection had two parts, one for the technical debt list and one for the effort data. The technical debt list had columns for technical debt attributes and rows for technical debt instances, as shown in Fig. 3. The table for recording technical debt management activities and effort was very simple to allow the project leader to define the activities and categories of effort. For example, the first entry in the table shown in Fig. 3 represents the project leader and two developers spending 40 min (so 120 person-minutes in total) in evaluating the principal and interest of the technical debt items in the list. The initial form of the spreadsheet was designed by the research team. When it was sent to the development team, they revised it according to the information availability and used it to log the data they collected. The spreadsheet was updated and version controlled to reflect changes in the technical debt items and effort spent in technical debt management at the beginning of each sprint.The set of questions used to interview the project leader at the end of the case study consisted of four sections – the profile of the interviewee, the original management practices, the experience of using the proposed technical debt management approach and technical debt measurement. The questions in the first section elicited the project leader's experience level in software development and role in the subject project. We used the questions in the second section to check if the participant had knowledge or past experience on managing technical debt, which would help us determine the effectiveness of the proposed approach. The third section was the main section of the interview. In this section we asked the project leader to elaborate on how he performed technical debt management, the factors he considered when he planned a sprint, how the technical debt information was used in the sprint planning, the impact of the proposed approach on the project, and the problems



ID	Identification Date	Owner	Type	Location	Description	Affected Scope	Coupled Scope	Principal	Interest Amount	Interest Probability
🟥 1	3/15/2012	Marcos, Edmilson	Design	Lingada.java, ItemGuarnicao.java, FuncionarioAccess.java, ItemTreinamento.java, RestricaoFuncionario.java	Name, but this data is stored in an external database. Thus, it is not possible to access the employer information as we can do with the other information stored in the system database.	Lingada.java, ItemGuarnicao.java, FuncionarioAccess.java, ItemTreinamento.java, RestricaoFuncionario.java		25h	5h	25%
🟩 2	3/15/2012	Marcos, Vagner	Documenta	MALO_SGI_TRANSHIP_Especificacao_Requisitos.doc	The Module of Allocation doesn't have a requirements specification document.			25h	7h	40%
🟦 3	3/15/2012	Edmilson	Design	AlocacaoController.java alocacar(...) method	This method checks the data about employers in Access database and, after that, sort the data according to a set of criteria. However, the method is currently very large and need to	com.kalisoftware.tranship.controller.AlocacaoController		8h	3h	10%
🟥 4	3/15/2012	Fabricio	Defect	customer inclusion (ClienteController.java) company inclusion (EmpresaController.java)	By entering a zip code in the inclusion of a company or customer, the system returns some districts as cities, but those districts don't exist on IBGE's table. This fact can cause errors when an invoice is created.	com.kalisoftware.tranship.controller.empresaController		12h	5h	50%
🟦 5	3/15/2012	Marcos, Fabricio	Design	FaturaController.java NFe.java	I need to make a verification with the activity name when I need to identify a service type or bill. This information is fixed in the code, and can bring errors when some update is	com.kalisoftware.tranship.controller.FaturaController com.kalisoftware.tranship.servico.notaFiscal.NFe		8h	4h	75%
🟩 6	3/15/2012	Marcos, Fabricio	Design	DespesaController.java	There is a need to locate the cost codes according to their numbering. However, this information is fixed in the code and any update or error on it can bring bad side effects to the system.	r.conciliacaoController, com.kalisoftware.tranship.controller.despesaController, com.kalisoftware.tranship.controller.componenteController		40h	20h	100%
🟩 7	3/15/2012	Fabricio	Design	AutoTracJob.java	High consumption of memory causing stack overflow. This method needs to run every 2 minutes, but because of this problem was only running once a day to keep the data updated.	com.kalisoftware.tranship.servico.AutoTracJob		15h	4h	15%
🟦 8	3/15/2012	Marcos, Fabricio	Defect	Parser.java	In the import of "Machine Daily" is necessary to associate the "Machine Daily 1" to "Machine Daily 2" (this mapping is one-to-one), but sometimes, we can have more than one Machine Daily 2. In this case, the system	com.kalisoftware.tranship.autotrac.parser		8h	0h	0%

#	TD Management Activity (Sprint I)	Effort (person-min)
1	Evaluate the current level of interest amount and probability of the TD items	120
2	Create a traceability matrix to help manage the TD items	30
3	Look for TD items in the project's artifacts	10
4	Meet with developers	10
5	Understand the TD items on the TD list	10
6	Fill out the TD list	40
	Total	220

Fig. 3 Data Collection Spreadsheet

and the benefits he observed. The questions in the last section were to elicit how effective the current cost measures of managing technical debt are.The questionnaire was developed by the research team in English (this version is available at http://www.technicaldebt.umbc.edu/index_files/QTSC.pdf). Then the second author translated it to Portuguese and conducted the interview. After the interview, the second author translated the results into English for analysis by the research team.

4.4 Data Analysis

The cost data gathered from the sprint planning meetings was compared with the project's baseline to determine the effect of the proposed technical debt management approach on the project. Besides the cost data, we collected a significant amount of qualitative data through the interview and the technical debt list. These qualitative data were coded according to a coding scheme developed based on the research questions. The coding was performed by the first author of this paper using a text editor, by highlighting chunks of text and annotating those chunks with the name of the associated code. The main codes of the scheme are shown in Table 2. With these codes, the data were grouped to form the chain of evidence to explain the changes that explicit technical management brought to the project and to reveal the rationale behind the decisions made in the sprint planning. It also helped us understand the benefits and problems of the proposed approach.

5 Results and Findings

The case study began in March 2012. First the project baseline was collected and the initial technical debt list was created, as described in section 4.3. By the end of September 2012, the project team finished documenting the initial 31 technical debt items. Among these items design debt accounted for the majority, but there were also other types of technical debt. We can see from Table 3 that a new type of debt (different from those in the initial list of types from the literature) was suggested and reported by the development team: process debt. The team members reported these items as faults in how some activities were performed because they were inappropriate for the project.

The initial rough estimates for principal and interest ranged from low to high. Later numeric estimates for principal of these items ranged from 4 to 40 person-hours, while the estimated interest amount ranged from 0 to 20 person-hours. The interest probability also varied greatly (0 %–100 %).

The project team started tracking technical debt from the beginning of October 2012. Since some of the originally identified technical debt items had already been paid off at that time, the

Table 2 Main Codes of the Coding Scheme

Main Code	Definition
Cost	Information regarding the cost incurred by technical debt management activities
Benefit	Information regarding the benefit related to the application of the TD approach
Decision	Decisions in the course of the study regarding sprint planning and technical debt repayment
Approach	Application process and user feedback of the TD approach
Measure	Information regarding technical debt evaluation method and activities

Table 3 Technical Debt Items by Type

Type	design	defect	documentation	testing	process	total
No. of items	18	7	4	1	1	31

initial technical debt list consisted of 22 items. Then the research team interacted with the project team to review the collected data and the process they

followed to manage the technical debt during the first sprint. After that, these technical debt items were tracked over the subsequent 3 sprints with their principal and interest being re-estimated as the situation of the project changed. Meanwhile the project leader kept the technical debt list updated as the existing items were paid off and new items were identified. Table 4 shows the changes in the technical debt list over the four sprints. While it can be seen from Table 4 that the number of technical debt items decreased from 22 to 15 (i.e. by 32 %), we cannot say that the amount of technical debt was reduced by 32 %, or that any reduction was due to explicitly managing technical debt. Capturing all the technical debt in the project was not a goal of this study, nor was reducing technical debt, thus we cannot draw any conclusions from the decrease in the number of debt items being tracked.

5.1 Costs of Technical Debt Management

In the four sprints where the proposed technical debt management approach was applied, the project leader documented and reported all the work related to technical debt management, including the responsible person and the time spent on it. According to these reports, the costs of managing technical debt came from several different activities, including identification of technical debt items, analysis and evaluation, communication and documentation. The project leader identified these activities in vivo, i.e. they were not predefined as categories of effort. Among these activities, identification (which in this case only happened once, in the fourth sprint) refers to developers finding low-quality software artifacts deemed to be technical debt instances from the modules they were in charge of. This was not done during the sprint planning meeting, but ahead of time in a fairly ad hoc way. Analysis and evaluation refers to understanding the technical debt items, estimating their principal and interest, creating a traceability matrix and mapping them to project artifacts. Communication refers to meetings of the project leader with the developers to gather information for technical debt management. Documentation refers to updating the technical debt list according to the analysis and evaluation results, documenting the decisions related to the technical debt items and the implementation results, including the date and the actual effort of paying them off. Table 5 shows the cost of each activity in the sprints.

From Table 5 we can observe that the major cost came from analysis and evaluation work. This was confirmed, through the final interview with the project leader, that analyzing and evaluating technical debt items was the most time-consuming work. Technical debt

Table 4 Changes in the Technical Debt List over the Sprints

Items	Sprint I	Sprint II	Sprint III	Sprint IV
Total (before the sprint started)	22	20	15	15
New	0	0	0	1
Paid-off	2	5	0	1
Deferred to the next sprint	20	15	15	15

Table 5 Costs (in person-hours) of technical debt management activities over the sprints

Items	Sprint I	Sprint II	Sprint III	Sprint IV
Identification	0	0	0	0.5
Analysis and evaluation	2.8	0.3	0.4	0.3
Communication	0.2	0	0	0
Documentation	0.7	0	0.1	0
Total	3.7	0.3	0.5	0.8

Identification seems to be high-cost work as well according to the figures for Sprint IV, but this is not conclusive because only one technical debt item was identified in this sprint. In the second and third sprints, the project leader focused on the items already on the technical debt list and didn't identify new technical debt items. Hence there was no identification effort in those two sprints, but there was in Sprint IV. The initial technical debt items for this case study were identified in the preparation phase, which also involved training/learning of the proposed approach. Since it would have been difficult to capture the effort involved in preparing the initial debt list, and because such effort is not particularly relevant to technical debt management in general, we excluded the identification effort from the first sprint. The time required for identification likely varies widely depending on how the debt is identified, e.g. as part of a code review vs. using a static analysis tool. In the case study, technical debt items were identified ad hoc, by suggestion of developers, not through the use of any tools or formal processes. One could speculate that technical debt identification might take less time when using code analysis tools or as part of an existing process like a code review.

The cost numbers in Table 5 also indicate the proposed approach has a significant initial overhead. When the technical debt list was used for the first time, the project team had to go over each item to understand it and re-estimate its value. But in the subsequent sprints they only needed to examine those items that had changed. Therefore, there was a significant drop in the cost from the first sprint to the subsequent ones. But even in the first sprint, the cost of managing technical debt was only about 10 % of the cost to implement an average use case.

According to the baseline data, it took 5 person-hours on average for the project leader to plan a sprint before the case study, i.e. before starting to manage technical debt explicitly. To determine how much effort was added to the sprint planning activity by explicit technical debt management, we look to Table 5 to calculate the average effort in managing technical debt per sprint −1.325 person-hours. Since most of this effort was expended while planning each sprint, we can conclude that, after applying the technical debt management approach, the average effort devoted to the planning phase of a sprint increased to 6.325 person-hours, or a 26 % increase. Although a 26 % increase seems to be significant, 1.3 person-hours is a small number compared to the cost of implementing a use case, which is 37 person-hours on average. Moreover, this increase was largely due to the aforementioned "initial overhead". The first release cycle had a 70 % increase in sprint planning effort (3.7 h) due to technical debt management, but then the extra cost went down to just over half a person-hour per sprint, or about a 10 % overhead.

5.2 Planning Process and Decision Factors

As mentioned in Section 4.3, we collected decision data from the technical debt list as well as the interview. By analyzing these decision data, we have obtained a clear picture of how a new sprint is planned and what factors could affect the decisions about technical debt. When the project leader plans a new sprint, he first "takes into consideration the current delivery (a set of use cases to be deployed to the customer)". Besides the delivery, "there are also improvement points requested by the customer". "With this in mind", the project leader will then check other factors such as the technical debt currently in the project. In other words, the project leader always gives the highest priority to the features and enhancements agreed to by the customer. The project leader's top concern is on-time delivery. If the project leader cannot make the delivery on time, he "runs the risk of losing the customer". In this situation he would delay some work to save time even if he has to pay high interest later because "paying high interest is still better than losing the customer". This rationale accounts for the major cause of technical debt incurrence in the project where most technical debt items were given an explanation similar to "this should be done, but we do not have time in this sprint". Therefore, "time and resource availability are crucial in deciding whether to delay or pay off the debt."

If the schedule and resources allow, the project leader will "check the value of the debt items, i.e. the principal and interest." The items with high interest amount and high interest probability will be more likely to be chosen to be paid off. But the final decision is also based on a "look for the tasks in the backlog." The project leader considers "a good opportunity to pay off a debt item if it is related to some backlog item or currently being maintained by a developer" because they can hit two birds with one stone. He also gave an example in which he decided to pay off a debt item associated with the function of generating financial reports, which was being developed at that time. This strategy is consistent with the proposed technical debt management approach in that these items have reduced principal. Besides these factors, the project leader also considered the impact of debt on other features of the system. The debt items that may affect more features will be given higher priority to be paid off.

In short, our results show that the actual process of deciding whether or not to pay off technical debt was affected by many factors. Among these factors, customer expectations have the top priority, followed by availability of the development resource, the interest of the technical debt items, the current status of the debt-infected modules and the impact of the debt on other features.

One of the contributions of this case study is that it allowed us, for the first time, to observe a real software project attempt to explicitly manage technical debt. While we provided a process for technical debt management to the project team, we expected that the team would have to modify it, or use it in a slightly different way than we intended, in order to serve their needs. This turned out to be true. In the proposed approach, we assumed that all the factors that may affect the decision on paying off or deferring technical debt could be reflected in the values associated with technical debt, i.e., principal and interest. For example, if a technical debt item needs to be paid off immediately due to its criticality, the interest estimate of this debt item should be increased because if it is not paid, it will cause serious problems, and thus higher cost in future. So in our proposed approach, our intent was that the priority of each debt item could be determined using only its principal and interest. By contrast, the actual decision making process, as observed in the case study, worked differently. The values of principal and interest were first estimated without considering other factors such as team availability and module criticality. In other words, principal and interest estimation was simple and rough, and was strictly focused on effort. During decision making, then, these estimates of principal and interest were combined with other factors to determine the final priority of a debt item. This meant that, sometimes, a debt item that should have had high priority because of its principal and interest estimates was deferred for reasons that could have, but were not incorporated into the principal and interest estimates, e.g. resource availability. Thus, although we had designed our proposed approach so that all factors could be incorporated into the notions of principal and interest, and so principal and interest could then be used as the sole criteria in decision making, the development team found it preferable to simplify the estimation of principal and interest (concentrating strictly on effort), but to be more holistic in actual decision making, by taking other factors into consideration at that time.

In summary, sprint planning could be affected by many factors, and not all these factors are easily incorporated into the notions of principal and interest. Effort (either expended or saved) is generally not the most important factor. The actual process, when technical debt management is incorporated, may deviate from the one we prescribed. In spite of these differences from our proposed technical debt management approach, the approach actually used in the case study is still an example of explicit technical debt management, and thus our goal of observing this process in an actual project was achieved. The ways in which the team tailored the process provides important insight into how technical debt management should be integrated into a real software development process. In addition, the process differences provided us with the opportunity to consider more factors and improve our approach, which is one of the objectives of our larger research agenda. Eventually we expect to provide a way for managers to prioritize technical debt items along with enhancements and bug fixes.

5.3 Benefits and Impact

Although this study focuses on the costs of explicit technical debt management, it was natural that the benefits observed in this study, especially from the interview, were also collected with the cost information as they are two sides of the same issue – technical debt management.

During the entire time period of the case study, there were no big changes in the development process, the staff and the project management, except the application of the technical debt management approach. Moreover, the project leader confirmed that he didn't have any experience in technical debt management before applying our approach. Therefore we are confident that the impact and benefits observed can be mainly attributed to the application of the approach.

An important benefit of the technical debt management approach is the increased awareness of the problems that may be overlooked otherwise. When we asked the project leader about his general impression of the technical debt management approach, he stressed that it was the technical debt list that helped him realize "many things were being overlooked in the project." He also admitted that in the technical debt list "there were so many points that had not been identified early in the project and could have been better handled earlier." When we asked the project leader if any of the decisions he made would have been different without using the approach, he gave a positive answer and further explained that he "would be restricted to the problems reported by the customer without the [technical debt management] approach and would not consider code fragments that should be improved, which can often be treated together with bug fixes identified by the user". Thus the problems could have been addressed earlier to avoid serious impact. This statement is consistent with the finding from a previous retrospective study (Guo et al. 2011) in which the subject project suffered from high change cost because their decisions in the early stage didn't consider the impact of the technical debt in their project. Moreover, the estimates of the principal and interest "make it easy to understand the gravity of the problem, thus allowing a quick decision" because he had to know "how much effort a debt would cost before allocating developers to pay it off". When we asked the project leader whether the integration of the approach impacted on his sprint planning, he responded that "just the fact of thinking about using it already impacted my activities because I started, in my decisions, to consider the impacts of not paying off a debt in a sprint."

6 Conclusion

As part of the larger research project aiming at technical debt theory construction, this study focused on the cost side of technical debt management. The objective was to uncover the costs that could be incurred by the proposed technical debt management approach. To achieve this objective, we selected a live software project and applied the technical debt management approach to it. In the course of using the approach during sprint planning, technical debt items were identified, the effort spent in managing technical debt was logged, and the decisions were also documented. By tracking technical debt in the subject project, we have identified a set of technical debt management activities and measured their costs. From the cost information we have gained insight into the feasibility of the technical debt management approach. In addition, this study also helped us understand how technical debt information contributes to decision making in release planning. Along with the benefits exhibited by applying the approach, we have been able to characterize both costs (quantitatively) and benefits (qualitatively) in a way that can help future projects and organizations decide if managing technical debt is feasible for them. Meanwhile, we also noticed some limitations of this study and hence will address them in our future work.

6.1 Costs and Benefits of Explicit Technical Debt Management

According to the findings of this study, technical debt management could incur different types of costs. Among these costs, analysis and evaluation took the top position, accounting for the majority of the total management cost. Analysis and evaluation was closely tied to the difficulties of quantifying and estimating the technical debt parameters, principal and interest. Thus, future advances in technical debt measurement (such as Marinescu 2012) are likely to have an impact on the overall cost of technical debt management. Identification of new technical debt items also incurred high cost, and many current research efforts are aimed in this area (Emden and Moonen 2002; Mo et al. 2013; Wang et al. 2013).

Since these conclusions about different types of management activities are drawn based on one case, i.e. a single software project, we would not predict that technical debt management will have the same cost pattern, e.g. cost categories and proportions, if it is applied to another project with a different size or development process. However, by digging into the reasons behind these costs, we are confident that identification and evaluation costs would still be more significant than other types of cost and thus should be addressed with a high priority.

As mentioned in Section 4.3, the technical debt items in the subject project were identified manually, a time-consuming task performed by the developers who had to look into the code, compare it with the criteria, and recall what they have done in the past. This situation would be similar in projects even with different characteristics unless an effective technical debt identification approach is applied.

Likewise, evaluation of technical debt was largely based on the developers' experience. A great amount of effort is required to reach a certain level of estimation accuracy even for experienced developers, who have to take into consideration many factors such as the impact from other changes, the future change of the debt-infected software artifacts and the change likelihood. When the developer lacks experience, the evaluation will be subject to large estimation errors. This remains true for any software project, so this area requires a breakthrough in research results.

Our results show that the overall cost of technical debt management is small compared to the release planning cost. However, we believe it is still worthwhile to improve the performance of technical debt identification and measurement because all types of management costs are overhead expenses, with an only indirect effect on the value of the software being developed. Thus advances in these areas will benefit overall efficiency for all software projects, not just those that have more significant technical debt management overhead.

We didn't observe any unusual characteristics that distinguish our study setting as highly unusual, other than the basic project characteristics that we have reported, nor in the conduct of the project during the case study. Thus we can, to some extent, be assured of the generalizability of our results to similar contexts. Therefore, we believe these findings are a contribution to the field in helping to understand the impacts of explicitly managing technical debt.

The change in the total cost over sprints shows there was a significant initial overhead for technical debt management, with a much reduced cost over subsequent sprints. Because of the initial overhead, technical debt management effort in the first sprint reached 3.7 person-hours, a 70 % increase in sprint planning effort, after incorporating technical debt management. In the subsequent sprints, the average increase dropped to 10 %, which has a very minor impact on the project in terms of management cost. Even in the first sprint, the cost of managing technical debt was only about 10 % of the cost to implement an average use case. Therefore, the amount of extra project cost added by adopting technical debt management could be considered reasonable.

Release planning in practice often follows a prescribed process with factors and criteria. When technical debt management was incorporated into the process, it became a decision factor and thus affected the process and the final decision. Through this study we identified a set of decision factors and revealed their relations. For example, customer requests and availability of development resources are given higher priority than the amount of principal or interest associated with a technical debt item. Other factors include the current status of the debt-infected modules and the impact of the debt on other features. These factors and priorities helped us understand how technical debt management works in release planning and where the possible benefits of technical debt management may come from.

Although this study centers on investigating the costs of managing technical debt, we also gained insights into the benefits of explicit technical debt management. The main benefit of technical debt management is the increased awareness of the problems that are as important as those raised by the customer, but may be overlooked otherwise. Thus these problems could be identified and handled earlier to avoid turning into bigger problems in the future. The technical debt list also facilitates comparing different technical debt items, thus decision making. Because of these benefits, the project leader in our subject project decided to continue to explicitly manage technical debt, even after the case study concluded. Since the benefit data were collected from the interview and are qualitative in nature, they cannot be compared quantitatively with the cost data. However, based on the benefits and the cost insights gained through this study, we can conclude that explicit technical debt management is an affordable approach that could benefit software projects in different ways.

6.2 Limitations and Threats to Validity

The main limitation of this study is actually the limitation of any case study. The generalization power or external validity (Yin 1994), of the study is restricted by the case we selected. In this study we selected a relatively small and young software project, which may have a different cost pattern and management style than bigger mature projects. For example, the proportion of the communication cost in a big project is usually higher than a small project. Having more people involved in meetings in which technical debt is being discussed would make those discussions more expensive. Managing the technical debt list might also involve more people, and thus be more expensive. On the other hand, a more mature organization might very well have more mature cost estimation procedures, which would make technical debt management more effective. In other words, our findings reflect experience in just one particular case, the goal being transferability, not generalizability. This means that the specific findings of this study may not apply to other contexts (e.g. larger more mature projects), but the general lessons learned should be instructive to those applying and studying this approach in any situation.

As mentioned in Section 4.2, the study was carried out on the project site, which is located outside of U.S., where the principal researchers were located. The physical distance and language barrier prevented the principal researcher from having direct contact with the project team. All qualitative data had to come through and be translated by our company contact. Therefore, there might be some subtle points that were lost in translation. This is a threat to internal validity (Yin 1994), as it affects our ability to accurately explain the phenomena that we observed. The language barrier is also a minor threat to construct validity, in that we cannot be sure that the study participants completely understood technical debt and our proposed management process in the way that we intended.

Another threat to internal validity (i.e. the extent to which we can claim to be accurately characterizing the effects of technical debt management) comes from the inaccuracy of technical debt estimation. Since the value of a technical debt item (i.e. its principal and interest) determines its payoff priority, inaccurate estimation may lead to a wrong decision. In this study there were technical debt items whose estimated values deviated far from their actual values. In particular, in two cases, the estimated principal was more than three times the actual effort expended when the technical debt item was actually paid off. The members of the development team attributed these deviations to a lack of experience with effort estimation. The decisions related to these items would also have been different if the estimated values had been closer to their actual values. Although this difference didn't affect addressing the research questions, more benefits might have been observed if they had used a more effective effort estimation approach.

To help ensure the reliability (Yin 1994) of our case study, we have documented our procedures carefully, and the co-authors have checked each others' work at each stage.

6.3 Future Work

Given the limitations of this study, we plan to carry out another case study using a bigger and more mature software project, which has a very different profile in terms of project cost, development cycle, etc., to gain new insights into the costs of technical debt management and the impact on decision making. We expect that combining the results from the two case studies could yield valuable findings from which more solid conclusions can be drawn. Meanwhile, we will keep improving our technical debt management approach, especially in the aspect of technical debt prioritization so that the prioritization of a technical debt item will depend on factors in addition to the estimated value of technical debt.

In addition to the case study, we also plan to conduct more retrospective studies, which are designed to address the benefit side of technical debt management. Then the two types of studies, together, would be able to achieve the goal of our technical debt research agenda – addressing the cost-benefit relationship implicit in the technical debt metaphor to build a technical debt theory.

References

Alberts CJ et al (1996) Continuous risk management guidebook. Software Engineering Institute Carnegie Mellon University, Pittsburgh

Alter S, Ginzberg M (1978) Managing uncertainty in MIS implementation. Sloan Manag Rev 20:23–31

Bachmann F, Nord R L, Ozkaya I (2012) Architectural tactics to support rapid and agile stability. CrossTalk, May/June-2012:20–25

Boehm BW (1991) Software risk management: principles and practices. IEEE Softw 8:32

Boehm BW (1981), Software engineering economics. Englewood Cliffs, Prentice-Hall, New Jersey, USA

Bohnet J, Döllner J (2011) Monitoring code quality and development activity by software maps. In: Proceedings of the 2nd workshop on managing technical debt (MTD '11), pp 9–16

Brondum J, Zhu L (2012) Visualizing Architectural Dependencies. In: Proceedings of the 3rd workshop on managing technical debt (MTD '12), pp 7–14

Brown N, Nord R L, Ozkaya I, Pais M (2011) Analysis and management of architectural dependencies in iterative release planning. In: Proceedings of the 9th working IEEE/IFIP conference on software architecture (WICSA), pp.103-112

CAST (2012) Cast worldwide application software quality study: summary of key findings. Cast report

Charette RN (1989) Software engineering, risk analysis and management Intertext publications. McGraw-Hill Book Co, New York

Cunningham W (1992) The wycash portfolio management system. In: Addendum to the proceedings on Object-oriented programming systems, languages, and applications, pp 29–30

Dalkey N, Helmer O (1963) An experimental application of the delphi method to the use of experts. Manag Sci 9: 458–467

DoD U (2006) Risk management guide for DoD acquisition. Department of Defense, USA

Emden E V, Moonen L (2002) Java quality assurance by detecting code smells. In: Proceedings of the ninth working conference on reverse engineering, pp 97–106

Fairley R (1994) Risk management for software projects. IEEE Softw 11:57–67

Fowler M (2003) Technical debt. Web. http://www.martinfowler.com/bliki/TechnicalDebt.html. Accessed 1 May 2008

Fowler M (2007) Design stamina hypothesis. Web. http://www.martinfowler.com/bliki/DesignStaminaHypothesis.html. Accessed 1 May 2008

Fowler M (2009) Technical debt quadrant. Web. http://martinfowler.com/bliki/TechnicalDebtQuadrant.html. Accessed 1 Dec 2013

Fowler M, Beck K, Brant J, Opdyke W, Roberts D (1999) Refactoring: improving the design of existing code, Addison-Wesley

Gaudin O (2009) Evaluate your technical debt with Sonar. http://www.sonarqube.org/evaluate-your-technical-debt-with-sonar. Accessed 1 June 2014

Guo Y, Seaman C, Gomes R, Cavalcanti A, Tonin G, Da Silva F Q B, Santos A L M, Siebra C (2011) Tracking technical debt – an exploratory case study. In: Proceedings of 27th IEEE international conference on software Maintenance (ICSM'11), pp 528–531

Higuera RP, Haimes YY (1996) Software risk management. Software Engineering Institute Carnegie Mellon University, Pittsburgh

ISO (2002) Risk management - principles and guidelines on implementation. International Organization for Standardization

Kontio J (2001) Software engineering risk management: a method, improvement framework and empirical evaluation. Dissertation, Helsinki University of Technology

Lester A (2008) Get out of technical debt. Web. http://petdance.com/perl/technical-debt/. Accessed 1 May 2008

Letouzey, J-L (2012) The SQALE method for evaluating technical debt. In: Proceedings of the 3rd workshop on managing technical debt (MTD'12), pp 31–36

Leung H, Fan Z (2002) Software cost estimation. Handbook of software engineering and knowledge engineering. World Scientific Pub Co, River Edge

Marinescu R (2012) Assessing technical debt by identifying design flaws in software systems. IBM J Res Dev 56(5):1–13

McConnell S (2007) 10x software development. Web. http://forums.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx. Accessed 1 May 2008

Mo R, Garcia, J, Cai Y, Medvidovic N (2013) Mapping Architectural Decay Instances into Dependency Models. In: Proceedings of the 4th workshop on managing technical debt (MTD '13), pp 39–46

Nord R L, Ozkaya I, Kruchten P, Gonzalez-Rojas M (2012) In search of a metric for managing architectural technical debt. In: Proceedings of the joint 10th working IEEE/IFIP conference on software architecture (WICSA) and the 6th European conference on software architecture (ECSA), pp 91–100

Nugroho A, Visser J, Kuipers T (2011) An empirical model of technical debt and interest. In: Proceedings of the 2nd workshop on managing technical debt (MTD '11), pp 1–8

Parkinson CN (1957) Parkinson's law and other studies in administration. Houghton Mifflin, Boston

Putnam LH (1978) A general empirical solution to the macro software sizing and estimating problem. IEEE Trans Softw Eng 4:345–361

Rothman J (2006) An incremental technique to pay off testing technical debt. Web. http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=COL&ObjectId=11011&tth=DYN&tt=siteemail&iDyn=2. Accessed 1 May 2008

Schmid K (2013) A formal approach to technical debt decision making. In: Proceedings of the 9th international ACM Sigsoft conference on quality of software architectures (QoSA'13), pp 153–162

Seaman C, Guo Y (2011) Measuring and monitoring technical debt. Adv Comput 82:25–46

Shepperd M, Kadoda G (2001) Comparing software prediction techniques using simulation. IEEE Trans Softw Eng 27:1014–1022

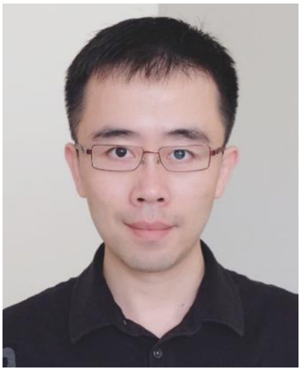
Sisti FJ, Joseph S (1994) Software risk evaluation method. Software Engineering Institute Carnegie Mellon University, Pittsburgh

Stamatelatos M (2002) Probabilistic risk assessment procedures guide for nasa managers and practitioners. NASA

Wang P, Yang J, Tan L, Kroeger R, Morgenthaler J D (2013) Generating precise pependencies for large software. In: Proceedings of the 4th workshop on managing technical debt (MTD '13), pp 47–50

Yin RK (1994) Case study research: design and methods, 2nd edn. Sage Publications, Thousand Oaks

Zazworka N, Seaman C, Shull F (2011) Prioritizing design debt investment opportunities. In: Proceedings of the 2nd workshop on managing technical debt (MTD '11), pp 39–42



Yuepu Guo is a PhD candidate in the Department of Information Systems at University of Maryland Baltimore County (UMBC). He received the BS degree in industrial economics from Tianjin University, Tianjin, P. R. China, in 1996, the MS degree in information systems from University of Maryland Baltimore County, Baltimore, USA, in 2009. His research interests include software process, maintenance and evolution. In particular, he is interested in and currently investigating issues of technical debt in software lifecycle, part of which forms his doctoral dissertation. He can be reached at yuepu.guo@umbc.edu

Rodrigo Oliveira Spínola is Senior Scientist at Fraunhofer Project Center for Systems and Software Engineering at Federal University of Bahia and Professor of Computer Science at the Salvador University where he leads the Technical Debt Research Team (tdresearchteam.com). Prof. Spínola holds a D.Sc. and a M.Sc. in System Engineering and Computer Science from the Federal University of Rio de Janeiro. From 2011 to 2012, he was a visiting researcher at the University of Maryland Baltimore County (UMBC) and the Fraunhofer Center for Experimental Software Engineering (CESE). His main research interests are on software evolution, technical debt, and ubiquitous computing.

Carolyn Seaman is an Associate Professor of Information Systems at the University of Maryland Baltimore County (UMBC). Her research generally falls under the umbrella of empirical studies of software engineering, with particular emphases on maintenance, organizational structure, communication, measurement, technical debt, and qualitative research methods. She is also a Research Fellow at the Fraunhofer Center for Experimental Software Engineering, Maryland, where she participates in research on experience management in software engineering organizations and software metrics. She holds a PhD in Computer Science from the University of Maryland, College Park, a MS in Information and Computer Science from Georgia Tech, and a BA in Computer Science and Mathematics from the College of Wooster (Ohio). She has worked in the software industry as a software engineer and consultant, and has conducted most of her research in industrial and governmental settings (e.g. IBM Canada Ltd., NASA, Samsung, Xerox).