Integrating Technical Debt Management and Software Quality Management Processes: A Normative Framework and Field Tests

Narayan Ramasubbu, *Member, IEEE Computer Society*
Chris F. Kemerer, *Member, IEEE Computer Society*

Abstract—Despite the increasing awareness of the importance of managing technical debt in software product development, systematic processes for implementing technical debt management in software production have not been readily available. In this paper we report on the development and field tests of a normative process framework that systematically incorporates steps for managing technical debt in commercial software production. The framework integrates processes required for technical debt management with existing software quality management processes prescribed by the project management body of knowledge (PMBOK), and it contributes to the further development of the software-specific extensions to the PMBOK. We partnered with three software product development firms at different process maturity levels to implement and test the framework in real-world software production. In terms of impact across the three firms the process framework contributed to an average 19% reduction in defects, which resulted in a net 43% reduction in technical debt-related failure costs after accounting for the additional process overhead. Overall, through the adoption and use of the process framework, the firms in the field test were able to integrate the processes necessary for technical debt management with their existing software quality management processes and accrue significant economic benefits.

Index Terms—Technical debt, software quality, software maintenance, software engineering economics, cost of quality, software product development, software process, software extension to PMBOK, field study

1 Introduction

THE importance of managing technical debt embedded in software products has been highlighted by several recent studies in the empirical software engineering literature [e.g., 1,2,3,4,5,6,7]. Technical debt, defined as the maintenance obligations arising from shortcuts taken during the design, development, and deployment of software systems [8,9], has been shown to significantly impact the reliability and long-term evolution of software systems [1,2,3,10]. Although academic research has moved beyond using technical debt only as a metaphor, and has compiled strong empirical evidence on the economic implications of technical debt, industry practitioners continue to find managing technical debt a challenging balancing act [2,11,12,13]. Both academic scholars and industry consultants have called for the development of normative frameworks and tools that help practitioners to systematically identify technical debt and assess the economic consequences of technical debt [14,15,16].

In this paper we respond to the above call and present a normative process framework for managing technical debt in commercial software product development. A salient feature of our framework is that it integrates the processes proposed for technical debt management with the well-established and widely-adopted software quality management process standards prescribed by the Project Management Body of Knowledge (PMBOK) guide and the software extension to the PMBOK that was jointly developed by the Project Management Institute (PMI) and the IEEE Computer Society. There are three main benefits from this integrated approach:

1. It enables uncovering of hidden technical debt embedded in systems. For example, established quality assurance and control practices such as orthogonal defect classification [17], cause and effect mapping [18], Pareto analysis [19,20], and capture-recapture techniques [21] can all be utilized to effectively associate software defects with specific design and deployment decisions made by programmers. Such associations make technical debt visible to the team and, thereby, facilitate the quantification of debt-related principal and interest [5,9].
2. It helps to bridge the gaps that exist between the technical and economic assessments of technical debt, which have been recognized as a key challenge in managing technical debt [1,9,12,14]. A tighter integration between technical debt management steps and established software quality control and assurance processes would enable practitioners to more effectively track the costs and benefits of technical debt akin to the existing best practices that help to optimize the cost of software quality [22,23,24,25].
3. It facilitates the wider adoption and continued use of technical debt management processes by

- *N. Ramasubbu is with the Joseph M. Katz Graduate School of Business, University of Pittsburgh, Pittsburgh, PA 15260. E-mail: narayanr@pitt.edu.*
- *C.F. Kemerer is with the Joseph M. Katz Graduate School of Business, University of Pittsburgh, Pittsburgh, PA 15260. E-mail: ckemerer@katz.pitt.edu.*

firms. Software quality management practices are well established and come under the purview of widely-adopted standards, such as the IEEE standard for software quality assurance processes (730-2014) and ISO/IEC system and software quality models (25010:2011). This has enabled normative quality frameworks to be widely adopted by industry practitioners, and firms use them to benchmark their practices and gain relevant certifications and industry-wide recognitions [26,27,28]. Thus, integrating technical debt management processes with the established quality frameworks can be expected to help industry practitioners to more easily adopt the prescribed steps and institutionalize them within their firms.

To test the effectiveness of the integrated process framework that we developed for aiding technical debt management we partnered with three commercial firms to conduct field tests. The three firms varied in their software development process maturity levels, but they all had a strong interest in better managing the technical debt embedded in their products.

The results from the field tests showed that our process framework helped the firms to reduce technical debt-related defects by 19% on average over a three-year period. Although implementing the processes prescribed by the framework increased the quality appraisal costs at the firms by 13% on average, the firms reaped benefits that significantly exceeded those costs. For example, technical debt-related failure costs at the firms were reduced by 43% on average, and the firms achieved an overall cost reduction of 14% per product release cycle on average.

In the following sections of the paper we provide the details of the integrated process framework, its field tests, and the implications we draw from the results of the field tests. In Section 2 we present the details of the processes we propose for technical debt management and their relationship with established software quality management processes. In Section 3 we enumerate the field test procedures and describe how they were implemented at the three research sites. Field study results are discussed in Section 4, and we conclude the paper in Section 5 with a discussion of the implications for research and practice as well as the potential extensions that can be pursued based on this study.

2 A Normative Process Framework for Managing Technical Debt

In this section we begin with an overview of the proposed process framework and then explain how the integration between technical debt management and quality management processes is achieved. The different components of the process framework are discussed to provide an overarching view, leaving out the specific implementation details at the research sites, which are fully described in Section 3.

2.1 Three-Step Process

As shown in Figure 1, we organize the different processes for technical debt management under three broad steps:

1. Make technical debt visible
2. Perform cost-benefit analysis
3. Control technical debt

To enact each of the above three steps, the framework considers specific *inputs*, *tools and techniques*, and *outputs*. This is similar to the organization of the various project management practices for each knowledge area covered by the PMBOK.

Step 1: Make Technical Debt Visible. This step involves the processes for identification and continuous tracking of technical debt. Since information pertaining to technical debt, including the various shortcuts taken by teams, the business and technical antecedents to those decisions, and their causal implications is often not readily available, or is distributed in complex ways across multiple artifacts and stakeholders, a systematic approach is needed to uncover and trace technical debt. While existing research has mainly focused on technical artifacts, such as source code, for identifying and measuring technical debt [3,6,7,29,30], we propose to expand the identification strategy to accommodate other organizational assets, including stakeholder views, risk exposure, and quality control data. This expansion should help teams to rigorously gather information and then estimate the economic implications of their technical decisions related to technical debt [2, 10]. Thus, as shown in Figure 1, we consider a broad set of *inputs* for identifying technical debt beyond only software assets, including information present in stakeholder registers, risk registers, defect tracking databases, and other organizational process assets used in software production [17,19,22,24,25].

Correspondingly the *tools and techniques* to assess the inputs go beyond the source code analyses discussed extensively in current technical debt research [3,29,30] and include broader quality and risk management techniques. This expanded toolkit includes root cause analysis methods such as cause-and-effect diagrams [10,18], orthogonal defect classification schemes [17], and fundamental cost of quality methods such as Pareto analysis [19,20] and statistical quality control methods [22,23,26].

The *outputs* of Step 1 that are related to the traditional software quality processes are captured as updates to the existing quality management plans and associated quality standards and metrics. To capture the technical debt-specific information identified in Step 1 we introduce a new artifact, called the *technical debt register,* which stores, for each software asset, the outstanding principal and associated interest estimated for the technical debt embedded in the asset. The technical debt register also stores the desired *control target* for each software asset's technical debt, which is populated during the cost-benefit analysis calculations discussed later in Step 2. Figure 2 elaborates the relationship of the new technical debt register with existing process assets used in commercial software production.

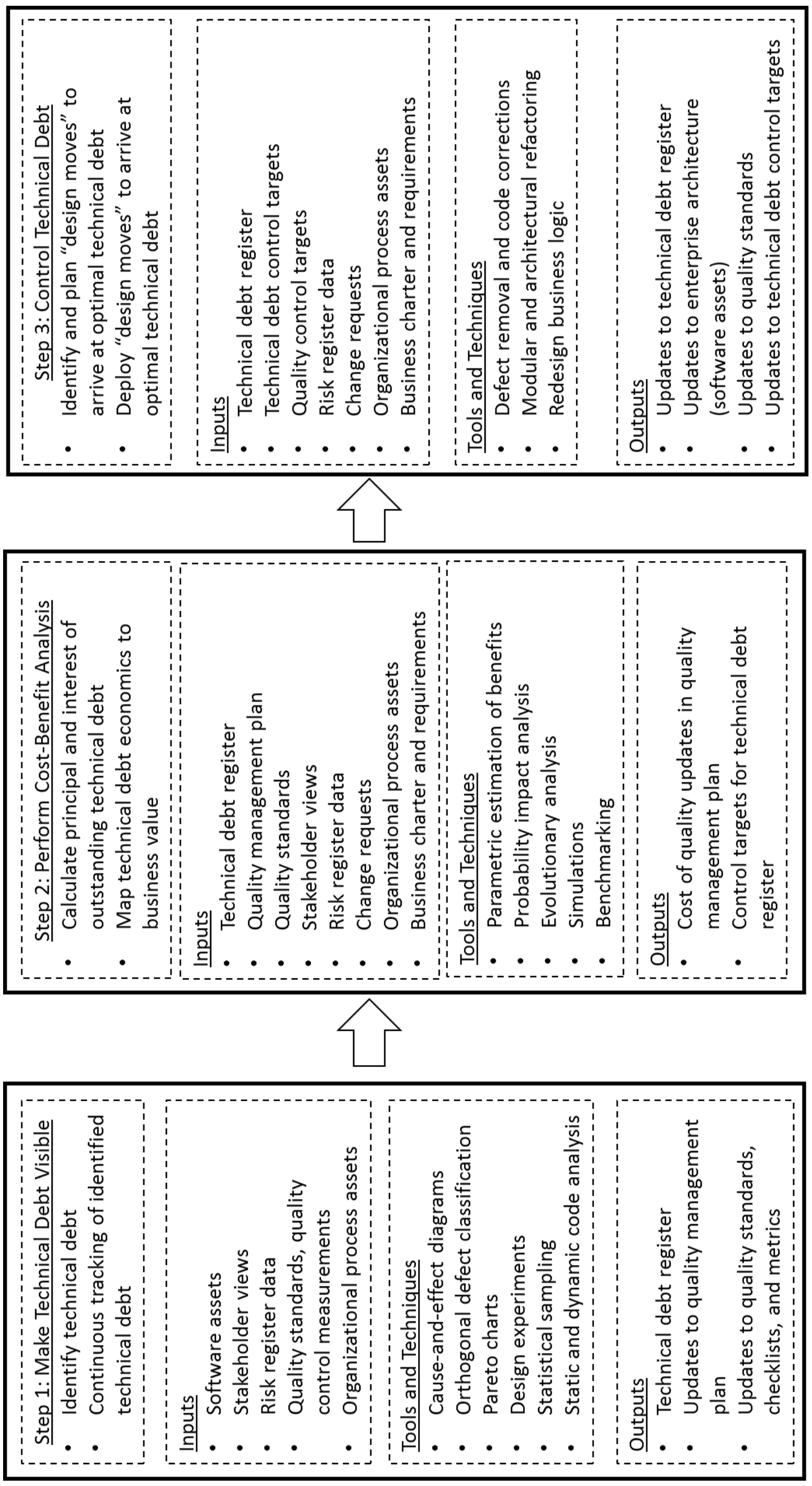

Step 1: Make Technical Debt Visible
• Identify technical debt
• Continuous tracking of identified technical debt
Inputs
• Software assets
• Stakeholder views
• Risk register data
• Quality standards, quality control measurements
• Organizational process assets
Tools and Techniques
• Cause-and-effect diagrams
• Orthogonal defect classification
• Pareto charts
• Design experiments
• Statistical sampling
• Static and dynamic code analysis
Outputs
• Technical debt register
• Updates to quality management plan
• Updates to quality standards, checklists, and metrics
Step 2: Perform Cost-Benefit Analysis
• Calculate principal and interest of outstanding technical debt
• Map technical debt economics to business value
Inputs
• Technical debt register
• Quality management plan
• Quality standards
• Stakeholder views
• Risk register data
• Change requests
• Organizational process assets
• Business charter and requirements
Tools and Techniques
• Parametric estimation of benefits
• Probability impact analysis
• Evolutionary analysis
• Simulations
• Benchmarking
Outputs
• Cost of quality updates in quality management plan
• Control targets for technical debt register
Step 3: Control Technical Debt
• Identify and plan "design moves" to arrive at optimal technical debt
• Deploy "design moves" to arrive at optimal technical debt
Inputs
• Technical debt register
• Technical debt control targets
• Quality control targets
• Risk register data
• Change requests
• Organizational process assets
• Business charter and requirements
Tools and Techniques
• Defect removal and code corrections
• Modular and architectural refactoring
• Redesign business logic
Outputs
• Updates to technical debt register
• Updates to enterprise architecture (software assets)
• Updates to quality standards
• Updates to technical debt control targets

Fig. 1. An overview of the integrated process framework for managing technical debt. Each of the three steps, make technical debt visible, perform cost-benefit analysis, and control technical debt, are enacted in conjunction with the software quality management processes followed in commercial software production. The inputs, tools and techniques, and outputs of the three steps are organized in a way similar to the process descriptions of the ten knowledge areas covered in PMBOK.

Figure 2 is a conceptual schema used to illustrate how the new process registers introduced for the purpose of technical debt management are integrated with other commonly used registers for requirements management, defects management, and risk management in software production. It should be noted that Figure 2 does not show the actual entity-relationship diagrams of a process database, which would vary depending on the implementations at different research sites.

In commercial software production the *requirements register* holds information on product requirements and links them to both the business objectives and to specific deliverables that satisfy them, and it helps stakeholders to have a global view with the ability to trace project objectives [31,32]. As shown in Figure 2, we extend the mapping of the business needs, specific software assets and their business values provided by traditional requirements registers to also include the construct of *design moves*. In this context a design move is a discrete strategic action performed on a software asset to, for example, enhance its functionality, alter modularity, or refactor with the goal of reducing technical debt embedded in the software asset [10]. Introducing the design move column into the requirements register helps us to track the impact of the objectives and actions of a team on software assets at a fine-grained level that is more suitable to accurately trace the evolution of technical debt in a system over its lifecycle. For example, the *design moves register* can help in keeping track of the changes in a software asset's degree of compliance with established design and programming standards across different product releases. While the design moves register shown in Figure 2 tracks the accumulation (or, conversely, depreciation) of technical debt as a result of a team's actions, the technical debt register serves to provide a *cumulative* snapshot of the state of technical debt per software asset along with the associated control targets desired by the team. Thus, the technical debt and design moves registers work together to improve the visibility of technical debt to stakeholders, and facilitate a process mechanism to measure and keep track of technical debt at both strategic and operational levels.

Finally, as shown in Figure 2, the technical debt and design moves registers can be linked to other software engineering process databases such as the *defect register* and the *risk register* for achieving a tighter integration between data related to the technical debt management and the quality management processes of a firm. The link between the technical debt and defect registers enables teams to assess the quality status of a software asset in a more reliable and complete manner by taking into account both the asset's defects backlog and the technical debt embedded in the asset. Similarly, the link between the design moves register, defect register, and risk register facilitates the estimation of risks of design actions that are enacted on software assets. Thus, we expect the proposed integration of different process registers as illustrated in Figure 2 to help teams to see the full range of cost and benefits scenarios as described next.

Step 2: Perform Cost-Benefit Analysis. This step addresses the need to consider the economic implications of accumulating or depreciating technical debt. Recent research has highlighted the benefits of considering technical debt management in software products with long lifecycles as an optimization problem [1,2], which is also aligned with practitioners' perceptions of it as a complex balancing act [13,14]. While a low technical debt approach is commonly perceived as superior, accumulating technical debt under some circumstances could be beneficial, for example, when software teams desire a quick rollout of functionality to attract early adopters [1,10]. Similarly, accumulating technical debt in circumstances where the risks associated with technical debt could be transferred to other players in the software product ecosystem also highlights the potential strategic benefits that stem from accumulating technical debt [10]. Moreover, depreciating technical debt with an aim to lower risk exposures may not be straightforward and/or beneficial [2,10]. Thus, researchers have prescribed a careful consideration of both the costs and the benefits of technical debt in order to adopt a contingency-based approach to manage technical debt [1,9,10,12,13,14]. To facilitate such a cost-benefit scenario analysis, our framework considers a range of inputs, including data from the technical debt register and other connected process registers described in Step 1. In addition, any change requests and business requirements proposed by stakeholders need to be taken into account for the scenario analysis. To operationalize the cost-benefit scenario planning we prescribe a longitudinal analysis that examines the evolution of the software assets over the planning horizon considered suitable by stakeholders. Although a longitudinal analysis is more complex and data intensive, it is essential to rigorously assess both the short-term and long-term benefits and costs associated with technical debt. During this analysis there is a need to extrapolate and forecast future long-term benefits and costs based on current situations encountered by teams. For example, engineering actions aimed at reducing technical debt of a software product may not be profitable if a firm decides to prematurely retire the product due to other business reasons. Therefore, and akin to prior research, we anticipate the need to involve multiple stakeholders and employ a combination of approaches such as expert judgment, heuristics, and probabilistic analyses [1,2,10,33]. Here we draw motivation from existing approaches to risk management and prescribe benchmarking, simulations, and probability impact analysis as appropriate techniques [34,35]. The outputs from cost-benefit analysis yield the appropriate *control target* for technical debt of the software assets considered, and, once known, must be incorporated into the technical debt register. As we consider technical debt as an integral dimension of the long-term total cost of quality of software products we recommend that the control targets for technical debt and software quality be synchronized. Hence, we propose coordinating the updates to the technical debt register, defect register, and quality management plans.

Step 3: Control Technical Debt. Once technical debt is made visible (Step 1) and appropriate targets are established for the desired range of technical debt (Step 2),

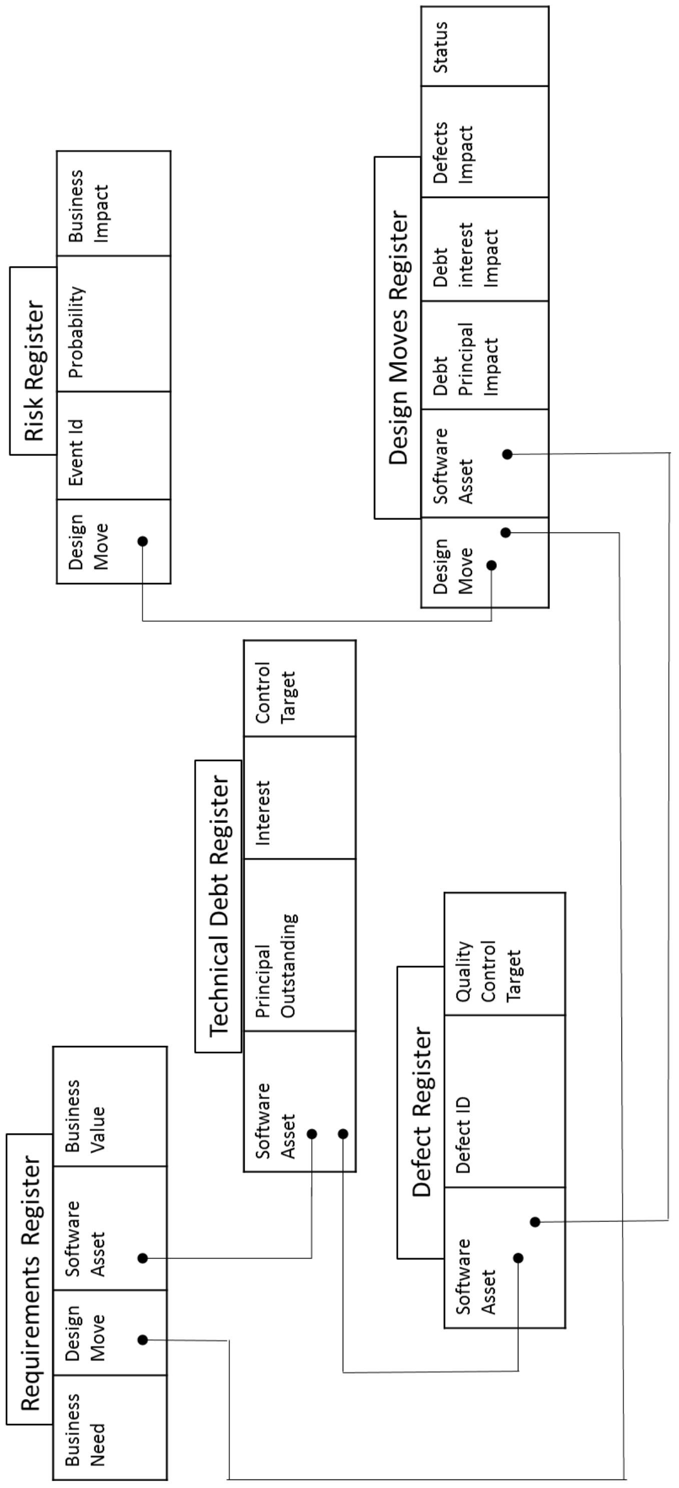


Fig. 2. Process registers used to manage technical debt. The relationships shown here form a conceptual schema, used only to situate the technical debt register in relation to the other widely used process databases in commercial software production. Fig. 2. does not depict the entity-relationship diagrams of any process registers, which would vary depending on the implementation scenarios.

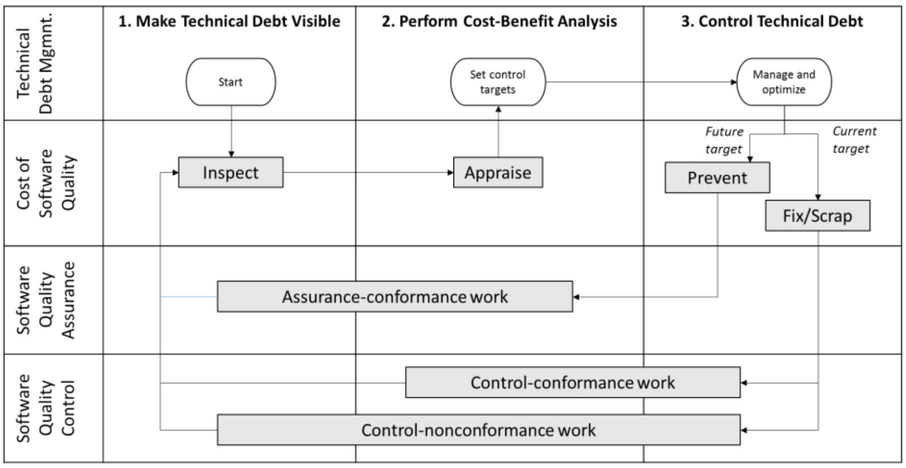


Fig. 3. Relation between technical debt management steps, software quality assurance and software quality control processes, and cost of software quality. The figure illustrates how the general flows of activities relate to each other; specific schedule-level alignment between the technical debt management cycle and the quality management cycle would vary depending on product release cycles and business context.

teams can start enacting control actions on the identified technical debt. Restructuring technical debt embedded in systems often involves a series of system changes that might involve both architecture-level and module-level alterations [2,10,14]. As noted before, we term these discrete actions as *design moves* that collectively help to bring the total technical debt of software assets within the targeted control ranges (derived in Step 2). Some examples of design moves are: refactoring to reduce software entropy, replacing legacy code, reducing complex coupling instances within subparts of a system, altering business rules and business logic, and removing design inconsistencies [10,30]. We conceive controlling technical debt as a dynamic process involving multiple design moves over the lifecycle of a system, which is analogous to a continuous statistical quality control process. The overarching goal is to sustainably maintain the technical debt of software assets involved in product development within a desired control range. Corrective actions, in the form of design moves, are enacted on software assets when the target control ranges are breached. The control ranges themselves are also periodically assessed as outlined in Step 2.

Figure 3 shows how the overall flow of steps in technical debt management generally relate to the PMBOK software quality management processes. While there is a sequential precedence from Step 1 to Step 3, it is possible to have multiple iterations of the sequence within a product release cycle. The precise alignment of schedules between the technical debt management process cycle, quality management process cycle, and product release cycle can vary depending on the business and software development context of the sites implementing our integrated framework. We discuss more on these and other implementation-specific variations in Section 3.

Typically, software quality management steps are categorized into *assurance-related* and *control-related* processes, and the total cost of software quality is split into *costs related to conformance work* (prevention, inspection, and appraisal costs) and *costs related to nonconformance work* (rework or failure costs) [22]. In Figure 3 we map those to our three proposed technical debt management steps: make technical debt visible, perform cost-benefit analysis, and control technical debt. Making technical debt visible and performing cost-benefit analysis steps contribute to the inspection and appraisal quality costs, respectively. Controlling technical debt can contribute to either conformance or nonconformance costs depending on the nature of design moves enacted. If the design moves enacted to optimize technical debt do not alter user-perceived software quality, then the costs of enacting the design moves are categorized as conformance costs [23,24]. In contrast, if the design moves alter user-perceived software quality, their costs are categorized as nonconformance quality costs, which would draw additional scrutiny from quality control personnel to assess if the design moves detrimentally alter the overall cost of quality planned for a product release cycle. Thus, the proposed framework provides a strong integration between the established software quality management processes in commercial software production and our new proposed steps for managing the technical debt of software assets.

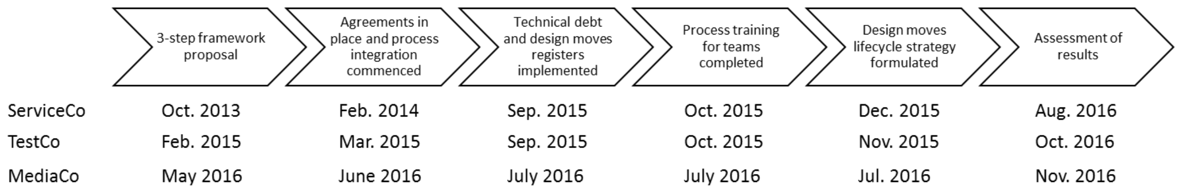
3 Research Sites and Field Test Procedures

To test the real-world usefulness of the proposed normative framework we partnered with three independent software organizations to implement the processes prescribed by the framework in their commercial software production. Similar to the action research and contextual approaches adopted by prior software studies [36,37], we collaborated closely with the three firms so that the process framework was used in real-world software production activities by the three firms. Table 1 provides an overview of these three field test research sites. The three sites varied in their business focus, software process maturity level, size, and age, and this variation provides an appropriate platform to assess the effectiveness of the normative framework over a

TABLE 1
OVERVIEW OF FIELD RESEARCH SITES

	ServiceCo[ǂ]	TestCo[ǂ]	MediaCo[ǂ]
Business Environment	Banking and insurance division of a Fortune 500 technology firm	Telecommunication test and measurement division of an electronic equipment manufacturer	Digital marketing product development division of a media firm
Software Process Maturity	CMMI Level 5	CMMI Level 3	Not assessed
Employees	100,000	4000	100
Revenues	US$ 10 Billion	US$ 1.25 Billion	US$ 3 Million
Product Age	17 years	7 years	2 years

[ǂ] *Names of the firms have been anonymized to adhere to the nondisclosure agreements signed with the companies.*



	3-step framework proposal	Agreements in place and process integration commenced	Technical debt and design moves registers implemented	Process training for teams completed	Design moves lifecycle strategy formulated	Assessment of results
ServiceCo	Oct. 2013	Feb. 2014	Sep. 2015	Oct. 2015	Dec. 2015	Aug. 2016
TestCo	Feb. 2015	Mar. 2015	Sep. 2015	Oct. 2015	Nov. 2015	Oct. 2016
MediaCo	May 2016	June 2016	July 2016	July 2016	Jul. 2016	Nov. 2016

Fig. 4. Field study timeline at the three research sites.

range of real-world situations. As would be expected, implementing the framework did increase process costs at the field sites, and these additional costs are factored into the economic results that we present in Section 4. The timeline of the various implementation steps at the field sites is shown in Figure 4.

3.1 ServiceCo

ServiceCo is a Fortune-500 technology firm with a diversified business. In October 2013 we entered into a research partnership with the banking and insurance business unit of the firm. The overarching goal of the research collaboration was to improve the business unit's capabilities to manage the technical debt embedded in its flagship software product that had evolved over 17 years. The business unit was assessed as operating at CMMI level 5[1] process maturity and had implemented a comprehensive statistical quality control regime for its software production since 2004. However, the cost of quality metrics considered by the firm did not explicitly track the technical debt burden at that time. In 2013, at a workshop conducted by the software engineering process group (SEPG) of the firm, we presented insights from our prior research on technical debt [1,2] along with a preliminary version of the normative framework (Figure 1). Subsequently, the firm launched an internal process initiative to develop organization-wide policies for managing technical debt.

As a part of this initiative the banking and insurance business unit of the firm agreed to conduct field tests of our framework and share data with us for research purposes. With nondisclosure agreements in place, the implementation of the process framework at ServiceCo was initiated in February 2014. The existing process databases at the firm already included the requirements, defect, and risk registers. However, as part of our field test and framework implementation, those registers had to be modified and integrated with the newly proposed technical debt and design moves registers. The introduction of design move as an important entity in the existing registers was a complex task as the firm had collected more than 10 years of software engineering process data in those registers and chose to port the data covering the entire lifespan of the banking and insurance software product into the new registries established for the field study. The legacy data spanning disparate product release cycles over ten years was reformulated to fit with the new schema involving technical debt and design move registers. Although this task was effort intensive and slowed down the infrastructure setup, the firm saw this as an opportunity to learn from its organizational memory. The exercise was completed after 18 months of effort in September 2015. Training for the SEPG and development personnel overlapped with the registry setup effort and was completed in October 2015.

With the entire infrastructure in place, ServiceCo product development teams started using the process framework proposed by our study for their release cycles in November 2015. A complete roadmap of the design moves planned for two product release cycles was pub-

[1] Capability Maturity Model Integration (CMMI) is a software process improvement model that is used to appraise the maturity of processes employed by software firms [38]. Level 5 is the highest level in the model representing a very high level of maturity that includes capabilities for causal analysis and resolution of issues as well as superior organizational performance management. Prior research has shown the benefits of higher CMM levels, including a reduction in high severity software defects [39].

lished by the firm a month later. To assess the effectiveness of the framework in aiding technical debt management, we utilize the results pertaining to these two release cycles that occurred during the period between November 2015 and August 2016.

3.2 TestCo

TestCo provides test and measurement solutions for telecommunication firms and was going through an organization-wide process revamp when we began collaborating with them for this study. The firm was in the process of implementing CMMI level 3[2] processes for software product development in February 2015, and we proposed to integrate the normative framework we had developed with the new processes being put in place. A product division within the firm that specialized in near field communication (NFC) technologies agreed to host our field study, and implemented the required infrastructure components by September 2015.

Even before the firm adopted CMMI level 3 processes individual product teams had a strong quality culture and followed statistical quality control approaches to continuously improve their products. However, the firm lacked a centralized process database as the product teams worked autonomously on their process initiatives. As part of the CMMI level 3 initiative at the firm a centralized process database comprised of the five registers shown in Figure 2 was set up. Thus, unlike the ServiceCo case, the infrastructure setup was relatively easy, since the technical debt and design move registers were instituted along with the requirements, defect, and risk registers at the same time, and there was no need to accommodate historical data. Another distinguishing feature was that the TestCo product development teams were smaller than the ServiceCo teams and utilized a model of product development that was similar to the Scrum model [40]. TestCo utilized Scrum masters for process governance, and product owners managed the releases, customer interaction, and requirements backlogs.

When our field study started in October 2015 the product offered by the NFC division of TestCo was 7 years old. Despite the high 'clock speed' of the test and measurement industry, the firm expected the market conditions to be supportive of the ageing product and wished to pursue active development of the product for at least another five years. Thus, the product development team members, Scrum masters, and product owners were keen to deploy an effective technical debt management framework to help them manage the product's continued evolution. We obtained data from four product release cycles over a twelve month period for testing the effectiveness of our process framework.

3.3 MediaCo

MediaCo is a startup firm which is a subsidiary of a large marketing and advertising agency. The firm developed a digital marketing platform that was in its second year of evolution after initial release in 2014. We had worked with the parent firm as part of another research project [10], and in March 2016 the product owner of the digital marketing platform invited us to prescribe a framework for managing technical debt for her new product development team. By this time we had already implemented the necessary infrastructures for the field study at ServiceCo and TestCo, and had arrived at the final versions of the normative process framework and schema for the various process registers shown in Figure 1 and Figure 2 respectively. However, implementing the infrastructure for our field study at MediaCo proved more challenging than the product owner and we had anticipated. In the end, though, they were able to successfully implement the framework, aided by the strong support of the product owner.

Product development teams at MediaCo followed an agile development approach and tracked requirements using disparate spreadsheets owned by individual teams. Defect reporting and tracking was accomplished through a web-based portal and discussion forum and, therefore, was more centralized. While those mechanisms were relatively rudimentary compared to the processes at ServiceCo and TestCo, we were still able to gather the necessary metrics required for our field study from the spreadsheets and defect-tracking portals. However, gathering information for the risk register proved to be challenging as the firm did not typically connect technical and business-related assessments of risks in its day-to-day operations. The process framework we proposed to the firm and report in this study required identifying the risks for each design move and quantifying the business impact associated with those risk events. To be able to enact this on a day-to-day basis, MediaCo had to reorganize its agile teams and product development procedures to involve business development personnel with risk management expertise in product team meetings. Given the startup environment, product development teams were initially less enthusiastic about the change to emphasize risk management. However, with the continued support of the product owner the teams agreed to implement the proposed processes as an experiment. All the process registers shown in Figure 2 became operational at MediaCo in July 2016, and we were able to utilize data from four product release cycles lasting until November 2016 for our analysis.

4 Results

For each of the three field sites we tracked cost of quality metrics that were related to technical debt at the lowest practical granularity level, including defects, prevention, appraisal, and failure costs. [22]. To adjust for size and scale effects, we used defects per kilo lines of code (KLOC) for comparison purposes, and all quality costs were normalized using percentage of overall release effort[3]. We performed within and cross-site comparisons

[2] CMMI level 3 deals with a set of process management capabilities that help a firm to achieve organizational process focus and improve in areas such as risk management, service continuity, and integration of work flows, incident resolutions, and performance management.

[3] Since our analysis involves within-case comparisons of defect density values before and after implementation of the process framework, variations in the use of programming languages across the three firms is not a concern [41]. When performing cross-case analysis, we examine the extent of benefits realization at each site and draw attention to firm-level

of those metrics to assess the impact of the adoption and use of our normative process framework. In all cases, we assessed the overall economic impact of the process framework use by measuring and evaluating overall risk exposure levels and the cost reduction (or increase) per product release cycle. Similar to prior research [2], risk exposure due to technical debt is derived using a probabilistic approach by taking into account both the business impacts (in USD) of technical debt-induced software errors and the probabilities of the occurrences of such errors. Similar to prior research [2,19,20], we utilized the historical error distribution patterns in the products we observed, and predicted for each software error type recorded at our sites the required bug-fixing and rework effort necessary to address customer-reported issues. Using those probabilistic predictions, we derived the business impact of software errors and the corresponding monetary value of risk exposures (see [2] for a detailed explanation of these procedures).

4.1 ServiceCo Results

At ServiceCo the cost of quality metrics were available at module-level granularity. Figures 5-8 present the results pertaining to the six modules that we observed during the field study. As Figure 5 shows, appraisal costs at ServiceCo increased, as would normally be expected, which reflects the cost of performing the additional processes prescribed by the framework. Statistical tests[4] comparing the before and after framework implementation scenarios indicate that appraisal costs increased by 8.2% of release cycle effort because of the framework implementation. In

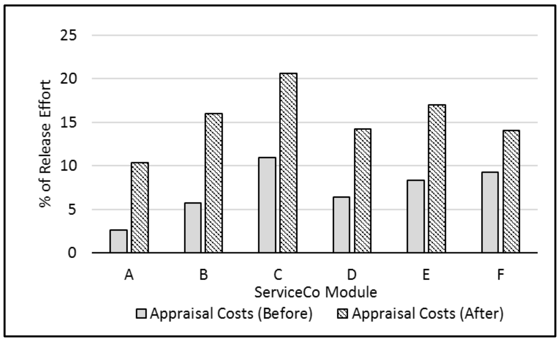


Fig. 5. Appraisal costs at ServiceCo before and after process framework adoption and use.

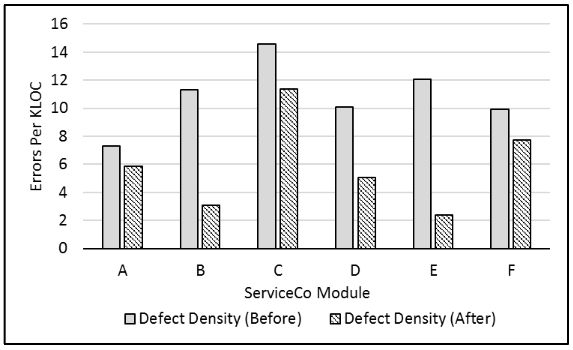


Fig. 6. Technical debt-related errors at ServiceCo before and after process framework adoption and use.

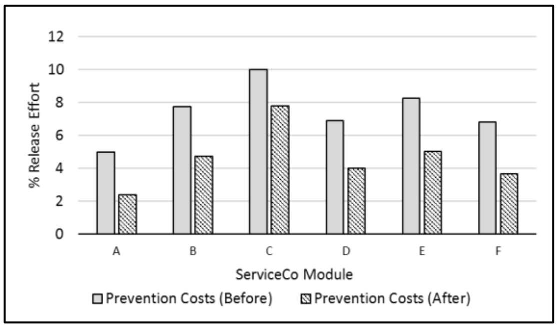


Fig. 7. Prevention costs at ServiceCo before and after process framework adoption and use.

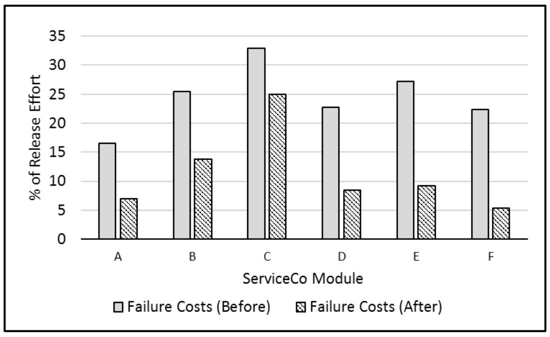


Fig. 8. Failure costs at ServiceCo before and after process framework adoption and use.

contrast, and positively, technical debt-induced errors, prevention costs, and failure costs decreased after the implementation of the process framework as shown in Figures 6, 7, and 8 respectively; these trends reflect the benefits accrued at ServiceCo. Prevention and failure costs decreased by 2.8% and 13.1% respectively. Also, errors related to technical debt decreased by 5 per KLOC. Overall, the increase in appraisal costs was smaller than the other benefits accrued, resulting in a net economic benefit of risk exposure reduction of $367,000, and an overall net cost reduction of $96,625 per product release. Thus, we conclude that the benefits of implementing the normative process framework clearly outweighed its costs at ServiceCo.

4.2 TestCo Results

During our field study at TestCo the product development teams were broadly organized into two groups pertaining to the two modules we observed (client side and server side development). We were able to track and compare the cost of quality metrics at the module level for both of these development teams. Figures 9-12 present the results for the two modules. As the figures show, costs of performing the additional processes prescribed by the framework are reflected in the increase in appraisal costs after the framework implementation (Figure 9). The offsetting benefits are reflected in the decrease of technical debt-induced errors, prevention costs, and failure costs after the implementation and use of the process framework as shown in Figures 10, 11, and 12 respectively. Similar to the tests reported for the ServiceCo case, we performed statistical tests by comparing the "before" and "after" framework implementation scenarios. Those tests

factors listed in Table 1.

[4] All statistical tests reported in the paper use two-tailed mean comparisons, checking for statistical significance at p<0.5 level.

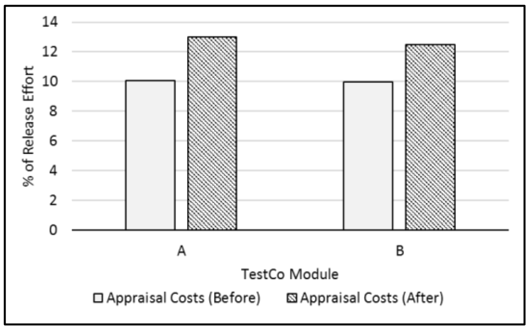


Fig. 9. Appraisal costs at TestCo before and after process framework adoption and use.

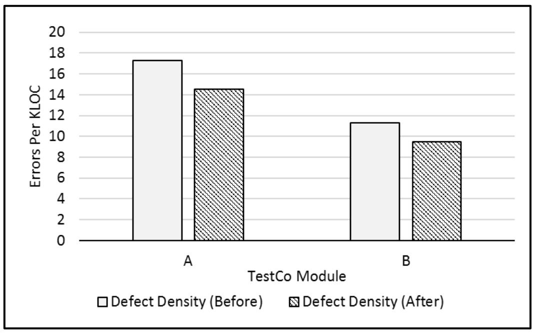


Fig. 10. Technical debt-related errors at TestCo before and after process framework adoption and use.

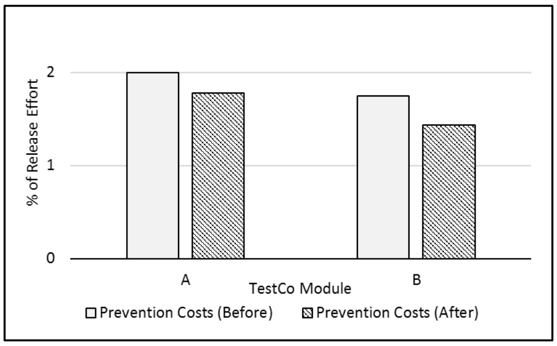


Fig. 11. Prevention costs at TestCo before and after process framework adoption and use.

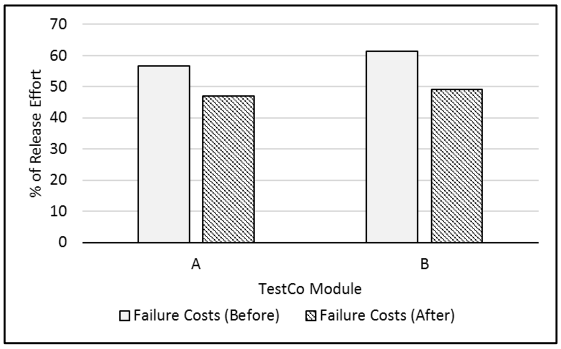


Fig. 12. Failure costs at TestCo before and after process framework adoption and use.

indicated that appraisal costs at TestCo increased by 2.7% of release cycle effort as a result of the framework implementation. However, the failure costs decreased by 10.9%. Also, errors related to technical debt decreased by 2.3 per KLOC on average. The impact on prevention costs at TestCo was minimal (a 0.3% decrease) and not statistically significant. Associated with the use of our process framework TestCo was able to reduce its risk exposure by \$56,657 and achieve a net cost reduction of \$14,440 per product release cycle. Thus, the economic benefits of implementing this process framework at TestCo were again positive at this second field test site.

4.3 MediaCo Results

At MediaCo the cost of quality metrics were only available at the product level. Figures 13-16 show the comparison of the before and after scenarios for appraisal costs, prevention costs, errors per KLOC, and failure costs respectively. As those comparisons reveal, both prevention and appraisal costs at MediaCo increased as a result of adopting the process framework. Those increases reflect the costs of performing the additional processes prescribed by the framework. In contrast, technical debt-induced errors and failure costs decreased after the implementation of the process framework, which are the benefits that the firm accrued due to the adoption of our process framework. Statistical tests comparing the before and after framework implementation scenarios show that appraisal costs increased by 12.8% and prevention costs increased by 3% because of the framework implementation. However, the benefits of the framework far outweighed those cost increases because failure costs decreased by a large margin of 43% and errors per KLOC decreased by 50.2%. Overall, the economic benefits attributed to the adoption and use of our process framework included a net reduction in risk exposure of \$1,112,866 for the entire platform and an overall cost reduction of \$60,025 per product release cycle. Thus, the benefits of implementing the normative process framework at MediaCo again outweighed its costs.

4.4 Field Study Results Summary

The empirical results show that all three of the field test site firms were able to reduce technical debt-related errors and improve the quality of their software products. These results were consistent despite the variances in size, age, business context and software process maturity levels of the three commercial organizations. Reductions in both risk exposures and failure costs were the key benefits of the framework adoption that yielded significant cost savings. And, the necessary increase in costs due to the additional process overheads resulting from framework adoption were, on average, only about 8% of overall product

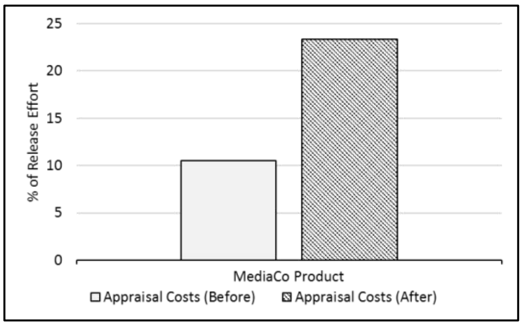


Fig. 13. Appraisal costs at MediaCo before and after process framework adoption and use.

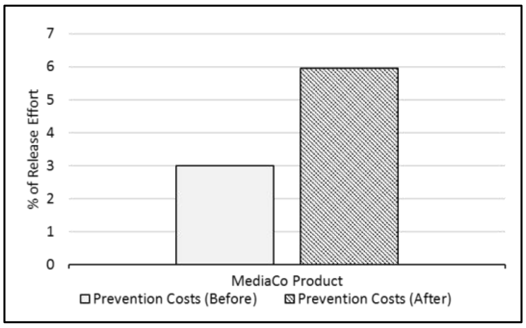


Fig. 14. Prevention costs at MediaCo before and after process framework adoption and use.

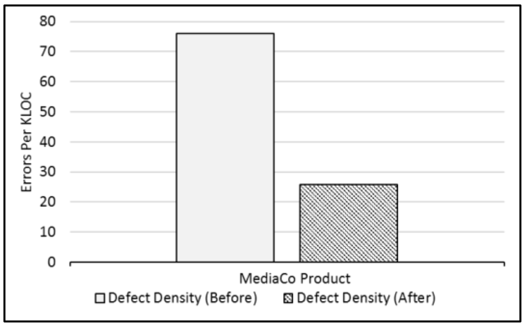


Fig. 15. Technical debt-related errors at MediaCo before and after process framework adoption and use.

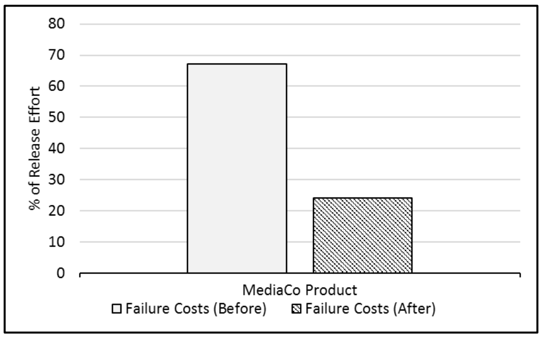


Fig. 16. Failure costs at MediaCo before and after process framework adoption and use.

release costs, and therefore were much smaller than the realized benefits. For example, failure costs alone were reduced by 22% on average. In addition, the firms were able to significantly reduce both their business risk exposures (on average by $512,714) and the total costs associated with their product release cycles (on average by $57,030).

Comparing the benefits realization across the different sites we see that costs of process overheads due to our framework were highest (about a 13% increase) at MediaCo, the firm with the lowest process maturity level in our field study sample. At the same time, MediaCo also reaped the greatest economic benefit (e.g., more than a million dollars' estimated worth of reduction in business risk exposure) from the framework adoption. This shows that, even though firms without established risk management and quality management processes would be expected to face additional overhead in implementing technical debt management policies, the benefits from systematic identification of technical debt and the subsequent reduction in risk exposures are likely to outweigh the costs of the process overhead.

5 Discussion

In this section we discuss the implications of our results for theory development and research on technical debt management. Also, we highlight best practices for achieving an effective alignment between processes for technical debt management with other organizational process assets based on our interactions with practitioners during the field studies. Finally, we conclude by highlighting research opportunities for expanding the normative process framework reported in the study.

5.1 Technical Debt Management through the Theoretical Lens of Cost of Quality

In their call for research on technical debt management, Brown *et al*. [9] noted that the economic impacts and risks of technical debt are inadequately understood, and highlighted the need for a defined set of processes for making the technical debt embedded in systems explicit, and subject to tracking and management. In this study we responded to this call and took a process view of managing technical debt. We proposed a normative framework that integrates the processes necessary for managing technical debt with established quality management processes in software production. The framework underpins a theoretical view that the trade-offs and economic consequences of actions related to technical debt can be better understood and managed using the established principles governing cost of quality issues in software production [22,23,24,38]. The development and test of the study's process framework thus contributes to theory development by bridging the existing gaps between the technical debt and software quality management literatures.

Viewing technical debt through the cost of quality theoretical lens provides a new opportunity to develop models that help shed light on the economic consequences of both the obligations and options resulting from technical debt [9,10]. The cost of quality literature provides a rigorous theoretical foundation for developing policies for technical debt management. For example, existing optimization models depicting the behavior of conformance and nonconformance quality costs [e.g., 42,43,44] can be utilized to understand the relationship between maintenance obligations (costs) and business opportunities (options or benefits) that stem from technical debt. In addition to making the above theoretical connection, the framework proposed in this study provides actionable steps for real-world implementation and empirical data collection. Thus, we believe the normative process framework proposed in this study can serve as a foundation for future studies aiming to develop and verify policies for managing technical debt.

5.2 Aligning Technical Debt Management with Organizational Process Assets: Best Practices

The process framework proposed in this study highlights the need for the inclusion of a wide range of organizational process assets beyond source code for managing technical debt. Such assets include the business plans, product requirements and fulfillment roadmaps, historical defect backlog and resolution data, and risk management policies of a software organization. Such assets provide important inputs to the technical debt management steps as shown in Figure 1, and we advocate a tight integration between the process registers holding information on such assets as illustrated in Figure 2. While the extent of process assets and their integration typically depends on the process maturity of a software organization, the adoption and use of the normative framework proposed by this study is not predicated on the achievement of any specific process maturity level. Based on our field study observations we identified a few practices that assisted in the implementation and continued use of the framework at the three research sites involved in the study. Those practices could be seen as important conditions needed for the implementation and use of process framework.

1. *Engagement between business and engineering stakeholders:* The framework advocates the involvement of stakeholders from both the business and engineering functions for technical debt management, and relies on practices to bring together data that are typically present within those siloes. Examples of such practices are business-goal driven funding mechanisms [45,46], value-driven software engineering and quality management [35,47,48], and better communication between the different stakeholders [49].
2. *Willingness to derive and use policies based on a probabilistic analysis framework:* To accommodate the uncertainties faced by businesses, the process framework embraces a probabilistic approach for estimating the economic impacts of actions related to technical debt. Thus, there would be a need for both the business and engineering teams using the framework to adapt their plans according to the situations that unfold within a planning horizon, rather than expect a deterministic policy.
3. *Limited process overhead:* As seen in our field studies, adoption of the framework imposes a process overhead that is reflected as an increase in the appraisal costs faced by the software teams. The adoption and continued use of the process framework hinges on the ability of firms to be able to limit such a process overhead [50]. Achieving a balance between the extent of centralized process compliance imposed on the software teams and the extent to which they can make appropriate decisions by themselves without the intervention of centralized quality control units would help in this regard [36].

5.3 Further Research

We acknowledge the need to conduct additional field tests using the normative process framework before we can generalize the results reported in this study. Such a generalization is possible with further research because the proposed process framework has been shown here to be amenable to both large and small software teams, and those that operate at different process maturity levels. Additional tests of the framework at different software organizations will also help to expand the tools and techniques considered for the three steps of technical debt management prescribed by the framework. Similarly, examining how the normative framework can be successfully implemented in organizations following different software development models and quality management standards would also help in future refinements of the framework.

As mentioned before, another stream of research opportunities stems from the integration capabilities of the proposed normative process framework. Apart from the potential usage of the cost of quality optimization models discussed in Section 5.1., there are opportunities to exploit the integration between the technical debt, design moves, and risk registers posited in this study (see Figure 2). Through this integration the maintenance obligations (principal and interest of technical debt) and options created by design actions of developers can be categorized into threats and opportunities. Then, appropriate processes can be defined for such actions as avoiding, accepting, mitigating, exploiting, transferring, or enhancing these risks. Future research could test the impacts of such an integrated process approach in influencing the risk-taking or risk-avoiding behaviors of software teams. This would help add nuance to the existing taxonomies of developer actions related to technical debt [9].

Overall, we believe an integrated approach to the management of technical debt as advocated in this study provides a strong theoretical and empirical foundation for continued research efforts to find processes that reduce the various frictions in software production [51].

References

[1] N. Ramasubbu and C.F. Kemerer, “Managing Technical Debt in Enterprise Software Packages,” *IEEE Trans. Software Engineering*, vol. 40, no. 8, pp. 758-772, Aug 2014.

[2] N. Ramasubbu and C.F. Kemerer, “Technical Debt and the Reliability of Enterprise Software Systems: A Competing Risks Analysis,” *Management Science*, vol. 62, no. 5, pp. 1487-1510, May 2016.

[3] A. MacCormack and D. J. Sturtevant, “Technical Debt and System Architecture: The Impact of Coupling on Defect-Related Activity,” *Journal of Systems and Software*, vol. 120, no. 10, pp. 170-182, 2016.

[4] Z. Li, P. Avgeriou, and P. Liang, “A Systematic Mapping Study on Technical Debt and its Management,” *Journal of Systems and Software*, vol. 101, no. 3, pp. 193-220, March 2015.

[5] E. Tom, A. Aurum, R. Vidgen, “An Exploration of Technical Debt,” *Journal of Systems and Software*, vol. 86, no. 6, pp. 1498-1516, June 2013.

[6] N.S. R. Alves, T.S. Mendes, M.G. de Mendonca, R.O. Spinola, F. Shull, and C. Seaman, “Identification and Management of Technical Debt: A Systematic Mapping Study,” *Information and Software Technology*, vol. 70, no. 2, pp. 100-121, Feb. 2016.

[7] R. Marinescu, “Assessing Technical Debt by Identifying Design Flaws in Software Systems,” *IBM Journal of Research and Development*, vol. 56, no. 2, pp. 9:1-9:13, Sep./Oct. 2012.

[8] P. Kruchten, R. L. Nord, I. Ozkaya, D. Falessi, “Technical Debt: Towards a Crisper Definition Report,” *ACM SIGSOFT Software Engineering Notes*, vol. 38, no.5, pp. 51-54, Sep. 2013.

[9] N. Brown, Y. Cai, Y. Guo, R. Kazman, et al., “Managing Technical Debt in Software-Reliant Systems,” *Proc. 2010. FSE/SDP Workshop on Future of Software Engineering Research*, pp. 47-52, 2010.

[10] J. Woodard, N. Ramasubbu, T.F. Tschang, and V. Sambamurthy, “Design Capital and Design Moves: The Logic of Digital Business Strategy,” *MIS Quarterly*, vol. 37, no. 2, pp. 537-564, 2013.

[11] P. Kruchten, R. L. Nord, and I. Ozkaya, “Technical Debt: From Metaphor to Theory and Practice,” *IEEE Software*, vol. 29, no. 6, pp. 18-21, Nov/Dec. 2012.

[12] A. Ampatzoglou, A. Ampatzouglou, A. Chatzigeorgiou, and P. Avgeriou, “The Financial Aspect of Managing Technical Debt: A Systematic Literature Review,” *Information and Software Technology*, vol. 64, no. 8, pp. 52-73, Aug. 2015.

[13] E. Lim, N. Taksande, and C. Seaman, “A Balancing Act: What Software Practitioners Have to Say about Technical Debt,” *IEEE Software*, vol. 29, no. 6, pp. 22-27, Nov./Dec. 2012.

[14] D. Falessi, M. A. Shaw, F. Shull, K. Mullen, M. S. Keymind, “Practical Considerations, Challenges, and Requirements of Tool-support for Managing Technical Debt,” *Proc. 4th International Workshop on Managing Technical Debt*, DOI: *10.1109/MTD.2013.6608673*, 2013.

[15] D. Falessi, P. Kruchten, R. Nord, and I. Ozkaya, “Technical Debt at the Crossroads of Research and Practice,” *ACM SIGSOFT Software Engineering Notes*, vol. 39, no 2, pp. 1-15, 2014.

[16] N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord, and I. Gorton, “Measure it? Manage it? Ignore it? Software Practitioners and Technical Debt,” *Proc. 10th Joint Meeting of European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 50-60, 2015.

[17] R. Chillarege, I.S. Bhandari, J.K. Chaar, M.J. Halliday, D.S. Moebus, B.K. Ray, and M.Y. Wong, “Orthogonal Defect Classification: A Concept for In-Process Measurements,” *IEEE Trans. Software Engineering*, vol. 18, no. 11, pp. 943-956, Nov. 1992.

[18] T. Nakajo and H. Kume, “A Case History Analysis of Software Error Cause-Effect Relationships,” *IEEE Trans. Software Engineering*, vol. 17, no. 8, Aug. 1991.

[19] C.Y. Huang, C.S. Kuo, and S.P. Luan, “Evaluation and Application of Bounded Generalized Pareto Analysis to Fault Distributions in Open Source Software,” *IEEE Trans. Reliability*, vol. 63, no. 1, Mar. 2014.

[20] G. Concas, M. Marchesi, A. Murgia, R. Tonelli, and I. Turnu, “On the Distribution of Bugs in the Eclipse System,” *IEEE Trans. Software Engineering*, vol. 37, no. 6, pp. 872-877, Nov./Dec. 2011.

[21] L.C. Briand, K.E. Emam, B.G. Freimut, and O. Laitenberger, “A Comprehensive Evaluation of Capture-Recapture Models for Estimating Software Defect Content,” *IEEE Trans. Software Engineering*, vol. 26, no. 6, pp. 518-540, June 2000.

[22] S.A. Slaughter, D.E. Harter, M.S. Krishnan, “Evaluating the Cost of Software Quality,” *Communications of the ACM*, vol. 41, no. 8, pp. 67-73, Aug. 1998.

[23] A. Poth and A. Sunyaev, “Effective Quality Management: Value- and Risk-Based Software Quality Management,” *IEEE Software*, vol. 31, no. 2, pp. 79-85, Nov./Dec. 2014.

[24] R. Hackbarth, A. Mockus, J. Palframan, and R. Sethi, “Improving Software Quality as Customers Perceive It,” *IEEE Software*, vol. 33, no. 4, pp. 40-45, Jul./Aug. 2016.

[25] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, “A Large-Scale Empirical Study of Just-in-time Quality Assurance,” *IEEE Trans. Software Engineering*, vol. 39, no. 6, pp. 757-773, Nov. 2012.

[26] D.A. Harter and S.A. Slaughter, “Quality Improvement and Infrastructure Activity Costs in Software Development: A Longitudinal Analysis,” *Management Science*, vol. 49, no. 6, pp. 784-800, June 2003.

[27] C.J. Corbett, M.J. Montes-Sancho, and D.A. Kirsch, “The Financial Impact of ISO 9000 Certification in the United States: An Empirical Analysis,” *Management Science*, vol. 51, no. 7, pp. 1046-1059, Jul. 2005.

[28] C.J. Corbett, “Global Diffusion of ISO 9000 Certification Through Supply Chains,” *Manufacturing and Service Operations Management*, vol. 8, no. 4, Oct. 2006.

[29] N. Zazworka, A. Vetro, C. Izurieta, S. Wong, Y. Cai, C. Seaman, and F. Shull, “Comparing Four Approaches for Technical Debt Identification,” *Software Quality Journal*, vol. 22, no. 3, pp. 403-426, Sep. 2014.

[30] B. Curtis, J. Sappidi, and A. Szynkarski, “Estimating the Principal of an Application’s Technical Debt,” *IEEE Software*, vol. 29, no. 2, pp. 34-42, Nov./Dec. 2012.

[31] A. van Lamsweerde, R. Darimont, and E. Letier, “Managing Conflicts in Goal-driven Requirements Engineering,” *IEEE Trans. Software Engineering*, vol. 24, no. 2, pp. 908-926, Nov. 1998.

[32] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe, “Toward Data-Driven Requirements Engineering,” *IEEE Software*, vol. 33, no. 1, pp. 48-54, Jan./Feb. 2016.

[33] J. Ropponen, and K. Lyytinen, “Components of Software Development Risk: How to Address Them? A Project Manager Survey,” *IEEE Trans. Software Engineering*, vol. 26, no. 2, pp. 98-112, Feb. 2000.

[34] B.W. Boehm, “Software Risk Management: Principles and Practices,” *IEEE Software*, vol. 8, no.1, pp. 32-41, Jan. 1991.

[35] A. Poth and A. Sunyaev, “Effective Quality Management: Value-and Risk-Based Software Quality Management,” *IEEE Software*, vol. 31, no. 6, pp. 79-85, Nov./Dec. 2014.

[36] N. Ramasubbu, “Governing Software Process Improvements in Globally Distributed Product Development,” *IEEE Trans. Software Engineering*, vol. 40, no. 3, pp. 235-250, Mar. 2014.

[37] H.D. Frederiksen and L. Mathiassen, “A Contextual Approach to Improving Software Metrics Practices,” *IEEE Trans. Engineering Management*, vol. 55, no. 4, pp. 602-616, Oct. 2008.

[38] R. Kishore, M. E. Swinarski, E. Jackson, and H. R. Rao, “A Quality-Distinction Model of IT Capabilities: Conceptualization and Two-Stage Empirical Validation using CMMi Processes,” *IEEE Trans. Engineering Management*, vol. 59, no. 3, pp. 457-469, Aug. 2012.

[39] D. Harter, C. F. Kemerer and S.A. Slaughter, “Does Software Process Improvement Reduce the Severity of Defects? A Longitudinal Field Study”, *IEEE Trans. Software Engineering*, v. 38, n. 4, pp. 810-827, July/Aug. 2012.

[40] D. P. Harvie and A. Agah, “Targeted Scrum: Applying Mission Command to Agile Software Development,” *IEEE Trans. Software Engineering*, vol. 42, no. 5, pp. 476-489, May 2016.

[41] C.F. Kemerer, “An Agenda for Research in the Managerial Evaluation of Computer-Aided Software Engineering Tool Impacts”, Proceedings of the 22nd Hawaii International Conference on System Sciences, v. II, pp. 219-228, Jan. 1989.

[42] C.D. Ittner, “Exploratory Evidence on the Behavior of Quality Costs,” *Operations Research*, vol. 44, no. 1, pp. 114-130, Feb. 1996.

[43] P. Nandakumar, S.M. Datar, and R. Akella, “Models for Measuring and Accounting for Cost of Conformance Quality,” *Management Science*, vol. 39, no. 1, pp. 1-16, Jan. 1993.

[44] G. Li and S. Rajagopalan, “Process Improvement, Quality, and Learning Effects,” *Management Science*, vol. 44, no. 11, pp. 1517-1532, Nov. 1998.

[45] M. Denne, and J. Cleland-Huang, “The Incremental Funding Method: Data-Driven Software,” *IEEE Software*, vol. 21, no.3, pp. 39-47, May/June 2004.

[46] P. Clements and L. Bass, “The Business Goals Viewpoint,” *IEEE Software*, vol. 27, no. 6, pp. 38-45, Nov./Dec. 2010.

[47] B. Boehm and L.G. Huang, “Value-Based Software Engineering: A Case Study, *IEEE Software*, vol. 36, no. 3, pp. 33-41, Mar. 2003.

[48] A. Poth and A. Sunyaev, “Effective Quality Management: Value- and Risk-Based Software Quality Management,” *IEEE Software*, vol. 31, no. 6, pp. 79-85, Nov./Dec. 2014.

[49] J. Schulenklopper and E. Rommes, “Why They Just Don’t Get It: Communicating about Architecture with Business Stakeholders,” *IEEE Software*, vol. 33, no. 3, pp. 13-19, May/June 2016.

[50] C.F. Kemerer, "Progress, Obstacles, and Opportunities in Software Engineering Economics", *Communications of the ACM*, v. 41, n. 8, pp. 63-66, Aug. 1998.

[51] P. Avgeriou, P. Kruchten, R.L. Nord, I. Ozkaya, and C. Seaman, "Reducing Friction in Software Development," *IEEE Software*, vol. 33, no. 1, pp. 66-73, Jan./Feb. 2016.

Narayan Ramasubbu received the bachelor of engineering degree from Bharathiar University, India, and the PhD degree from the University of Michigan, Ann Arbor, MI. He is an associate professor at the Katz Graduate School of Business at the University of Pittsburgh. Prior to his academic career, he was a senior developer at SAP AG and CGI Inc. His current research interests include software-driven innovation, software engineering economics with a focus on globally distributed product development and service delivery, and the design, implementation, and governance of enterprise information systems. He is an associate editor at *Management Science*, and a member of the IEEE Computer Society.

Chris F. Kemerer received the BS degree from the Wharton School at the University of Pennsylvania, and the PhD degree from Carnegie Mellon University. He is the David M. Roderick Professor of Information Systems at the Katz Graduate School of Business at the University of Pittsburgh, and an adjunct professor of software engineering in the School of Computer Science at Carnegie Mellon University. Previously, he was an associate professor at MIT. His current research interests include management issues in information systems and software engineering, and the adoption and diffusion of information technologies. He has served in a number of editorial positions, including as the editor-in-chief of *Information Systems Research*, and is a past associate editor of *IEEE Transactions on Software Engineering*. He is an INFORMS Information Systems Society Distinguished Fellow and an ISI/Thomson Reuters Highly Cited Researcher in Computer Science. He is a member of the IEEE Computer Society.