Managing Technical Debt in Enterprise Software Packages

Narayan Ramasubbu, *Member, IEEE*, and Chris F. Kemerer, *Member, IEEE*

Abstract—We develop an evolutionary model and theory of software technical debt accumulation to facilitate a rigorous and balanced analysis of its benefits and costs in the context of a large commercial enterprise software package. Our theory focuses on the optimization problem involved in managing technical debt, and illustrates the different tradeoff patterns between software quality and customer satisfaction under early and late adopter scenarios at different lifecycle stages of the software package. We empirically verify our theory utilizing a ten year longitudinal data set drawn from 69 customer installations of the software package. We then utilize the empirical results to develop actionable policies for managing technical debt in enterprise software product adoption.

Index Terms—Technical debt, enterprise software, software platforms, customer satisfaction, software quality, technology adoption, COTS, software evolution, software maintenance, software management, longitudinal data

1 Introduction

The term "technical debt" is used as a metaphor to refer to the likely long-term costs associated with software development and maintenance shortcuts taken by programmers to deliver short-term business benefits [1], [2], [3]. For example, it is well documented that software developers often face a resource conflict—a situation where developers have to satisfy multiple demanding goals, such as improving the functionality of software packages under tight budgets and aggressive delivery deadlines [4]. In these situations, and others, developers tend to tradeoff potential longer-term benefits of thorough conformance to development and maintenance standards for the immediate short-term business benefits of rapidly adding functionality by taking shortcuts (i.e., incurring technical debt). Therefore, there is growing interest in the software engineering and technology management research communities to empirically make use of the technical debt metaphor to derive appropriate decision models for software product management [2], [5], [6].

We reviewed the emerging literature on technical debt and provide a summary of this literature in Table 1. We note four interrelated research streams: (a) taxonomy, frameworks, and perspectives, (b) measuring and visualizing technical debt, (c) valuation of the impacts of technical debt, and (d) managerial decision frameworks. Overall, the models of technical debt management discussed in these four streams of literature have tended to emphasize the *costs* of debt at the expense of the offsetting *benefits* derived from the debt. While the focus on cost, often measured in terms of deterioration of software quality, has aided managers in monitoring and avoiding excessive buildup of technical debt, a balanced perspective accounting for *both* the business benefits and the costs associated with incurring technical debt has received less emphasis.[1] Furthermore, the prevailing cross-sectional models do not account for the evolutionary nature of technical debt accumulation throughout the lifecycle of software packages. Such cross-sectional analyses of technical debt are likely to be a limited view, failing to account for the interplay between the benefits and costs of technical debt over the lifespan of software, which is typically measured in multiple years for successful implementations [7], [8], [9], [10].

In this paper we add to the understanding of technical debt in practice by developing and testing a theory on the costs and benefits of technical debt that specifically accounts for the different pathways in which a software package evolves, contingent on the manner in which customers of the package adopt it.[2] To incorporate this evolutionary perspective we track how 69 customers (Fortune-500 firms) of a commercial enterprise software package adapted and built upon the functionality released by the package's vendor over its ten year lifecycle. The vendor's release of software features followed an "S"-shaped curve—a pattern documented in the management research literature as a generalized pattern of diffusion of products in a larger population [11], [12]. Using this "S"-shaped feature release pattern of the vendor as an anchor, and utilizing detailed package configuration information from the

• *The authors are with the Joseph M. Katz Graduate School of Business, University of Pittsburgh, Pittsburgh, PA 15260. E-mail: narayanr@pitt.edu, ckemerer@katz.pitt.edu.*

Manuscript received 29 July 2013; revised 9 May 2014; accepted 20 May 2014. Date of publication 1 June 2014; date of current version 13 Aug. 2014. Recommended for acceptance by D.I.K. Sjøberg.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2014.2327027

1. It should be noted that some more recent research is more balanced, e.g., Lim *et al*. document the emerging practitioner's perspective on balancing technical debt [6] and Kruchten *et al*. propose an organizing landscape for technical debt, and call for further research that "expresses all software development activities in terms of sequences of changes associated with a cost and a value over time" [3].

2. A position paper with a preliminary version of the theoretical model was presented at the *4th International Workshop on Managing Technical Debt* [10]. In the current paper we have modified and expanded the model and report a comprehensive empirical evaluation of the model using a rich, longitudinal data set drawn from a set of commercial enterprise software package installations.

TABLE 1
Technical Debt Literature

No	Theme	Literature Reference
1	Taxonomy, frameworks, and perspectives	[1], [2], [3], [6], [13], [14], [15], [16], [17], [18], [19]
2	Measuring and visualizing technical debt	[5], [20], [21], [22], [23], [24], [25]
3	Valuing the impact of technical debt	[26], [27], [28], [29], [30], [31], [32], [33], [34]
4	Managerial decision frameworks	[17], [20], [25], [35], [36], [37], [38], [39], [40]

69 customer implementations, we traced the different evolutionary pathways of features growth, comparing them with the evolution of the base version supplied by the vendor. Based on these patterns of features growth we grouped customers of the software package into *base*, *early*, and *late adopter* groups.

Utilizing software quality and customer satisfaction valuation metrics, we compare and contrast the performance of these three adopter groups, inferring the costs and benefits of accumulating technical debt at different stages of the package's lifecycle. Finally, we derive a set of managerial policies specific to both higher and lower technical debt scenarios that can help managers to make informed decisions on software investments, such as refactoring or architectural improvements, at different points in the lifespan of the software package.

The rest of the paper is organized as follows. In Section 2 we document the evolution of the software package we study and establish the empirical context of our theory development. We proceed to present our formal hypotheses in Section 3, describe our data collection in Section 4, the analysis methodology in Section 5, and present the results of our hypotheses testing in Section 6. We discuss the broader implications of the results for both practice and future research in Section 7.

2 Research Approach

We sought to study the evolution of a software product over a multi-year period in order to gain a better understanding of the costs and benefits of technical debt over its entire lifespan. Kemerer and Slaughter [7] review the literature in the software evolution area and provide guidance on studying and understanding "how a software system travels as it evolves over time." To identify and compare the evolutionary paths of software systems, they prescribe a sequence analysis method, which involves mapping events into discernable and time-ordered phases. Once the distinct phases in the evolution of a software system are identified, one can employ statistical analysis, such as time-series analysis, for modeling and analyzing outcome metrics of interest over the different evolutionary phases [7]. We follow this two-stage approach for theory development and empirical analysis. First, we establish the pattern of evolution of a software package and anchor our theory development on this empirical pattern. Then, we utilize a series of statistical models to test our theory.

2.1 Evolution of the Software Package and Its Variants

We collected a variety of longitudinal data from a leading commercial European software product development firm and its major customers. The data pertained to a recently retired enterprise software package that was originally launched in 2002. The package was sold as modifiable commercial off-the-shelf software to customers (COTS). This *base version* could be altered in a variety of ways (i.e., customized) to suit the specific needs of a customer. The vendor provided a standard way to modify the package through supported Application Program Interfaces (APIs). However, these APIs did not support addition of new business functionality (termed "transactions" by the vendor), but rather provided a variety of data import and export provisions, and allowed for altering business rules through existing variables in the database. To add additional business functionality to the package customers had to modify source code through an integrated development environment (IDE) toolkit provided by the vendor.[3]

Through source code modification an aggressively growth-oriented customer could add business functionality at a faster pace than the vendor firm could fulfill using its standard releases. We term this group of customers as "early adopters". As is typical in the industry, the vendor firm made no guarantees to preserve added or custom-modified code by early adopters during its regular updates to the base package.[4]

In contrast to the early adopters, a different group of customers, termed "late adopters", may not utilize all of the features offered by the vendor's package, and had the opportunity to install only a subset of the available features.[5] Late adopters tend to shun potentially volatile early ("beta") functionality and custom modifications, and thus avoid any potential conflicts with future updates from the vendor. Thus, depending on the extent of customization and utilization, the evolution of package variants at customer sites could be significantly different from the base package offered by the vendor over its entire lifecycle. Such variations in growth trajectories of installed software packages at these customer sites provided an excellent research context for our theory development on the costs and benefits of

3. Both the vendor-supplied software package and IDE supported a proprietary object-oriented programming language that was similar to C++. All source code modifications were automatically tracked and labelled with a timestamp and modifier identification code, which greatly benefited our analysis.

4. Once source code is modified, automatic updating of the package by the vendor was permanently disabled. In the absence of automatic updates, customers had to manually install patches supplied by the vendor and manage the associated integration testing verification procedures themselves.

5. The vendor's product release notes provided details on the levels of test coverage for different modules as well as transactions that were considered "beta" functionality or "release candidates" that were bundled into the base package. Late adopters typically omitted such modules with low test coverage and "beta" transactions.

TABLE 2
Software Package Variants

Base adopter	Early adopter	Late adopter
Customer adopts package as provided by vendor	Customer modifies the software package by altering source code (adding new modules and/or altering the functionality of existing modules)	Customer selects a sub-set of modules from the vendor's package and implements them without any modifications

technical debt. The longitudinal nature of our data set allows for the testing of various hypotheses of expected behavior over time, and the single vendor source holds many variables constant, allowing for greater focus on the elements that do change that are the goal of this research.

The categorization of the package variants as base, early, and late adopters is summarized in Table 2, and the patterns of cumulative functionality growth among these package variants at 69 customers (Fortune 500 firms) are presented in Fig. 1.

Fig. 1 depicts how the vendor and its customers added business functionality over the 120 month life of the package. The vendor firm and its customers measured business functionality in units they termed "transactions." A *transaction* in this context refers to the execution of program modules that accomplish specific business functionality. For example, "create a sales order", "reconcile lead accounts", and "launch a promotions campaign" are some commonly-used transactions in the software package. Similar in spirit to Function Points, the transactions metric captures the customer-focused functionality offered by the software package [41]. Also, it maps well to metrics in established software process models, such as development velocity, which is the rate at which features or customer requirements are implemented in a product [42], [43]. In addition, it avoids the potential for error in measuring complex and diverse internal interfaces between the various program modules (often coded in multiple programing languages) and databases, and instead provides a consistent, vendor-created metric. We utilized the cumulative number of transactions available in the package to depict its growth.

As can been seen from Fig. 1 the growth of the vendor supplied package (base version, the dark black line) follows an S-shaped pattern. The vendor appears to follow a feature release strategy that is in line with an S-shaped product growth model in the broader population, which has been established as an empirical generalization—a regularity that repeats over a variety of circumstances—in the management research literature on the diffusion of innovations [44], [45]. The S-curve product growth follows from the different demand patterns for a new product over time. Early adopters of a new product typically generate a small initial demand for the product. If the product proves successful with the early adopters, the value of the product begins to diffuse through the population, resulting in a sudden "takeoff" in the demand for the product. Over time, product growth eventually tapers off due to a number of factors, including population size limitations and the emergence of other, new competing technologies.

Fig. 2 isolates and illustrates the three distinct phases of evolution of the base version of the software package. Point *t* in the curve (takeoff point) shows the point at which product's functionality growth "takes off" after a slow initial growth; point *s* in the curve (saturation point) shows the point at which the functionality growth of the product begins to decline. In the S-shaped or logistic growth model, the inflection point *i* on the curve, lies at the half way mark of the total functionality present in a product at the end of its lifespan. The takeoff, inflection, and saturation points on the growth curve provide us with the events to demarcate the evolution of the software package into sequential phases for further analysis.

There were 11 customers in our data set who were *base adopters*. That is, the package evolution at these customer sites followed the exact trajectory as the vendor's base package. This functionality growth for the base adopter group of customers is shown in Fig. 3. The functionality growth pattern of the base version of the software package provided by the vendor serves as a benchmark to compare the growth trajectories of variants of the package that result from different patterns of functionality adoption (e.g., through custom source code modifications or omissions of functionality).

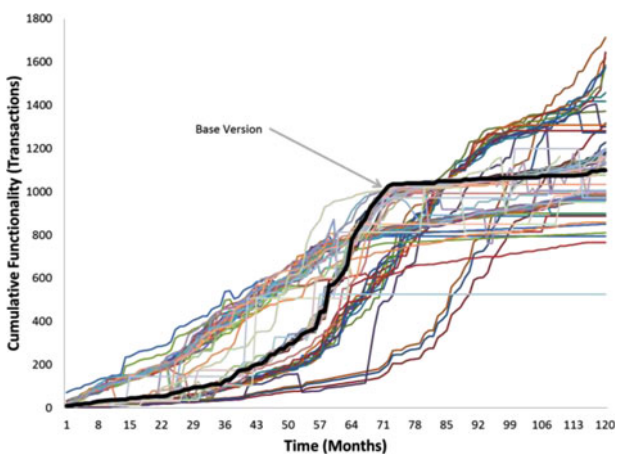

Fig. 1. Functionality growth of software package variants.

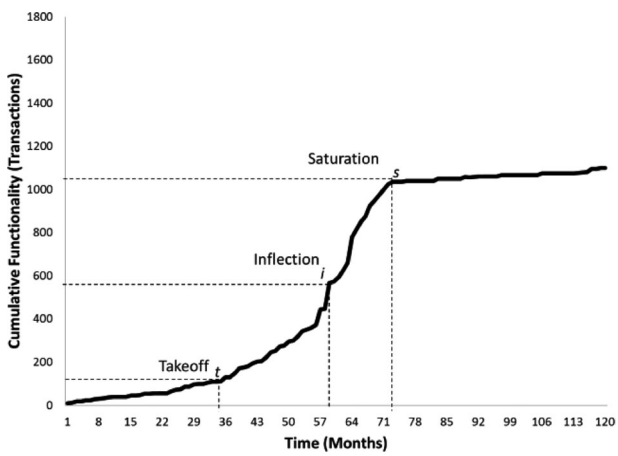

Fig. 2. Base package growth.

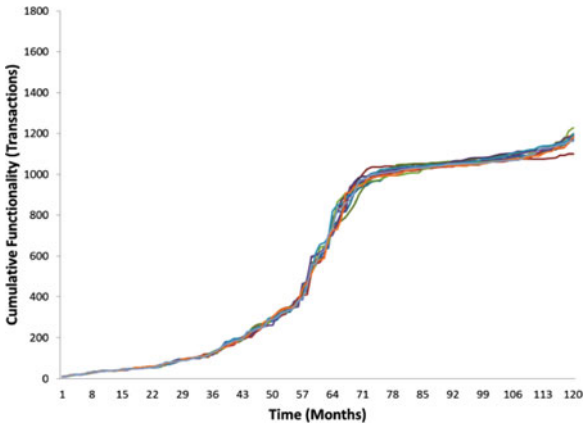

Fig. 3. Base adopter group ($N = 11$).

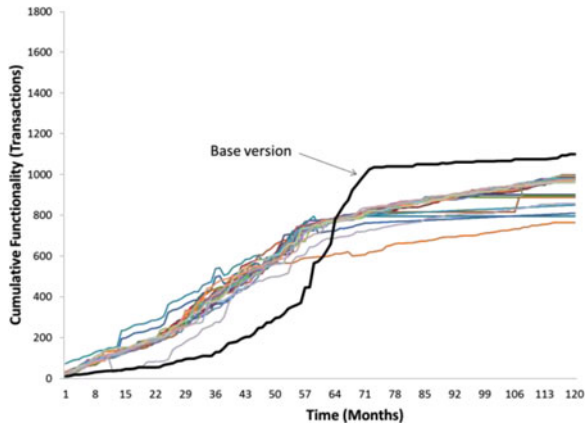

Fig. 4. Early adopter group ($N = 22$).

The base version attempts to satisfy the common requirements of the entire customer population of the package, whereas the individual package variants reflect customer-specific needs and processes. Enterprise software vendors typically provide a predictable roadmap[6] for the base version of their packages, and take responsibility for preserving compatibility requirements and adherence to standards throughout the base version package's lifespan [46], [47]. In our case, the vendor provided an annual forecast of expected feature releases to its customers, and in every monthly release note clearly marked features that were considered "beta" or marked for further refinement and features that were marked for no further support going forward.[7]

As summarized in Table 2, we consider two important categories of package variant groups belonging to the *early adopters* and *late adopters*. An early adoption scenario occurs when a customer adds functionality more rapidly than the general customer population of the base package, thereby "taking off" earlier than the base package. However, as can be seen in Fig. 4, these early adopter package variants also tend to have an earlier saturation point than the base version of the package. Twenty-two customers in our data set fit this early adopter pattern.

One possible explanation suggested by the literature for this observed pattern of early saturation of functionality growth among early adopters could be that early adopter package variants suffer from the burden of technical debt [2], [5], [17]. Since the vendor's maintenance policy specifically did not support custom-modified software code, there is a high probability that the early adopters risked incurring technical debt since they trade-off the long-term benefits of a likely better technical design and adherence to vendor-specified standards for perceived short-term business benefits realized through the more rapid addition of new functionality. While the rapid addition of business functionality in early adopter variants can be expected to yield business benefits, it must be noted that these benefits could come at the expense of thorough investments in adhering to the software standards supported by the vendor. Accumulating non-standard functionality in the software package is also likely to accentuate code decay, causing performance bottlenecks and creating software code that is more difficult to maintain [48]. Thus, the early adoption strategy in this software package's context could, after the initial early takeoff, lead to accumulation of technical debt that causes the rate of functionality growth to decrease rapidly over time.

On the other hand, a late adopter scenario occurs when customers tend to selectively implement features available in the base package; only functionality that has matured and is likely to be free of initial build errors and incompatibilities is accepted by late adopters. Thus, the takeoff of functionality growth among late adopters is typically delayed as compared to the base version's S-curve growth. Potential benefits of late adoption, such as avoiding the risks of premature functionality, come at a cost of lowered levels of business functionality, especially in the early stages of the package lifecycle, which can be seen as a form of opportunity cost for the organization following this strategy. The lack of business functionality in late adopter package variants could lead to lowered levels of short-term return on investment for customers. However, as the software package evolves, the data for the 15 late adopters depicted in Fig. 5 show that late adopter variants saturate more slowly.

One possible explanation suggested by the literature for the late saturation points observed in the late adopter package variants could be that these packages have lower technical debt [2], [5], [17]. Lower technical debt in these late adopter package variants, in turn, might provide an opportunity for customers to more rapidly add functionality during the later stages of product evolution, and thereby derive higher business value by prolonging the saturation of the functionality growth. Therefore, late adopter package variants experience delayed takeoff and saturation

6. Product roadmapping is a process that aids a vendor to plan the timing of its product development phases, the resources needed to support the development effort, and the various technologies that might be used within the vendor's current and future product offerings [46], [47]. Thus, a product roadmap offers a longitudinal framework to integrate market, product, and technology concerns of a firm.

7. We mapped the 120 monthly release notes to transactions in the base package and identified the initial release and subsequent modification events for all transactions. Following this, we tracked all source code modifications and omissions of transactions in the 69 customer installations corresponding to the 120 monthly releases of the vendor's base package. Treating the 69 source code repositories as branches to the base package, we utilized an automatic SVN log comparison tool to compare the branches. All the results returned by the tool were further manually verified so that we have reliable comparison of the growth models of the base, early adopter, and late adopter packages.

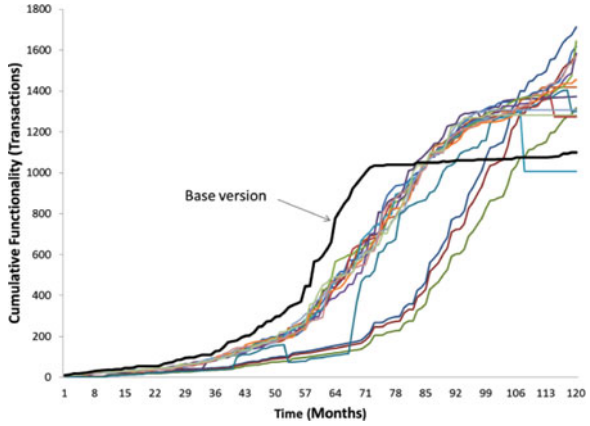

Fig. 5. Late adopter group ($N = 15$).

as compared to the base and early adopter groups. In the following sections we develop our hypotheses to formally test these assertions.

2.2 Assessing the Business Value of Package Variants

The evolution of different variants of a software package under the early and late adopter modes vary significantly. To map the functionality gain or loss in a package variant to business value we consider two performance metrics: software quality and customer satisfaction. We chose these performance metrics because they map to both the long term costs (poor software quality) and short term benefits (higher customer satisfaction) of technical debt as suggested by the literature, and the measurement and ecological validity of these constructs have been well established in prior literature, including in the context of enterprise software [49], [50], [51], [52]. Furthermore, these metrics were contemporaneously collected by the vendor throughout the package's lifespan, and hence are reliable performance metrics with consistent measurement.[8]

In order to systematically examine the business value of both early and late adopter strategies we make use of the takeoff and saturation point milestones that demarcate a package's evolutionary stages, and develop a set of hypotheses to assess the state of software quality and customer satisfaction of the package variants at these phases.

3 Theory and Hypotheses

Our theory proposes to explain the patterns of evolution of the early and late adopter software package variants through the lens of the costs and benefits of technical debt. We posit that early adopters of enterprise software may incur technical debt early in the life of the product and have to bear the costs of this technical debt later. In contrast, late adopters are likely to pay an opportunity cost for not incurring technical debt early, but may gain value through increased functionality growth at the later stages of the product's lifecycle.

Early adopters tend to incur technical debt because of the maintenance policies and product development strategy of the software package vendor. As mentioned in Section 2.1, the package vendor does not guarantee preservation of custom-modified code and does not provide standard ways to extend the package beyond the published application programming interfaces for the current business functionality. Any new transactions added by an early adopter through custom modification of software code could potentially be in conflict with future developments introduced by the vendor. As such, the maintenance burden of custom-modified code lies with the customer. All else being equal, this maintenance burden translates to additional costs, which keep accumulating for an early adopter. With product growth over time the effort required to maintain the customizations and to integrate newly released functionality with the custom-modified software code could become significantly higher enough that they could hinder future functionality additions [17]. Conversely, late adopters do not incur such a maintenance burden and are more likely to continue to add functionality even in the later stages of the product's lifecycle. This contrasting growth scenario for early and late adopters is captured in our first set of hypotheses:

Hypothesis 1a. *Early adopters will be less likely to add features late in the package's lifespan, and the final cumulative amount of functionality in package variants of early adopters will be lower than that of base and late adopters.*

Hypothesis 1b. *Late adopters will be more likely to add more features late in the package's lifespan, and the final cumulative amount of functionality in package variants of late adopters will be higher than that of base and early adopters.*

Our second set of hypotheses focuses on the state of software quality in the early and late adopter variants of a software package. Since our theory posits that early adopters are more likely to incur technical debt than late adopters, and prior studies on technical debt have shown a negative correlation between higher levels of technical debt and software quality [27], [29], [40], our hypotheses are:

Hypothesis 2a. *All else being equal, throughout the evolution of the software package software quality of early adopter package variants will be lower than that of the base package.*

Hypothesis 2b. *All else being equal, throughout the evolution of the software package the software quality of late adopter package variants will be higher than that of the base package.*

To assess the customer satisfaction of early and late adopter package variants we need to account for the lifecycle phase of the software package. In the early stages of a package's evolution, savvy early adopters tend to look for novel business functionality, and therefore a rapid growth in functionality is likely to be relatively more satisfying [45], [50]. Hence, we expect that, all else being equal, the customer satisfaction of early adopter package variants will be greater than that of both the base package and the late adopter variants in the early stage. Since administrators of late adopter package variants are highly selective in choosing feature bundles for installation, end customers may not get the opportunity to access relatively novel, but potentially immature, features of the base package [17]. Thus, in the stage before takeoff we expect that the customer satisfaction of late adopter package variants to be lower than that of

8. These metrics and their data collection are further elaborated in Section 4.

both the base package and the early adopter variants, all else being equal.

In contrast to the phase before takeoff, after the saturation point of base package functionality growth in late adopter package variants tends to exceed that of the early adopter variants. Thus, during the last stages of a product's lifespan we expect that the customer satisfaction of early adopter variants to be lower than that of the base package, and the customer satisfaction of late adopter variants would be higher than that of the base package. This leads to our final set of hypotheses:

Hypothesis 3a. *All else being equal, customer satisfaction of the early adopter package variants will be higher than that of base adopters before the point of takeoff and lower than base adopters after the point of saturation.*

Hypothesis 3b. *All else being equal, customer satisfaction of the late adopter package variants will be lower than that of base adopters before the point of takeoff and higher than base adopters after the point of saturation.*

4 Data Collection and Variables

We collected data from a leading European software product development firm and its major customers. The firm specializes in enterprise business applications such as enterprise resource planning (ERP) systems, customer relationship management (CRM) systems, and supply chain management (SCM) systems. When we completed our data collection in 2012 the firm employed more than 50,000 employees around the world and had revenues exceeding 10 billion Euros.

We collected monthly functionality release data from the product management division of the firm for the entire 10 year lifecycle of a recently retired enterprise software package. We had access to data from 75 customers[9] (Fortune 500 companies) of the vendor firm who had purchased the package during its initial launch in 2002, and collected detailed data on the configurations of the package installations and their functionality growth over the 10 year period from 2002 until 2012 when the package was officially retired.

In addition, the product management division of our research site had contemporaneously communicated with the customer firms through an electronic medium they called their "Customer Support Network" (CSN), which was the channel through which all software updates and notifications were sent. Individual project milestones, bug reports, and maintenance requests by customer firms who installed the package were logged by this CSN system. Software developers at the research site could also remotely assist customers in software installation and other maintenance activities through the same CSN system.

We were able to obtain longitudinal and contemporaneous software quality and customer satisfaction data through the configuration management systems at the client sites and the vendor-supported CSN system. Similar to other studies [28], we parsed the source code of the package variants present in the client configuration management systems to check for the presence of different shortcuts and design standard violations. Software quality was measured using three variables: 1) the backlog (number) of unresolved errors, 2) the average error processing time (in days), and 3) the percentage of errors exceeding the standard response time mentioned in the service level agreement (SLA) contract between the firm and the customer.

Also, every month customers were requested to complete an electronic customer satisfaction survey. To increase the response rate the CSN system issued pop-up reminders every time a customer logged in to check the status of a bug report, or to download a patch. We utilized the archival mechanisms in the CSN system to derive customer satisfaction and software quality data of the package variants at the 75 customer installations. Customer satisfaction was contemporaneously measured in the survey using a 10 point scale (1- very low, 10-very high). Since the functionality growth data was reported in monthly intervals, we utilized the average customer satisfaction in a month as our performance metric.

The above constructs and variables in our data set are summarized in Table 3. Taken together, these metrics map to the performance measures outlined in Kruchten *et al.*'s recently proposed technical debt landscape, spanning the entire internal-external spectrum [3].[10]

5 Analysis

5.1 Categorizing Base, Early, and Late Adopter Variants

As a first step we analyzed the functionality growth of the base package to precisely determine the takeoff and saturation milestones we described in Section 2. A conceptual definition of takeoff in the management literature is the point of transition where large increases of sales growth occurs, which has been operationalized using a heuristic based on data-derived threshold measures [12]. The threshold value is derived using a plot of percentage increase in functionality relative to the base functionality value, and we identify a takeoff as the first instance when the relative growth of functionality crosses the threshold value. In our data set the threshold growth rate of the base version was 15.92 transactions per quarter until the third year of the package's launch (month 35), when the functionality growth started crossing the threshold value to reach a growth rate range of 18-28 transactions per quarter. Thus, this process identified the takeoff point as month 35.

By a similar approach the rate of decrease in functionality growth with respect to base functionality is measured to derive a threshold value of saturation. The saturation in

9. There were a total of 117 firms (customers) who had purchased the product in 2002. We approached all 117 firms to allow the use of their data for our research, but only 75 granted permission, yielding a 64 percent response rate. We signed a collective non-disclosure agreement with the vendor firm and the 75 customers. With the permission of firms that declined to participate in our study, we did check sample data for potential systematic non-responsiveness bias only. No major differences across the participating and non-participating firms were observed.

10. A discussion of mapping the data collected as part of this study and the Kruchten *et al.* [3] technical debt landscape is presented separately in the online Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2014.2327027.

TABLE 3
Key Constructs in the Longitudinal Data

Construct	Variable	Description
Software Size [7], [41], [52], [53], [54], [55]	• Transactions • KLOC • Function Points	A *transaction* is a customer-focused functionality offered by the software package. The number of transactions offered by the software package at any point in time in its lifespan was highly correlated (0.9 Pearson correlation metric) with both KLOC and Function Points.
Software Quality [3], [4], [8], [24], [29], [33]	• Backlog of unresolved errors • Average error processing time • Percentage of errors exceeding the SLA response time for errors	We measured *backlog* as the number of unresolved errors in the monthly feature release cycle followed by the vendor firm. Processing time for resolved errors was retrieved from the error logs maintained by the vendor. We calculated and tracked average error processing times for the 120 monthly release periods. We retrieved the standard response time for error resolution from the Service Level Agreements (SLA) signed by the vendor and the customer firms.
Customer Satisfaction [49], [50], [51], [56]	• Vendor's customer survey measure on 10-point scale (1-very low, 10-very high)	The scores were contemporaneously reported directly by the customer firms to the vendor. Individual customers had multiple customer satisfaction reports in the monthly releases cycles over the 10-year lifespan of the software package (minimum 1, maximum 11, and average 6.61). We utilized the average customer satisfaction for each of the monthly release cycles.

features growth occurred in the seventh year of the package's lifecycle (month 73). Over the 120 month life of the package the firm added a total business functionality of 1,100 transactions. The inflection point of a logistic growth curve (S-curve) is the point in time when half of the total functionality growth has occurred (550 transactions in our case). Thus, we derived the inflection point as month 59, when the cumulative functionality crossed the 550 transactions mark for the first time.

In the second step we analyzed the functionality growth of packages at the 75 customer installations. We needed to exclude six customer installations due to missing data, and therefore examined the growth patterns of the remaining 69 data samples in our analysis. In order to assess the business value of different adopter groups at these commercial customer installations, we iteratively grouped the 69 data samples into four categories using the following procedure:

1) For each lifespan phase (before takeoff, the growth phase between takeoff to saturation, and the saturation phase before retirement), compute if the cumulative functionality present in a customer installation is significantly different (i.e., at $p < 0.05$ levels) from that of the base version.
2) If the functionality difference is uniformly not different from zero, label as "base package group", else
3) If the functionality difference is uniformly higher than zero up to the inflection point, label as "early adopter candidate", else
4) If the functionality difference is uniformly lower than zero up to the inflection point, label as "late adopter candidate", else
5) Label as "transitional candidate".

We then examined each of the candidate groups to further refine the categorization of a customer installation. We verified that the crossover point of the most aggressive early adopter variant occurred after the inflection point of the base package, and that the crossover point of the least conservative late adopter variant occurred after the saturation point of the base version. This ensured that the boundary conditions of our categorization were reliably met, and therefore any conclusions drawn about the general characteristics of early or late adopter strategies from our analysis would have conceptual validity.

This sorting resulted in 48 of the installations being clearly categorized as one of: base, early adopter, or late adopter. The additional 21 customer installations were identified as "transitional candidates", i.e., not one of the first three canonical cases, and were not used to test the first set of hypotheses, but were set aside to be examined later (in Section 6.4).

5.2 Hypotheses Testing

Table 4 presents the mean-level data aggregated across the base, early and late adopter package categories at the three milestones in their evolution. The mean level comparisons across the package variant categories provide preliminary support for our hypotheses. For example, as shown in Table 4, there is a six-fold difference in the features added during the saturation phase between late adopters and early adopters (863 versus 137). We can also see that the software quality of early adopter variants is better than that of both the base package and the late adopter variant (lower backlog of errors, processing time, and SLA violations for late adopters and higher backlog of errors, processing times, and SLA violations for early adopters). But, customer satisfaction with the early adopter variants averaged 34 percent

TABLE 4
Mean Performance Metrics Across Package Variants

	Base Package Group			Early Adopter Variants			Late Adopter Variants		
	Before Takeoff	Growth Phase	Saturation Phase	Before Takeoff	Growth Phase	Saturation Phase	Before Takeoff	Growth Phase	Saturation Phase
Total Features Fulfilled	71	904	196	227	563	137	39	489	863
Features Growth Per Month	36.65	349.88	1062.82	110.81	558.84	870.48	21.65	189.37	1084.86
Customer Satisfaction	5.97	8.14	5.05	7.98	5.97	2.98	3.60	4.02	9.17
Backlog of Unresolved Errors	81	162	121	1261	1229	1107	6	38	99
Average Error Processing Time	15.98	16.09	15.90	33.77	22.20	27.67	2.20	3.48	5.2
% of Errors Exceeding SLA Response Time	13%	22%	14%	31%	28%	44%	0%	3%	6%

higher (7.98 versus 5.97) than the base package before takeoff. The customer satisfaction of the late adopter variants during this stage is about 40 percent lower than that of the base package (3.60 versus 5.97). In the final stages of the package's evolution the early adopter variants depict the highest levels of customer satisfaction, which is about 82 percent higher than that of the base package (9.17 versus 5.05). In contrast, the early adopter variants are associated with the lowest customer satisfaction levels, which are about 41 percent lower than that of the base package (2.98 versus 5.05). These mean comparison results are consistent with all of our hypotheses pairs.

To further validate our hypotheses by controlling for the functionality (and thereby size) differences across the different package variants at different points in time [53], [54], [55], we performed a series of statistical tests, which we discuss below.[11]

Hypotheses 1a and 1b: Adoption behavior and cumulative features. Our first set of hypotheses posited that early-stage adoption behavior predicts the cumulative functionality present in an individual customer's package variant at the end of the product's life span. To test this we derived a dependent variable Y as the difference in the number of features between a customer's package variant and the base package at the end of the product's lifespan ($\text{month} = 120$). Next, we defined a scalar independent variable X_s as the difference between the number of features in a package variant at the time of takeoff ($\text{month} = 35$) and the number of features in the base package at that time, and testing the regression Equations (1) and (2) specified below:

$$\mathrm{Y} = \alpha_0 + \alpha_1 \, X_s + \varepsilon \tag{1}$$

$$\mathrm{Y} = \beta_0 + \beta_1 \, (\text{Early adopter}) + \beta_2 \, (\text{Late adopter}) + \varepsilon \tag{2}$$

where, *Early adopter* and *Late adopter* are categorical variables indicating whether a specific package variant is in the early or late adopter category as explained in Section 5.1.

For our first set of hypotheses to be true, α_1 in regression Equation (1) needs to be negative, indicating that future features are negatively associated with the number of features in a package variant at the time the base package is taking off. Furthermore, β_1 and β_1 in the regression Equation (2) need to be statistically significant ($p < 0.05$) and $\beta_1 < 0$ and $\beta_2 > 0$.

Hypotheses 2a and 2b: Adoption behavior and software quality. Our second set of hypotheses related feature adoption behavior and software quality of the package variants. To test our hypotheses we performed the regression specified in Equation (3).

$$\mathrm{Q} = \gamma_0 + \gamma_{\mathrm{FD}} \, \mathrm{FD} + \gamma_2 \mathrm{P}_t + \gamma_3 \, \mathrm{P}_s + \gamma_4 \, (\text{Early adopter}) + \gamma_5 \, (\text{Late adopter}) + \varepsilon \tag{3}$$

where, Q is an inverse quality metric,[12] FD is the number of transactions deployed in a package variant, P_t is a categorical variable indicating whether a product is in a phase before takeoff, and P_s is a categorical variable indicating whether a product is in a phase after the saturation point.

To derive the coefficients of Equation (3) we utilize a panel data regression model controlling for serial correlations across the different months in the 10 year evolution of the software period. The hypotheses would be supported if $\gamma_4 > 0$, and $\gamma_5 < 0$, which indicates that early adopter variants have more defects and late adopter package variants have less number of defects as compared to the base package. Also, we expect $\gamma_{\mathrm{FD}} > 0$, indicating that defects tends to increase along with the software package's size.

Hypotheses 3a and 3b: Adoption behavior and customer satisfaction. The regression model specification to test our

11. We thank the associate editor and an anonymous reviewer for their detailed recommendations to explicitly clarify the confounding effects of size differences among the package variants and suggestions for some specific statistical models to test our hypotheses.

12. Note that our defect metrics are inverse quality metrics, that is, an increase in the defects indicates a decrease in software quality.

TABLE 5
Regression Results[t]

Model →	Equation (1)	Equation (2)	Equation (3)	Equation (4)
Variable	Y = (number of features in package variant – base package) at month = 120		Software Quality (Backlog of errors[#])	Customer Satisfaction
X_s (feature difference in month = 35)	−0.31 (0.006) α_1	—	—	—
Transactions (package size[†])	—	—	0.21 (0.000) γ_{FD}	0.9×10^{-4} (0.000) ζ_{FD}
Before takeoff	—	—	17.03 (0.600) γ_2	−2.19 (0.000) ζ_2
After saturation	—	—	−32.99 (0.255) γ_3	−3.17 (0.000) ζ_3
Early adopter	—	−173.22 (0.000) β_1	824.35 (0.000) γ_4	−2.01 (0.000) ζ_4
Late adopter	—	330.34 (0.000) β_2	−29.17 (0.004) γ_5	−4.10 (0.000) ζ_5
Early adopter X Before takeoff	—	—	—	4.20 (0.000) ζ_6
Early adopter X After saturation	—	—	—	0.13 (0.000) ζ_7
Late adopter X Before takeoff	—	—	—	1.79 (0.000) ζ_8
Late adopter X After saturation	—	—	—	8.22 (0.000) ζ_9
Adj. R-Squared	13.13	75.53	89.62	91.00
Model F-value / Wald Chi2 (Prob > F/Chi2)	8.25 (0.006)	75.01 (0.000)	3056 (0.000)	36933 (0.000)

Note:
[t]*Intercept terms were included in the regression estimations, but were not statistically significant at usual levels. Two-tailed P-values are listed in parentheses; P-values are derived from robust standard errors.*
[#]*Results are presented for the backlog of unresolved errors—an inverse measure of software quality. Regression results were similar for the other quality metrics (average error processing time, and % of errors exceeding SLA response time).*
[†]*Results are consistent for other measures of code size, KLOC and Function Points. The three software size measures, transactions, KLOC, and Function Points were highly correlated (Pearson correlation = 0.9).*

customer satisfaction hypotheses is shown in Equation (4).

$$\begin{aligned}\text{CSAT} = {} & \zeta_0 + \zeta_{\text{FD}}\text{FD} + \zeta_2\text{P}_\text{t} + \zeta_3\text{P}_\text{s} + \zeta_4\ (\text{Early adopter}) \\ & + \ \zeta_5\ (\text{Late adopter}) + \zeta_6\ (\text{Early adopter}) * \text{P}_\text{t} \\ & + \ \zeta_7\ (\text{Early adopter}) * \text{P}_\text{s} + \zeta_8\ (\text{Late adopter}) * \text{P}_\text{t} \\ & + \ \zeta_9\ (\text{Late adopter}) * \text{P}_\text{s} + \varepsilon\end{aligned} \tag{4}$$

where, CSAT is customer satisfaction and the other variables are same as in Equation (3).

To test predictions consistent with Hypotheses 3a and 3b we need to consider the marginal effects of early and late adoption conditions, taking into account the interaction terms involving the phase of a package's evolution (before and after saturation). Hypothesis 3a compares customer satisfaction between the early adopter and base adopter groups before takeoff and after saturation. This involves assessing the regression coefficients in Equation (4) by assigning relevant values for the categorical variables to yield the following four conditions: (1) base adopters before takeoff (early adopter = 0, late adopter = 0, $\text{P}_t = 1$, and $\text{P}_s = 0$), (2) early adopters before takeoff (early adopter = 1, late adopter = 0, $\text{P}_\text{t} = 1$, and $\text{P}_\text{s} = 0$), (3) base adopters after saturation (early adopter = 0, late adopter = 0, $\text{P}_\text{t} = 0$, and $\text{P}_\text{s} = 1$), and (4) early adopters after saturation (early adopter = 1, late adopter = 0, $\text{P}_\text{t} = 0$, and $\text{P}_\text{s} = 1$). Predictions consistent with hypothesis 3a on a conservative test that holds the package size the same across the conditions should yield $\zeta_4 + \zeta_6 > 0$ and $\zeta_4 + \zeta_7 < 0$ for the "before takeoff" and "after saturation" conditions, respectively.

Similarly, to test hypothesis 3b we assess the marginal effects for the following four conditions (1) base adopters before takeoff (early adopter = 0, late adopter = 0, $\text{P}_\text{t} = 1$, and $\text{P}_\text{s} = 0$), (2) late adopters before takeoff (early adopter = 0, late adopter = 1, $\text{P}_\text{t} = 1$, and $\text{P}_\text{s} = 0$), (3) base adopters after saturation (early adopter = 0, late adopter = 0, $\text{P}_\text{t} = 0$, and $\text{P}_\text{s} = 1$), and (4) late adopters after saturation (early adopter = 0, late adopter = 1, $\text{P}_\text{t} = 0$, and $\text{P}_\text{s} = 1$). For hypothesis 3b to be true a conservative test that holds package size the same across the conditions should yield $\zeta_5 + \zeta_8 < 0$ and $\zeta_5 + \zeta_9 > 0$ for the "before takeoff" and "after saturation" conditions respectively.

6 Results

The results of our regression analysis (Equations (1)–(4)) are presented in Table 5. Equations (1) and (2) were estimated using ordinary least-square regressions, and Equations (3) and (4) were estimated using balanced panel-data regressions adjusting for serial correlation and heteroskedasticity to derive robust standard errors. We performed the standard regression diagnostic tests, including tests to verify assumptions of normal distribution of errors and check the influence of multicollinearity and outliers. We did not detect any problems, and all of the models were statistically significant ($p < 0.01$), giving us confidence that the results reported in Table 5 are robust and unbiased estimates.

6.1 Hypotheses 1a and 1b: Adoption Behavior and Cumulative Features

Referring to results presented in Table 5 for Equation (1), we see that end of lifespan features are negatively correlated with the number of features at takeoff ($\text{month} = 35$). Also, for Equation (2) we can see that the coefficients for early

and late adopter variables are statistically significant, establishing that adoption behavior early in the package's lifecycle significantly determines the cumulative features at the end of the package's lifespan. Furthermore, we see that early adopters have a reduction in the end of lifespan cumulative number of features as compared to the base package (coefficient $\beta_1 = -173.22$). In contrast, late adopters have an increase in the end of lifespan cumulative number of features (coefficient $\beta_2 = 330.34$). These results show strong support for Hypothesis 1a and 1b.

6.2 Hypotheses 2a and 2b: Adoption Behavior and Software Quality

Referring to results presented in Table 5 for Equation (3), we can see that the number of transactions in a package variant is positively associated with the number of unresolved errors, as might be expected. The sequential phase of the package's lifespan (before takeoff or after saturation) did not have a statistically significant impact on the number of unresolved errors. However, controlling for package size and phase of lifespan, the results show that late adopters have a reduction in the number of unresolved errors (coefficient $\gamma_5 = -29.17$) as compared to the base package, whereas, early adopters have a large increase in unresolved errors (coefficient $\gamma_4 = 824.35$). These results show strong support for Hypotheses 2a and 2b, and establish that the software quality of early adopters is significantly lower than that of late adopter groups.

6.3 Hypotheses 3a and 3b: Adoption Behavior and Customer Satisfaction

To test whether our customer satisfaction hypotheses are supported we refer to results presented in Table 5 for Equation (4). We see that the marginal effect of early adoption before takeoff is positive ($\zeta_4 + \zeta_6 = -2.01 + 4.20 = 2.19 > 0$ in Equation (4)). The positive marginal effect indicates that early adopters have higher customer satisfaction levels than base adopters before takeoff. Applying the early adoption and after saturation condition to Equation (4), we see that the marginal effect on customer satisfaction is negative ($\zeta_4 + \zeta_7 = -2.01 + 0.13 = -1.88 < 0$). The negative marginal effect shows that early adopters have lower levels of customer satisfaction than the base adopters after the saturation point. Together, these results validate Hypothesis 3a.

Comparing the late adopters and the base adopters, we see that the marginal effects on customer satisfaction is negative before takeoff and positive after saturation ($\zeta_5 + \zeta_8 = -4.10 + 1.79 = -2.31 < 0$ and $\zeta_5 + \zeta_9 = -4.10 + 8.22 = 4.12 > 0$). This shows that late adopters have lower levels of customer satisfaction than base adopters before takeoff, but after the saturation point the customer satisfaction of late adopters becomes higher than the base adopters, validating Hypothesis 3b.

Also, we see that, all else equal, an increase in the number of features in a package variant is associated with an increase in customer satisfaction (positive and significant ζ_{FD} in Equation (4) in Table 5). However, as predicted by hypotheses 3a and 3b, there is a statistically significant difference in customer satisfaction across adopter groups, which also depends on the lifecycle stage of the product. For example, setting early adopters = 1 and before takeoff = 1, and holding the package size at 100 transactions in Equation (4),[13] we see that the net estimated effect on customer satisfaction is 0.9×10^{-2} ($= 4.2 - 2.01 - 2.19 + 0.9 \times 10^{-4} * 100$). That is, on average, the customer satisfaction of an early adopter is positive before takeoff, and the more transactions an early adopter adds to a package, the customer satisfaction before takeoff correspondingly increases. However, late adoption of the package has a negative effect, reducing customer satisfaction by 4.49 points ($4.10 - 2.19 + 0.9 \times 10^{-4} * 100 + 1.79 = -4.49$). This pattern reverses after the saturation point. Accounting for the growth of the package until the saturation point, we consider the case of a package size of 1,000 transactions across early and late adopters. After crossing the saturation point, we see that early adopters of the package have a negative customer satisfaction of $-4.96 (= -2.01 + 0.9 \times 10^{-4} * 1000 - 3.17 + 0.13)$ as compared to the positive customer satisfaction for late adopters of $1.04 (= -4.10 + 0.9 \times 10^{-4} * 1000 - 3.17 + 8.22)$. Overall, the results show strong support for hypotheses 3a and 3b.

6.4 Package Variants in Transition

We now proceed to examine the 21 "transitional" candidate cases, which refer to situations where customers alter their feature adoption behavior at different stages of a software package's evolution and therefore could not be appropriately examined as "pure" cases. In such situations a package variant transitions from one of the canonical patterns (base package, early adopter, or late adopter) to a different one. In order to examine such transitions we separated transitional candidates into 14 cases with unidirectional transitions, that is, a one-time movement, and seven cases with multiple transitions in their lifespan.

Any significant change in the feature adoption behavior of a customer alters the package growth trajectory in significant ways. For example, an early adopter customer could invest in refactoring activities, which, in turn, could potentially increase the rate of feature growth during the saturation phase, and tends to push the package variant's growth towards that of the base version. We observed three cases in our data set where such a transition occurred before the takeoff milestone, and four cases with such a transition after the takeoff milestone.

A late adopter package variant might transition to an aggressive growth mode if customers begin to add functionality rapidly, which pushes its growth trajectory towards that of the base version. We observed two cases where late adopter variants transitioned to the base package group before takeoff. In five other cases, package variants in the base group moved all the way to the early adopter variant category before takeoff occurred. Apart from these one-time transitions we also observed seven cases where customers transitioned across the base package growth trajectory multiple times in the 120 month lifespan of the package.

By considering a transition as the unit of analysis, we examined the impact of a shift in feature adoption strategy on customer satisfaction in the 21 transitional cases in our sample. Transitions before the takeoff of a product could be seen as experimentation and a learning exercise carrying

13. Although package size varies across the early and late adopter groups we hold it constant in this illustration for simplicity.

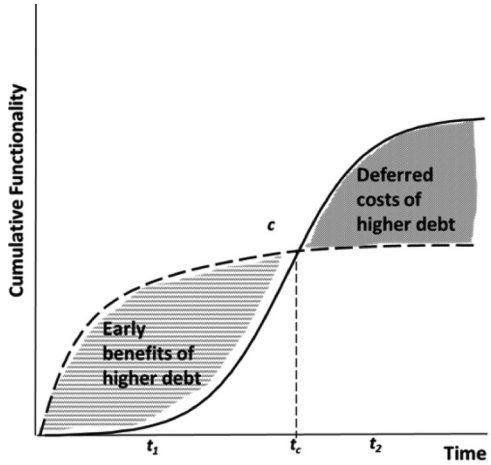

Fig. 6. Expected growth in higher-debt scenario.

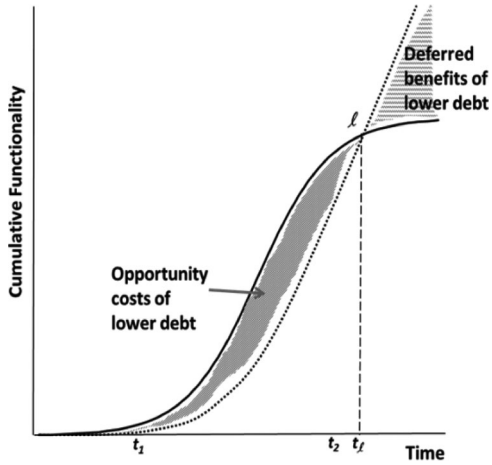

Fig. 7. Expected growth in lower-debt scenario.

relatively lower risks as compared to the transitions in product growth after the takeoff milestone. However, transitions after a package has taken off run the risk of contradicting original customer expectations that led customers to adopt the package in the first place [56]. For example, a transition from the early adopter to the base package might alienate business customers who preferred the rapid expansion of functionality. Similarly, a transition from late adopter towards the base package may not satisfy business customers who prefer to be selective in their functionality choices (late adopters), but demand higher quality. Such fluctuations in software performance that defy customer expectations have been shown to have a negative impact on customer satisfaction in prior research [49], [50]. Thus, we could expect a drop in customer satisfaction when a transition happens in the growth trajectory of package variants that have taken off. Also, such a drop in customer satisfaction might be more pronounced for transitions after takeoff as compared to the transitions before takeoff.

Since the number of transition cases is small, we do not have a sufficient basis for rigorous testing of the above postulates. However, in order to provide at least some exploratory results we calculated the difference in customer satisfaction levels between the package groups experiencing transition before and after the takeoff milestone after normalizing for package size. Our results indicate that customer satisfaction levels dropped by more than 91 percent for transitions after takeoff, as compared to a decrease of 66 percent for transitions before takeoff. Our analysis also revealed that for the transitions that happened after takeoff, customer satisfaction levels that dropped immediately after the transition did not improve until the end of the lifespan of the package. In contrast, for transitions that happened before takeoff, the initial drop in customer satisfaction tended to recover just after the takeoff milestone. These results suggest the desirability of judiciously timing changes to adoption behavior, a practice which we discuss further in Section 7.

7 Discussion

7.1 Inferring Costs and Benefits of Technical Debt

Our results provide a basis for interpreting the effects of customer feature adoption behavior on software quality and customer satisfaction as the effects of technical debt in enterprise software. Early addition of features before takeoff, especially through custom modification of enterprise package software code, is associated with poor software quality throughout the package's lifespan and can be interpreted as the cost of accumulating technical debt. However, incurring this technical debt does bring about short-term business benefits, such as an improvement in customer satisfaction proportional to the number of transactions (features) added.

Similarly, the late adopter behavior of avoiding immature ("beta") features and custom modification of features can be interpreted as the costs of avoiding technical debt (or perhaps as accumulating a "technical credit"). The cost of avoiding technical debt manifests itself in poor customer satisfaction scores before takeoff. However, the strategy of avoiding technical-debt pays off in the longer run in terms of significantly higher software quality of the package throughout its lifespan and further endows a customer with the ability to continue adding features at a faster pace during the later stage of the package's lifecycle.

We illustrate this interplay between costs and benefits of technical debt in Figs. 6 and 7 (not drawn to scale) as effects proportional to the relative difference in cumulative features between the package variants and vendor's base package.

Fig. 6 shows that the functionality growth of a higher-debt variant (dashed curve) is expected to takeoff and saturate much earlier than the base package (S-curve). The growth trajectories of the base version and the higher-debt variant intersect at point *c*, corresponding to time t_c in the lifespan of the implementations. The area between the curves before t_c reflects the extent of benefits that a higher-debt regime offers, which comes at a cost reflected by the area between the curves after t_c. Thus, this model reflects the underlying optimization problem: customers who reap the benefits of early functionality that comes at the expense of software development standards must pay back these costs over the longer term [1], [2], [5]. The overall value of incurring debt by a customer, then, will depend on the rate of functionality growth of the customer's package variant, the customer's ability to derive business value from the added functionality, and the lifespan of the software package.

Referring to Fig. 7, the growth trajectories of the lower-debt variant (dotted curve) and the base package (S-curve) intersect at *l*, corresponding to time t_l. The area between the

curves before t_l reflects the extent to which the lower-debt regime has lost business functionality, which can be seen as opportunity costs associated with lower-debt variants. These costs can be offset in the long-term, in part, by the benefits of a slower saturation of the lower-debt growth. Such benefits of the lower-debt variants are reflected in Fig. 7 as the area between the curves after t_l. Thus, our generalized conceptualization incorporates the underlying optimization problem of lower-debt growth in the following way: customers who incur costs due to an early focus on reliability and quality of a software package are positioned to reap the benefits of a functionality takeoff during the later stages of the package's lifecycle [13], [17]. The value of a lower-debt strategy for a customer, then, depends on the rate of functionality growth of a customer's package variant, the customer's ability to derive business value from the increased reliability and end-stage functionality, and the lifespan of the package.

7.2 Managerial Implications for Software Development Investments

Our empirical results and the general conceptualization based on our theory establish the underlying tradeoffs between the costs and benefits of accumulating technical debt at different stages of a package's evolution. The evolutionary model we conceptualized highlights several opportunities for development investments and transitions in functionality growth trajectories of the package variants throughout their lifespan. Based on our model and results, we highlight six key development decision points throughout the evolution of a software package—three on the higher-debt trajectory (HD1-3 in Fig. 8) and three on the lower-debt trajectory (LD1-3 in Fig. 8).

The first development decision points on the higher-debt and lower-debt growth trajectories (HD1 and LD1, respectively) pertain to fruitful opportunities for software managers to alter their existing package growth trajectories before the general takeoff of the base version supplied by the vendor in the marketplace (before t_1). Since the package has not widely taken off in the market, managers could treat their debt-taking or debt-avoiding strategies as early experiments only, and account for any associated costs as learning investments. Altering the growth trajectories at HD1 and LD1 would also be relatively easier, as these occur in the early phase of the package lifecycle when the code base is relatively smaller. Further, as our results indicated, any drop in customer satisfaction levels is likely to be able to be recovered once the package takes off.

Once a package takes off (after t_1), the package vendor typically provides development roadmaps and details about anticipated release of features in the future, which could be utilized for calculating the potential debt-obligations of higher-debt package variants and the losses due to lack of functionality in lower-debt package variants. Thus, HD2 is a good point to get a reliable estimate of the extent of the debt obligation of the higher-debt variant that needs to be paid off at some point in the future. Similarly, for the lower-debt product variant, LD2 is a decision point where a software manager could reliably estimate the opportunity costs of missed functionality and determine whether to alter growth paths.

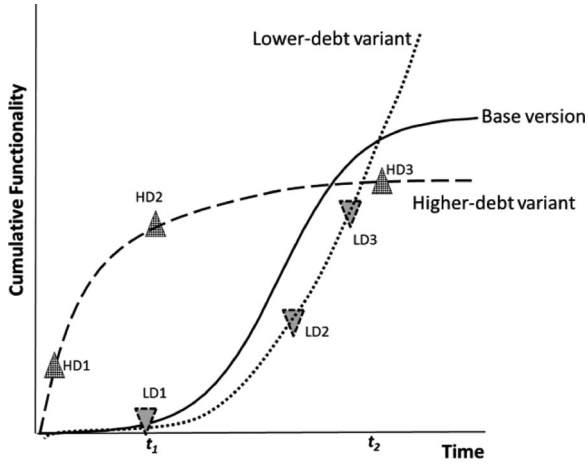

Fig. 8. Package variants and development investment decisions. HD1, HD2, and HD3 are decision points in the high-debt product evolution trajectory. LD1, LD2, and LD3 are decision points in the low-debt product evolution trajectory.

HD3 occurs at the end stage of high technical debt growth trajectory when the performance of the higher-debt variant has saturated. If the package variant is retired at this point (say, for example, substituted by a new technology), then, the accumulated debt could be partially written off. However, if the customer firm is not able to retire the package variant at this stage, it faces a debt obligation and a development investment targeted at reducing the accumulated technical debt, for example, through investments in refactoring, could potentially prolong the life of the package.

Similarly, LD3 is the end stage decision point on the lower-debt growth pattern, which presents an opportunity for a customer to recover lost value due to lack of functionality. If the customer could rapidly ramp-up its development at this stage, it would be able to reap significant benefits due to increased performance of its package variant at a relatively cheaper cost as compared to the higher-debt and base packages, as well as compared to any emerging technological substitutes.

In order to judiciously time the above-mentioned development decisions there is a need for customers of enterprise software packages to periodically benchmark their current growth trajectory with the base version supplied by the vendor. At a minimum we recommend a systematic comparison at the following key timelines: (1) the first instance when a package vendor announces a package roadmap, (2) whenever the cost of quality exceeds the cost of new functionality implementation, and (3) the first instance when functionality growth slows below the established threshold (see Section 5.1 for the heuristic on threshold calculation). A chief criterion that influenced our choice of these timelines was that managers need to be able to observe key events related to the three important milestones in the evolution of a software package considered in our model, namely, the takeoff, inflection, and saturation points.

A key observable event that coincides with the takeoff of a package is the roadmap announcement by the package vendor, which is often well-publicized. Typically a vendor announces a package roadmap when their installed base has reached the critical size for the firm to reap economies of scale in both product development and support [46], [47].

The roadmap announcement helps the vendor to plan support and development activities. We recommend that customers invest in debt-reducing or functionality-increasing development activities by benchmarking their package growth trajectories with that of the vendor's publicly announced roadmaps.

The second milestone in our model, the inflection point, occurs during the active growth phase of a package. Since it is difficult to assess the inflection point before the package is retired, we recommend a heuristic to assess whether the growth trajectory of a package is starting to reverse its concavity. One suggested heuristic we propose is based on the ratio of costs expended by a customer on maintenance activities (cost of quality) and costs of implementing new functionality. The first instance when maintenance costs of a package exceed costs of developing new functionality indicates that the structural complexity of the package at a customer site is starting to be a burden on future functionality growth. Thus, we recommend customers to keep track of this metric to time their major development decisions after the widespread takeoff of a software package.

The final milestone in our model, the saturation point, is easily observable by customers as its identification is based on historical growth rate thresholds. As soon as the growth rate of functionality observed in a time period decreases beyond the threshold of growth rates observed in previous time periods we recommend customers to formulate their development strategies for the saturation phase. For example, customers would need to assess if they aim to prolong the life of the package by investing in appropriate development activities, such as refactoring, in order to reduce structural complexity or if they could replace the saturating package with some new technologies available in the market.

7.3 Contributions and Further Research

This study makes several contributions. First, the technical debt accumulation model advanced in this study adopts an evolutionary perspective, allowing for the analysis of technical debt over the entire lifespan of a software package, moving beyond the prevalent cross-sectional models in the literature. Second, our modeling approach emphasizes a balanced analysis of the costs and benefits of technical debt, and thereby facilitates an unbiased comparison of the debt-accumulating and debt-avoiding strategies pursued by end customers over a multi-year period. Finally, this study makes an empirical contribution by validating the theoretical model using longitudinal data from real world implementations of a commercial software package.

We note several opportunities for further research. First, expanding the analysis presented in this paper by investigating the underlying reasons for debt-taking or debt-avoiding behavior of customers could help account for possible customer heterogeneity and explain the transitions in functionality growth trajectories in a more detailed way than was possible with the data set available to us here. Second, the empirical investigation of the relative effects of debt-reducing and debt-increasing growth transitions on customer satisfaction at different levels of structural complexity and at different lifecycle stages of an enterprise software package could possibly be expanded with a larger sample size. Third, building on recent efforts to address technical debt in software ecosystems [23], the model developed in this study could be extended to address software package ecosystems where growth trajectories of several components of the ecosystems are intertwined with each other. Such an analysis would help firms assess the costs and benefits of technical debt-taking or debt-avoiding behavior for the entire software systems infrastructure supporting their business operations. Fourth, in order to expand an understanding of the dynamics of software development and refactoring investments for early and late adopters, scholars could conduct in-depth qualitative case studies [57] at specific software package implementation sites. Finally, future research could compare and contrast the technical debt accumulation patterns in a wider range of types of software packages, along with the analysis of different types of refactoring activities, which would aid the broader generalization of the results reported in this paper.

7.4 Conclusion

In this study we developed and validated a theory of technical debt accumulation in a commercial enterprise software package that explicitly measures the impact of tradeoffs between functionality growth and both software quality and customer satisfaction at different stages of the package's multi-year lifespan. Our theory and empirical results establish the different dynamics of costs and benefits of technical debt for early adopters and late adopters of enterprise software packages. For early adopters of an enterprise software package the benefits of initial higher customer satisfaction due to early addition of functionality before the package's takeoff in the wider population is offset by the costs of technical debt that add up in the form of poor software quality throughout the package's lifespan. For late adopters the costs of avoiding technical debt manifest themselves in the form of poor customer satisfaction scores before takeoff, which are subsequently offset in the longer run through higher software quality and the ability to continue adding functionality at a faster pace during the later stage of the package's lifecycle.

The evolutionary models of technical debt advanced in this paper move beyond the prevalent cross-sectional approaches in the literature, taking an important step towards establishing a balanced perspective of managing technical debt in enterprise software packages.

Acknowledgments

The authors thank the Associate Editor and three anonymous referees for their helpful comments on the original draft of this manuscript. In addition, we thank S. Daniel, D. Eargle, S. Janansefat, N. Nagappan, S. Singh, J. Woodard, T. Zimmerman, seminar participants at Microsoft Research, Redmond, WA, and workshop participants at the 4th *International Workshop on Managing Technical Debt*, San Francisco, CA for their comments on earlier versions of the paper.

References

[1] W. Cunningham, "The wycash portfolio management system," in *Proc. Object-Oriented Program. Syst., Lang., Appl.*, Vancouver, B. C., Canada, 1993, pp. 29–30.

[2] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazworka, "Managing technical debt in software-reliant systems," in *Proc. FSE/SDP Workshop Future Soft. Eng. Res.*, Santa Fe, NM, USA, 2010, pp. 47–52.

[3] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *IEEE Soft.*, vol. 29, no. 6, pp. 18–21, Nov./Dec. 2012.

[4] A. MacCormack, R. Verganti, and M. Iansiti, "Developing product on "internet time": The anatomy of a flexible development process," *Manage. Sci.*, vol. 47, no. 1, pp. 133–150, 2001.

[5] C. Seaman and Y. Guo, "Measuring and monitoring technical debt," in *Proc. Adv. Comput.*, 2011, pp. 25–46.

[6] E. Lim, N. Taksande, and C. Seaman, "A balancing act: What software practitioners have to say about technical debt," *IEEE Soft.*, vol. 29, no. 6, pp. 22–27, Nov./Dec. 2012.

[7] C. F. Kemerer and S. Slaughter, "An empirical approach to studying software evolution," *IEEE Trans. Softw. Eng.*, vol. 25, no. 4, pp. 493–509, Jul./Aug. 1999.

[8] D. Harter, C. F. Kemerer, and S. Slaughter, "Does software process improvement reduce the severity of defects? a longitudinal field study," *IEEE Trans. Softw. Eng.*, vol. 38, no. 4, pp. 810–827, Jul./Aug. 2012.

[9] M. J. LaMantia, Y. Cai, A. MacCormack, and J. Rusnak, "Analyzing the evolution of large-scale software systems using design structure matrices and design rule theory: Two exploratory cases," in *Proc. 7th Working IEEE/IFIP Conf. Softw. Archit.*, Vancouver, BC, Canada, 2008, pp. 83–92.

[10] N. Ramasubbu and C. F. Kemerer, "Towards a model for optimizing technical debt in software products," in *Proc. 4th Int. Workshop Managing Tech. Debt*, San Francisco, CA, USA, May 20, 2013, pp. 51–54.

[11] V. Mahajan, E. Nullier, and F. M. Bass, "Diffusion of new products: Empirical generalizations and managerial uses," *Marketing Sci.*, vol. 14, no. 3, pp. G79–G88, 1995.

[12] P. N. Golder and G. J. Tellis, "Will it ever fly? modeling the growth of new consumer durables," *Marketing Sci.*, vol. 16, no. 3, pp. 256–270, 1997.

[13] M. Fowler. (29 July 2013). "Technical debt," [Online]. Available: http://martinfowler.com/bliki/TechnicalDebt.html.

[14] C. Izurieta, A. Vetro, N. Zazworka, Y. Cai, C. Seaman, and F. Shull, "Organizing the technical debt landscape," in *Proc. 3rd Int. Workshop Managing Tech. Debt*, Zurich, Switzerland, 2012, pp. 23–26.

[15] T. Klinger, P. Tarr, P. Wagstrom, and C. Williams, "An enterprise perspective on technical debt," in *Proc. 2nd Int. Workshop Managing Tech. Debt*, Honolulu, Hawaii, USA, 2011, pp. 35–38.

[16] T. Theodoropoulos, M. Hofberg, and D. Kern, "Technical debt from the stakeholder perspective," in *Proc. 2nd Int. Workshop Managing Tech. Debt*, Honolulu, Hawaii, USA, 2011, pp. 43–46.

[17] J. Woodard, N. Ramasubbu, T. 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.

[18] N. Ernst, "Technical debt and requirements," in *Proc. 3rd Int. Workshop Managing Tech. Debt*, Zurich, Switzerland, 2012, pp. 61–64.

[19] S. McConnell. (2013). "Managing technical debt," Presented at the 4th Int. Workshop Tech. Debt, Keynote Presentation, San Francisco, CA, USA, 2013.

[20] C. Y. Baldwin and A. MacCormack, "Finding technical debt in platform and network architectures," Presented at the Mathworks Inc., Natick, MA, USA, 2011.

[21] J. Brondum and L. Zhu, "Visualising architectural dependencies," in *Proc. 3rd Int. Workshop Managing Tech. Debt*, Zurich, Switzerland, 2012, pp. 7–14.

[22] J. D. Morgenthaler, M. Gridnev, R. Sauciuc, and S. Bhansali, "Searching for build debt," in *Proc. 3rd Int. Workshop Managing Tech. Debt*, Zurich, Switzerland, 2012, pp. 1–6.

[23] J. D. McGregor, J. Y. Monteith, and J. Zhang, "Technical debt aggregation in ecosystems," in *Proc. 3rd Int. Workshop Managing Tech. Debt*, Zurich, Switzerland, 2012, pp. 27–30.

[24] R. Gomes, C. Siebra, G. Tonin, A. Cavalcanti, F. Q. B. D. Silva, A. L. M. Santos, and R. Marques, "An extraction method to collect data on defects and effort evolution in a constantly modified system," in *Proc. 2nd Int. Workshop Managing Tech. Debt*, Honolulu, Hawaii, USA, 2011, pp. 27–30.

[25] B. Boehm, "Assessing and avoiding technical debt," Presented at the 3rd Int. Workshop Managing Tech. Debt, Zurich, Switzerland, 2012.

[26] J. D. Groot, A. Nugroho, T. Back, and J. Visser, "What is the value of your software?" in *Proc. 3rd Int. Workshop Managing Tech. Debt*, Zurich, Switzerland, 2012, pp. 37–44.

[27] F. A. Fontana, V. Ferme, and S. Spinelli, "Investigating the impact of code smells debt on quality code evaluation," in *Proc. 3rd Int. Workshop Managing Tech. Debt*, Zurich, Switzerland, 2012, pp. 15–22.

[28] B. Curtis, J. Sappidi, and A. Szynkarski, "Estimating the principal of an application's technical debt," *IEEE Soft.*, vol. 29, no. 6, pp. 34–42, Nov./Dec. 2012.

[29] N. Zazworka, C. Seaman, F. Shull, and M. Shaw, "Investigating the impact of design debt on software quality," in *Proc. 2nd Int. Workshop Managing Tech. Debt*, Honolulu, Hawaii, USA, 2011, pp. 17–23.

[30] A. Nugroho, J. Visser, and T. Kuipers, "An empirical model of technical debt and interest," in *Proc. 2nd Int. Workshop Managing Tech. Debt*, Honolulu, Hawaii, USA, 2011, pp. 1–8.

[31] J. Heintz and I. Gat, "Investigating from assessment to reduction: Reining in millions," Presented at the 2nd Int. Workshop Managing Tech. Debt, Honolulu, Hawaii, USA, 2011.

[32] N. Brown, R. L. Nord, I. Ozkaya, and P. Kruchten, "Quantifying the value of architecting within agile software development via technical debt analysis," Presented at the 2nd Int. Workshop Managing Tech. Debt, Honolulu, Hawaii, USA, 2011.

[33] J. Bohnet and J. Döllner, "Monitoring code quality and development activity by software maps," in *Proc. 2nd Int. Workshop Managing Tech. Debt*, Honolulu, Hawaii, USA, 2011, pp. 9–16.

[34] K. J. Sullivan, W. G. Griswold, Y. Cai, and B. Hallen, "The structure and value of modularity in software design," in *Proc. 8th Euro. Softw. Eng. Conf. Held Jointly 9th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, Vienna, Austria, 2001, pp. 99–108.

[35] Y. Guo and C. Seaman, "A portfolio approach to technical debt management," in *Proc. 2nd Int. Workshop Managing Tech. Debt*, Honolulu, Hawaii, USA, 2011, pp. 31–34.

[36] W. Nichols, "A cost model and tool to support quality economic trade-off decisions," Presented at the 2nd Int. Workshop Managing Tech. Debt, Honolulu, Hawaii, USA, 2011.

[37] N. Zazworka, C. Seaman, F. Shull, and M. Shaw, "Prioritizing design debt investment opportunities," in *Proc. 2nd Int. Workshop Managing Tech. Debt*, Honolulu, Hawaii, USA, 2011, pp. 39–42.

[38] J. L. Letouzey and M. Ilkiewicz, "Managing technical debt with the Sqale method," *IEEE Softw.*, vol. 29, no. 6, pp. 44–51, Nov./Dec. 2012.

[39] C. Seaman, Y. Guo, C. Izurieta, Y. Cai, N. Zazworka, F. Shull, and A. Vetro, "Using technical debt data in decision making: Potential decision approaches," in *Proc. 3rd Int. Workshop Managing Tech. Debt*, Zurich, Switzerland, 2012, pp. 45–48.

[40] W. Snipes and B. Robinson, "Defining the decision factors for managing defects: A technical debt perspective," in *Proc. 3rd Int. Workshop Managing Tech. Debt*, Zurich, Switzerland, 2012, pp. 54–60.

[41] C. F. Kemerer and B. Porter, "Improving the reliability of function point measurement: An empirical study," *IEEE Trans. Softw. Eng.*, vol. 18, no. 10, pp. 1011–1024, Nov. 1992.

[42] N. C. Olsen, "Survival of the fastest: Improving service velocity," *IEEE Softw.*, vol. 12, no. 5, pp. 28–38, Sep. 1995.

[43] M. Lindvall, D. Muthig, A. Dagnino, C. Wallin, M. Stupperich, D. Kiefer, J. May, and T. Kähkönen, "Agile software development in large organizations," *IEEE Softw.*, vol. 37, no. 12, pp. 26–34, Dec. 2004.

[44] V. Mahajan, E. Muller, and Y. Wind, "New-product diffusion models: From theory to practice," *New-Product Diffusion Models*, V. Mahajan, E. Muller, and Y. Wind, Eds. New York, NY, USA: Springer, 2000.

[45] E. M. Rogers, *Diffusion of Innovations*. New York, NY, USA: Free Press, 2003.

[46] I. J. Patrick and A. E. Echols, "Technology roadmapping in review: A tool for making sustainable new product development decisions," *Technol. Forecasting Soc. Change*, vol. 71, no. 1/2, pp. 81–100, 2004.

[47] P. Groenveld, "Roadmapping integrates business and technology," *Res.-Technol. Manage.*, vol. 50, no. 6, pp. 49–58, 2007.

[48] S. G. Erick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, "Does code decay? assessing the evidence from change management data," *IEEE Trans. Softw. Eng.*, vol. 27, no. 1, pp. 1–12, Jan. 2001.
[49] N. Ramasubbu, S. Mithas, and M. S. Krishnan, "High tech, high touch: The effect of employee skills and customer heterogeneity on customer satisfaction with enterprise system support services," *Decis. Support Syst.*, vol. 44, no. 2, pp. 509–523, 2008.
[50] S. Kekre, M. S. Krishnan, and K. Srinivasan, "Drivers of customer satisfaction for software products: Implications for design and service support," *Manage. Sci.*, vol. 41, no. 9, pp. 1456–1470, 1995.
[51] G. Succi, L. Benedicenti, and T. Vernazza, "Analysis of the effects of software reuse on customer satisfaction in an rpg environment," *IEEE Trans. Soft. Eng.*, vol. 27, no. 5, pp. 473–479, May 2001.
[52] C. F. Kemerer and M. C. Paulk, "The impact of design and code reviews on software quality: An empirical study based on psp data," *IEEE Trans. Softw. Eng.*, vol. 35, no. 4, pp. 534–550, Jul./Aug. 2009.
[53] G. Gill and C. F. Kemerer, "Cyclomatic complexity density and software maintenance productivity," *IEEE Trans. Softw. Eng.*, vol. 17, no. 12, pp. 1284–1288, Dec. 1991.
[54] S. Chidamber, D. P. Darcy, and C. F. Kemerer, "Managerial use of object oriented software metrics," *IEEE Trans. Softw. Eng.*, vol. 24, no. 8, pp. 629–639, Aug. 1998.
[55] D. I. K. Sjoberg, A. Yamashita, B. Anda, A. Mockus, and T. Dyba, "Quantifying the effect of code smells on maintenance effort," *IEEE Trans. Softw. Eng.*, vol. 39, no. 8, pp. 1144–1156, Aug. 2013.
[56] E. Bridges, C. K. B. Yim, and R. A. Briesch, "A high-tech product market share model with customer expectations," *Marketing Sci.*, vol. 14, no. 1, pp. 61–81, 1995.
[57] P. Runeson and M. HÖst, "Guidelines for conducting and reporting case study research in software engineering," *Empir. Softw. Eng.*, vol, 14, no. 2, pp. 131–164, 2009.

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 assistant professor at the Katz Graduate School of Business at the University of Pittsburgh. Previously, he was an assistant professor at the School of Information Systems at the Singapore Management University. Prior to his academic career, he was a senior developer at SAP AG and CGI Inc. His current research interests include software engineering economics with a focus on globally distributed product development and service delivery; the design, implementation, and governance of enterprise information systems; end-user interaction and user-led innovation; and IT strategy and generating business value from IT. He is a member of the IEEE.

Chris F. Kemerer received the BS degree, *magna cum laude*, from the Wharton School at the University of Pennsylvania, and the PhD degree from Carnegie Mellon University. He is the David M. Roderick chair in information systems at the Katz Graduate School of Business at the University of Pittsburgh, and an adjunct professor 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 as the departmental editor for information systems at *Management Science*, and was recently named as an INFORMS Information Systems Society Distinguished Fellow. He is a member of the IEEE Computer Society and is a past associate editor of *IEEE Transactions on Software Engineering*, and is an ISI/Thomson Reuters Highly Cited Researcher in Computer Science. He is a member of the IEEE.