Managing Technical Debt in

Enterprise Software Packages

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

**Abstract**— We develop an evolutionary model of technical debt accumulation to facilitate a rigorous and balanced analysis of the benefits and costs of incurring technical debt at different lifecycle stages of a large commercial enterprise software package. Our model focuses on the optimization problem involved in managing technical debt, and illustrates the different tradeoff patterns between customer satisfaction and software quality under low and high technical debt scenarios at different lifecycle stages of the software package. We empirically validate the model utilizing a rich, longitudinal dataset drawn from 69 customer installations of the software package, spanning its entire 10 year lifecycle. We then utilize the empirically-validated model to derive actionable policies for managing technical debt and design investments in enterprise software product development.

**Index Terms**—technical debt, software platforms, customer satisfaction, software quality, customer adoption, COTS, software evolution, software maintenance, longitudinal data.

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

————————————————

1. N. Ramasubbu is with the Joseph M. Katz Graduate School of Business, University of Pittsburgh, Pittsburgh, PA 15260. E-mail: narayanr@pitt.edu.
2. 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.

—————————— ◆ ——————————

# 1 Introduction

T

he term “technical debt” is used as a metaphor to refer to the likely long-term costs associated with software design shortcuts taken by programmers to deliver short-term business benefits [[1](#_ENREF_1), [2](#_ENREF_2), [3](#_ENREF_3)]. For example, it is well documented that software developers often face a resource crunch—a situation where developers have to satisfy multiple demanding goals, such as improved functionality of software packages under tight budgets and aggressive delivery deadlines [[4](#_ENREF_4)]. In these situations and others developers tend to tradeoff potential longer-term benefits of software design investments for the immediate short-term business benefits of rapidly adding functionality by taking design shortcuts (i.e., incurring technical debt). There is a growing interest in the software engineering and technology management research communities to use the technical debt metaphor to derive appropriate decision models for software product management [[2](#_ENREF_2), [5](#_ENREF_5), [6](#_ENREF_6)].

We reviewed the emerging literature on technical debt and provide a summary of the literature in Table 1. We note four interrelated research streams on technical debt: (a) taxonomy, frameworks, and perspectives (b) measuring and visualizing technical debt, (c) valuation of the impacts of technical debt, and (d) managerial decision frameworks. 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 to monitor and avoid 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]](#footnote-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 analyses of the impact of technical debt on performance 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](#_ENREF_7), [8](#_ENREF_8), [9](#_ENREF_9), [10](#_ENREF_10)].

In this paper we add to the understanding of technical debt in practice by developing and testing an evolutionary model of technical debt accumulation by specifically accounting for the different pathways in which a software package evolves contingent on the way technical debt is managed by the customers of the package vendor[[2]](#footnote-2). To incorporate this evolutionary perspective we build on a widely-shared insight from the management research literature on the adoption and diffusion of products that consumer adoption patterns typically follow an “S”-shaped curve with distinct takeoff and saturation points [[11](#_ENREF_11), [12](#_ENREF_12)]. By anchoring our model development on this empirically-validated and generalized pattern of longitudinal customer adoption, our approach facilitates a rigorous comparison of the individual effects of incurring debt—both costs and benefits—against a reliable benchmark that is available at all stages of a multi-year software lifespan.

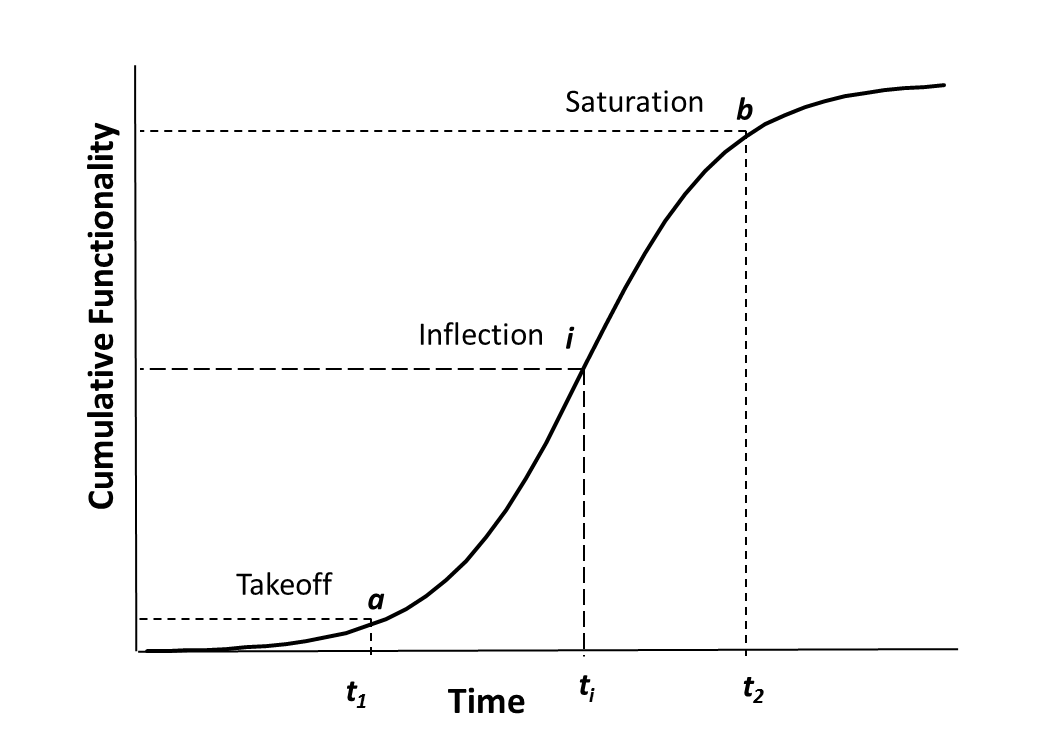


Fig. 1. Theoretical Growth in a Base Software Package

To empirically test our conceptual model we utilized a rich, longitudinal dataset spanning the 10 year lifecycle of a large commercial enterprise software package along with detailed performance data from 69 large customers (Fortune-500 firms) who implemented and sometimes customized the package for their business. Utilizing detailed package configurations from the 69 implementations, we traced the different evolutionary pathways of technical debt accumulation and compared it with the evolution of the base version supplied by the vendor. Utilizing customer satisfaction and software quality as performance metrics, we illustrate 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-debt scenarios that can help managers to make informed decisions on design investments such as refactoring or architectural improvements at different points in the lifespan of the software package.

TABLE 1  
Technical Debt Literature

|  |  |  |
| --- | --- | --- |
| No | Theme | Literature Reference |
| 1 | Taxonomy, frameworks, and perspectives | [[1](#_ENREF_1), [2](#_ENREF_2), [3](#_ENREF_3), [6](#_ENREF_6), [13](#_ENREF_13), [14](#_ENREF_14), [15](#_ENREF_15), [16](#_ENREF_16), [17](#_ENREF_17), [18](#_ENREF_18), [19](#_ENREF_19)] |
| 2 | Measuring and visualizing technical debt | [[5](#_ENREF_5), [20](#_ENREF_20), [21](#_ENREF_21), [22](#_ENREF_22), [23](#_ENREF_23), [24](#_ENREF_24), [25](#_ENREF_25)] |
| 3 | Valuing the impact of technical debt | [[26](#_ENREF_26), [27](#_ENREF_27), [28](#_ENREF_28), [29](#_ENREF_29), [30](#_ENREF_30), [31](#_ENREF_31), [32](#_ENREF_32), [33](#_ENREF_33), [34](#_ENREF_34)] |
| 4 | Managerial decision frameworks | [[17](#_ENREF_17), [20](#_ENREF_20), [25](#_ENREF_25), [35](#_ENREF_35), [36](#_ENREF_36), [37](#_ENREF_37), [38](#_ENREF_38), [39](#_ENREF_39), [40](#_ENREF_40)] |

The rest of the paper is organized as follows. In section 2 we develop our conceptual model, elaborating how we integrate the theoretical pattern of customer adoption of a software product with the different trajectories of technical debt accumulation in the model. In section 3 we elaborate the empirical setting in which we collected our data for validating our conceptual model. We present our analysis methodology and hypotheses test results in section 4, and discuss the broader implications of these results for practice and for future research in section 5.

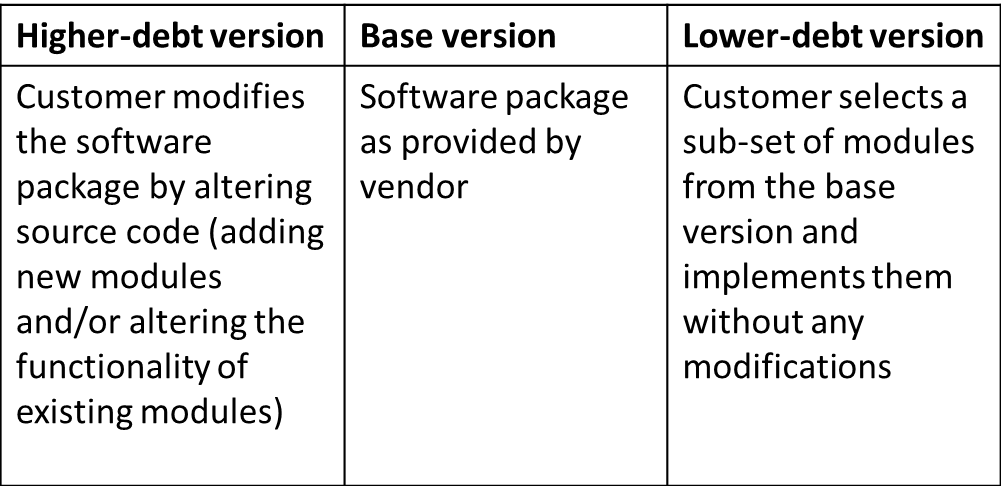
# 2 Conceptual Model

## 2.1 Generalized Customer Adoption Pattern

The cumulative distribution of adoption of a wide variety of consumer products over time has been shown to follow an S-shaped pattern [[11](#_ENREF_11), [12](#_ENREF_12), [41](#_ENREF_41), [42](#_ENREF_42), [43](#_ENREF_43), [44](#_ENREF_44), [45](#_ENREF_45)]. Such a product growth model has been established as an empirical generalization—a regularity that repeats over a variety of circumstances, and has been extensively utilized in the management research literature on diffusion of innovations [[41](#_ENREF_41), [46](#_ENREF_46)]. 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.

Figure 1 presents such a pattern of product growth that matches with the general consumer adoption of the product; point *a* in the curve (takeoff point corresponding to time *t1*) shows the point at which the customer adoption of the product “takes off” after a slow initial growth; point *b* in the curve (saturation point corresponding to time *t2*) shows the point at which the customer adoption of the product begins to decline. Formally, the growth of functionality can be modeled using a logistic growth curve defined by F = K / 1 + e –(x+yt), where *F* is the functionality added, *K* is the maximum equilibrium value of the functionality, *t* is the time variable, *y* is the rate of functionality growth coefficient, and *x* is a constant used to position the curve on the time scale. In the logistic growth model, the inflection point *i* on the curve, corresponding to time *ti* lies at the half way mark of the total functionality present in a product at the end of its lifespan.

In the context of enterprise software packages, the customer adoption pattern of the base version of a package provided by the vendor serves as a reliable benchmark to compare the growth trajectories of variants of the package that result from different patterns of technical debt accumulation by various package adopters. The base version provided by the vendor attempts to satisfy the common requirements of the entire customer population of the package. Furthermore, enterprise software vendors typically provide a predictable roadmap[[3]](#footnote-3) 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 [[47](#_ENREF_47), [48](#_ENREF_48)]. As shown in Table 2, we consider two important categories of package variants: higher-debt and lower-debt versions. On one hand, when customers take design shortcuts and modify source code to rapidly add functionality they incur technical debt. The growth of functionality in such higher-debt package variants exceeds that of the benchmark base package provided by the vendor. On the other hand, when customers not only do not add or change vendor-provided functionality, but only adopt a subset of the base package, they prefer to be late adopters, and the resulting lower-debt package variants have a functionality growth that is lower than that of the benchmark. We next elaborate these higher-debt and lower-debt scenarios.

TABLE 2  
Software Package Variants

## 2.2 Higher-Debt Scenario in Enterprise Software Packages

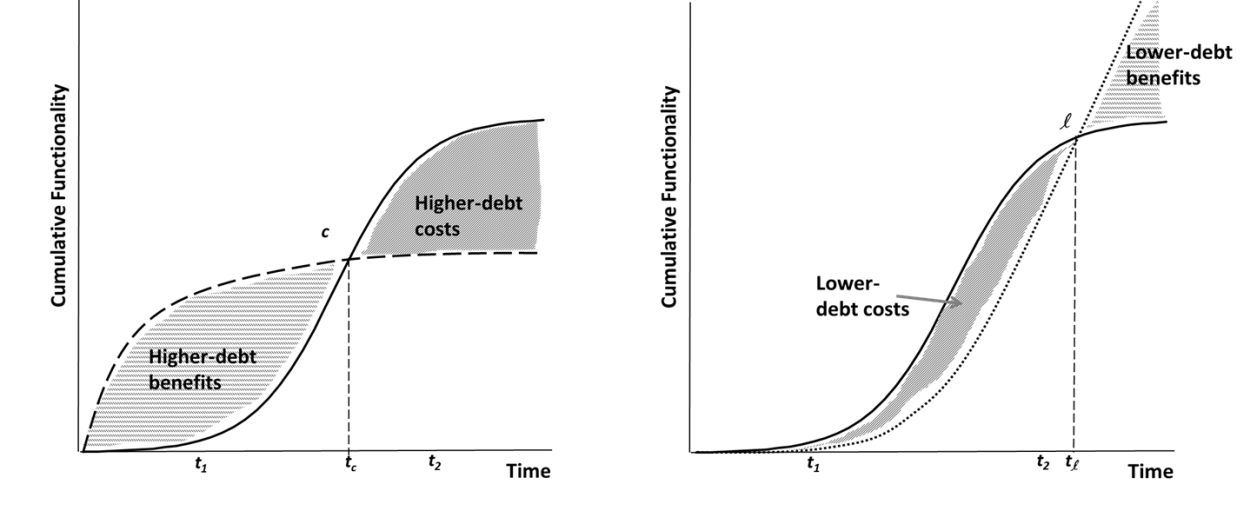


Fig. 3. Expected Growth in Lower-Debt Scenario

Fig. 2. Expected Growth in Higher-Debt Scenario

A higher-debt scenario occurs when a software package customer adds functionality more rapidly than the general customer population of the base package, often by taking design shortcuts. Such customers incur technical debt because they tradeoff the long-term benefits of a likely better technical design and adherence to standards for short-term business benefits realized through the rapid addition of new functionality [[2](#_ENREF_2), [5](#_ENREF_5), [17](#_ENREF_17)]. Seen from an adoption point-of-view, customers who create higher-debt variants can be viewed as early adopters of business functionality that is still not available in the base package in the market.

While the rapid addition of business functionality in higher-debt variants can be expected to yield business benefits, it must be noted that these benefits often come at the expense of investments in software design. Such functionality tends to accentuate code decay, causing performance bottlenecks and creating software code that is more difficult to maintain [[49](#_ENREF_49)]. Thus, in higher-debt variants it can be expected that, after the initial early takeoff, the rate of functionality growth will decrease rapidly over time; higher-debt package variants can be expected to have an earlier saturation point than the base version of the package, all else being equal [[17](#_ENREF_17)].

The early takeoff and saturation in functionality of higher-debt variants can be conceptually modeled using a variety of mathematical distribution functions, such as the logistic and exponential saturation function. Formally, the exponential saturation function can be modeled as F= K (1-e-yt), where *F* is the functionality added to the package, *K* is the maximum value of the functionality at saturation, *t* is the time variable, and *y* is the rate of functionality growth coefficient. An illustration of the higher-debt growth as an exponential saturation curve is shown in Figure 2.

Figure 2 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 *tc* in the lifespan of the implementations. The area between the curves before *tc* reflects the extent of benefits that a higher-debt regime offers, which comes at a cost reflected by the area between the curves after *tc*. Thus, this model clearly reflects the underlying optimization problem: customers who reap the benefits of early functionality that comes at the expense of software design must pay back these costs over the longer term [[1](#_ENREF_1), [2](#_ENREF_2), [5](#_ENREF_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.

## 2.3 Lower-Debt Scenario in Enterprise Software Packages

A lower-debt scenario occurs when customers are late adopters of functionality available in a software package. Such 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 in the lower-debt variants. Thus, the takeoff of lower-debt variant growth is typically delayed as compared to the base version’s S-curve growth, as shown in Figure 3. Potential benefits of lower-debt variants, 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 is a form of opportunity cost for the organization adopting the lower-debt approach. Such lack of business functionality in lower-debt package variants could lead to lowered levels of short-term return on investment for customers. However, as the software package evolves, lower-debt variants can be expected to saturate more slowly relative to the base version because the structural complexity of lower-debt variants would be lower than that of the base version (as well as the higher-debt variants), and thus the rate of code decay in lower-debt variants would be correspondingly lower than that of the base version and higher-debt versions [[17](#_ENREF_17), [49](#_ENREF_49), [50](#_ENREF_50)]. Lower-debt variants can be expected to pay off during the final stages of the package’s lifecycle, providing an opportunity for customers to leverage the relatively lower levels of structural complexity to rapidly add functionality, and thereby derive higher business value.

The growth of a lower-debt variant can be modeled using an exponential growth curve (as well as other distributions such as the logistic curve), which satisfies our conceptualization of delayed takeoff and slower saturation throughout the package’s lifespan in comparison to the base version’s S-curve growth, and the higher-debt variant’s exponential saturation curve. Formally, the growth of a lower-debt variant represented as an exponential function is given by F= eyt, where *F* is the functionality added to the package, *t* is the time variable, and *y* is the rate of functionality growth coefficient.

Referring again to Figure 3, the growth trajectories of the lower-debt variant and the base package intersect at l, corresponding to time *tl* . The area between the curves before *tl* reflects the extent to which the lower-debt regime has lost business functionality, which can be seen as the 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 Figure 3 as the area between the curves after *tl*. Thus, our 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](#_ENREF_13), [17](#_ENREF_17)]. The overall 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.

## Assessing the Business Value of Incurring or Avoiding Technical Debt

As explained above, the evolution of different variants of a software package under the higher and lower-debt modes vary significantly. The extent of functionality gained or lost for different package variants under the higher-debt and lower-debt regimes can be calculated as the area between the functionality growth curves, as illustrated in Figures 2 and 3. To map the functionality gain or loss in a package variant to business value we consider two performance metrics: customer satisfaction and software quality. We chose these performance metrics because they map directly to the short term benefits and long term costs of technical debt, and the measurement and ecological validity of these constructs have been well established in prior literature, including in the context of enterprise software [[51](#_ENREF_51), [52](#_ENREF_52), [53](#_ENREF_53), [54](#_ENREF_54)].

In order to systematically examine the business value of lower-debt and higher-debt variants we consider three important milestones that demarcate a package’s evolutionary stages, and assess the state of customer satisfaction and software quality of the package variants at each of these three milestones. The first important milestone is the *takeoff* of a package, defined earlier, which is a tangible indicator that the growing demand for the package is fueled by the needs of a large mainstream customer population beyond a small set of early adopters. Diametrically opposite to takeoff is the *saturation* *point*, which refers to the point in time when functionality growth tends to stop—a point that clearly demarcates the growth and saturation stages of the package. The milestone that occurs in the period between the takeoff and saturation points (i.e., during the active growth phase) is the point of diminishing returns or *inflection* *point*, which separates the functionality growth curve exactly into two equal regions of opposite concavity. Since we conceptualized the cumulative functionality growth of a base package as a logistic function (S-curve), the inflection point milestone occurs at a time when the functionality added is *K*/2, where *K* is the total functionality added in a package’s entire lifespan.

Next we describe the empirical setting in which we collected data for this research. Following this we present our methodology to segment the different customer installations in our dataset into the base, higher-debt, and lower-debt categories. Then we develop specific hypothesis on the level of customer satisfaction and software quality in the base, higher-debt, and lower-debt customer installations at different points in the evolution of the software package. For ease of readability we present hypotheses and corresponding results in sequence, grouped according to the data involved for the hypothesis testing.

# 3 Empirical Setting and Data Collection

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 data from the product management division of the firm on one of its recently retired enterprise software packages that was originally launched in 2002. The package was sold as modifiable off-the-shelf software to customers (base version). The base version could be altered in a variety of ways (i.e., customized) to suit the specific needs of a customer. For example, an aggressively growth-oriented customer could add features at a faster pace than the firm could fulfill using its standard releases. As is typical in such cases, the firm made no guarantees to preserve such custom-modified or added code during its regular updates to the base package. Thus, the risk of customization and any related design shortcuts was fully the responsibility of the customers. In contrast to these customers, a different customer may not utilize all the features offered and had the opportunity to install only a subset of the available features[[4]](#footnote-4). Thus, depending on the extent of customization and utilization, the evolution of a package variant at a customer site could be significantly different from the base package over its entire lifecycle. Such variations in growth trajectories of installed software packages at customer sites provided an excellent research context for testing our model of technical debt accumulation patterns. Their longitudinal nature allows for testing of various hypotheses of expected behavior over time, and the single source holds many variables constant, allowing for greater focus on the elements that do change that are the focus of this research.

We collected monthly functionality release data from the product management division of the firm for the entire 10 year lifecycle of the software package. The firm characterized functionality in units termed “transactions”. In the context of the firm’s software package a *transaction* 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, transactions capture the customer-focused functionality offered by the software package [[55](#_ENREF_55)]. 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 [[56](#_ENREF_56), [57](#_ENREF_57)]. 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. We utilized the cumulative number of transactions available in the package to model its growth over its ten year lifespan. This characterization of functionality growth as the independent variable in the model is attractive to our research, as it was not developed or measured by the research team, thereby reducing the opportunity for researcher measurement bias in testing the research model.

We had access to data from 75 customers[[5]](#footnote-5) (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 the CSN system. Software developers at the research site could also remotely assist customers in software installation and other maintenance activities through the CSN system.

We were able to obtain longitudinal and contemporaneous customer satisfaction and quality data via a feature of the CSN system. 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. Similarly, quality was measured using three measures: 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. Taken together, these customer satisfaction and quality metrics map directly to the performance measures outlined in Kruchten et al.’s recently proposed technical debt framework, spanning the entire internal-external spectrum [[3](#_ENREF_3)].

# Analysis

## Categorizing Base, Lower-Debt, and Higher-Debt Variants

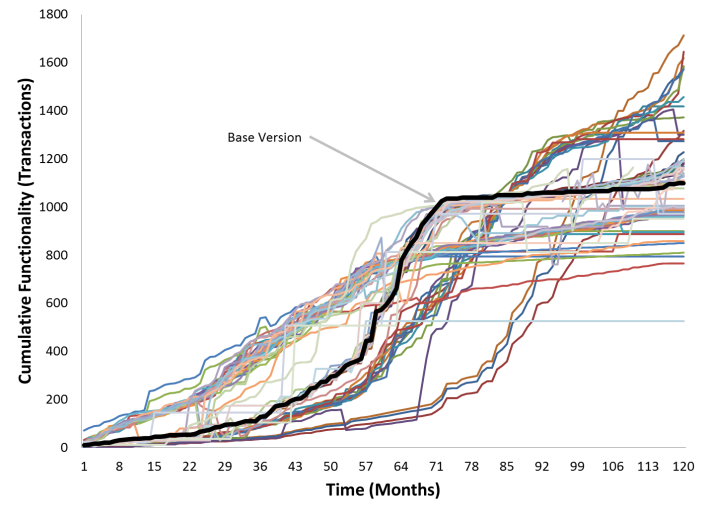


Fig. 5. Functionality Growth at Customer Installations

As a first step we analyzed the functionality growth of the base package to determine the three important milestones we identified in section 2.4. 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](#_ENREF_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 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 1100 transactions. Recall that the inflection point of a logistic growth 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.

Figure 4 visually presents the three different phases of the actual evolution of the base package, demarcated by the three major milestones. As seen there the actual data for the base version’s growth clearly follows the S-curve pattern we conceptualized, and adheres to the empirical generalization of adoption patterns established in the diffusion of innovations literature [[11](#_ENREF_11), [41](#_ENREF_41), [45](#_ENREF_45)]. This result supports the overall validity and relevance of our dataset to investigate the different patterns of growth by holding the growth curve of the base package as a theoretically grounded and empirically reliable benchmark in our models.

In the second step we analyzed the functionality growth of packages at the 75 customer installations. We needed to exclude 6 customer installations due to missing data, and therefore examined the growth patterns of the remaining 69 data samples in our analysis. These patterns relative to the growth of the base version are visually shown in Figure 5. As the figure shows, functionality growth at the different customer installations varies, with some installations closely following the base version, but other installations deviating significantly from it. We note that the rate of growth at some customer installations is systematically above or below the base version at the three phases of the package’s evolution (before takeoff, growth phase, and saturation phase), but a few other customer installations depict significant transitions, crossing the base version curve one or more times during the package’s lifespan.

In order to assess the business value of differing approaches to technical debt at these commercial customer installations, we iteratively grouped the 69 data samples into four categories using the following procedure:

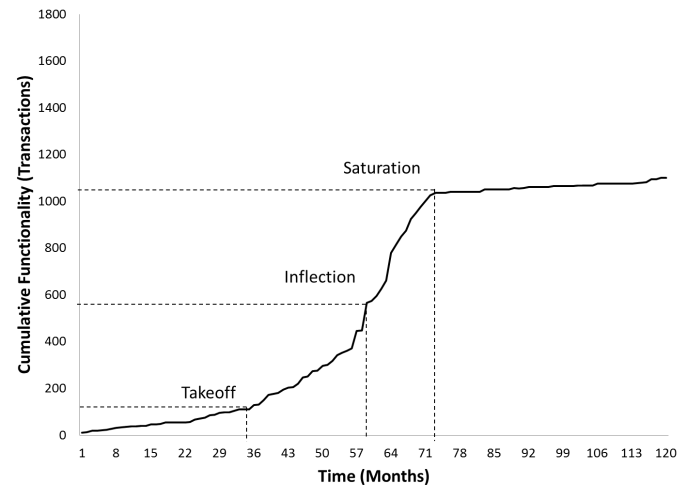


Fig. 4. Base Package Growth

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 “higher-debt candidate”, else
4. If the functionality difference is uniformly lower than zero up to the inflection point, label as “lower-debt 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 higher-debt variant occurred after the inflection point of the base package, and that the crossover point of the least conservative lower-debt 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 higher or lower-debt strategies from our analysis would have conceptual validity.

This sorting resulted in 48 of the installations being clearly categorized as one of: base, higher-debt, or lower-debt. The growth trajectories of the base package group is shown in Figure 6, and that of the higher-debt and lower-debt package variants relative to the base version are shown in Figures 7 and 8. As the figures show, the base package group traces the S-curve growth of the vendor supplied base package very closely, higher-debt variants follow what appears to be an exponential saturation pattern, and lower-debt variants follow what appears to be an exponential growth pattern. These results are consistent with the evolutionary model of technical debt accumulation patterns we proposed in section 2, and provide an appropriate characterization of the customer installations for testing our hypotheses related to the business value of the different package variants. 21 customer installations that were identified as “transitional candidates”, i.e., not one of the first three canonical cases, were set aside to be examined later in the analysis (see Section 4.3).

## Business Value of Higher-Debt and Lower-Debt Variants

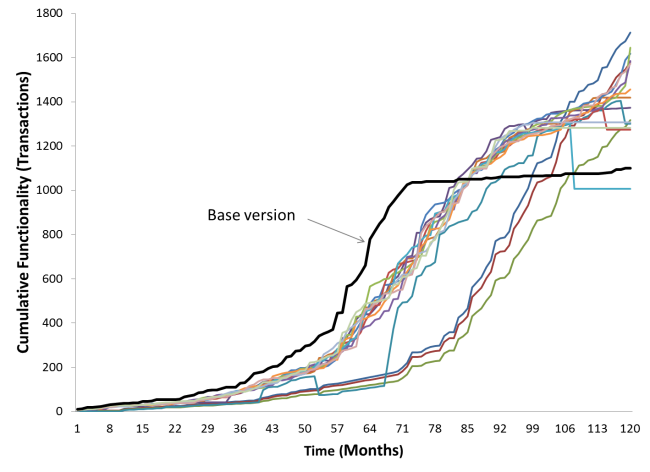


Fig. 8. Lower-debt Group (N=15)



Fig. 9. Sample Variance Distribution

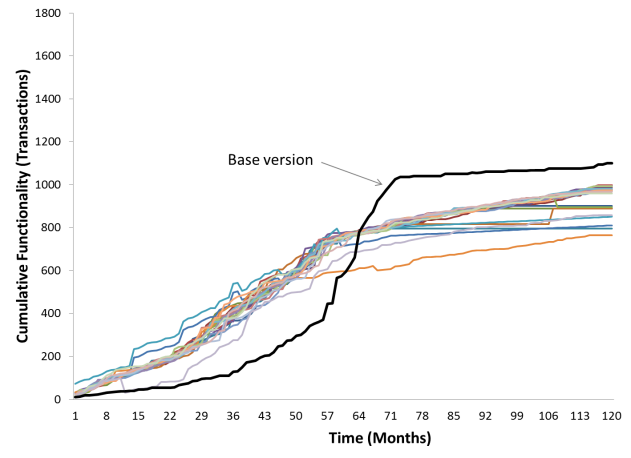


Fig. 7. Higher-debt Group (N=22)

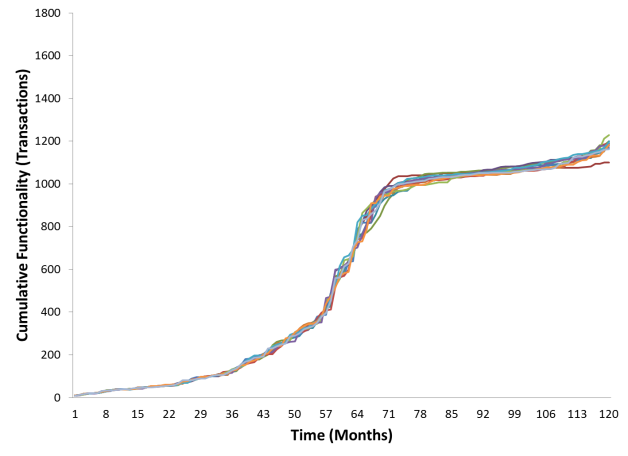


Fig. 6. Base Package Group (N= 11)

Note: The vendor’s base version growth curve is omitted in Figure 7 to insure visibility of the eleven cases in the base package group.

Once we validated the overall distributional properties of the growth trajectories of the base, higher-debt and lower-debt variants, we proceeded to develop and test specific hypotheses on the business value of the higher-debt and lower-debt variants, using both software quality and customer satisfaction as performance metrics. Table 3 presents the mean-level data aggregated across the base, higher-debt and lower-debt package categories at the three milestones in their evolution. We analyzed the variations within each of the package categories at different time periods and observed that the variations followed a normal distribution pattern. One such example is shown for illustrative purposes in Figure 9, which shows the distribution of the functionality within variants in the higher-debt category before takeoff at month 29[[6]](#footnote-6).

**4.2.1. Technical Debt and Software Quality**

Our first set of hypotheses focuses on the state of software quality in the higher-debt and lower-debt variants of a software package. Prior studies on technical debt have shown a negative correlation between higher levels of technical debt and software quality [[27](#_ENREF_27), [29](#_ENREF_29), [40](#_ENREF_40)]. Seeking to test for the presence of such a pattern in our dataset, we propose the following confirmatory hypotheses that compare the quality levels of higher-debt and lower-debt package variants with the base version of the package.

**Hypothesis 1a.** *Throughout the evolution of the software package software quality of higher-debt variants will be lower than that of the base package.*

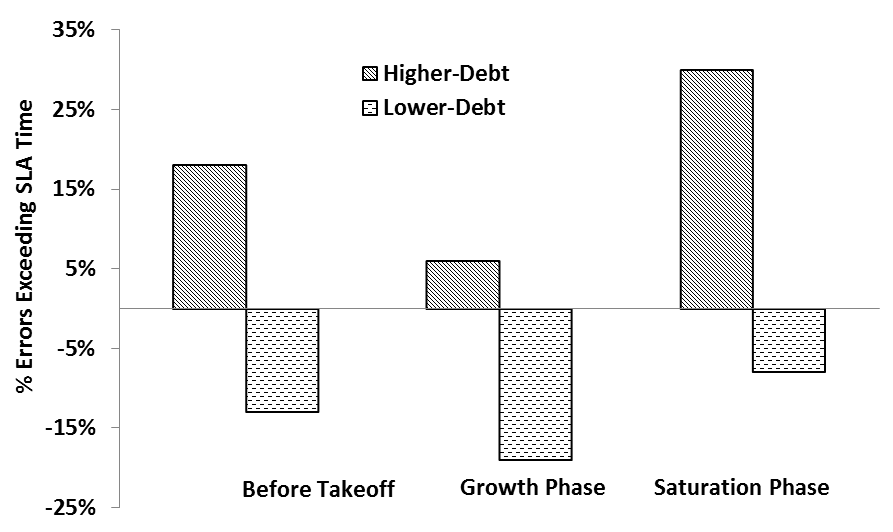


Fig. 11. Percentage Errors Exceeding SLA

**Hypothesis 1b.** *Throughout the evolution of the software package the software quality of lower-debt variants will be higher than that of the base package.*

TABLE 3  
Mean Performance Metrics Across Package Variants

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

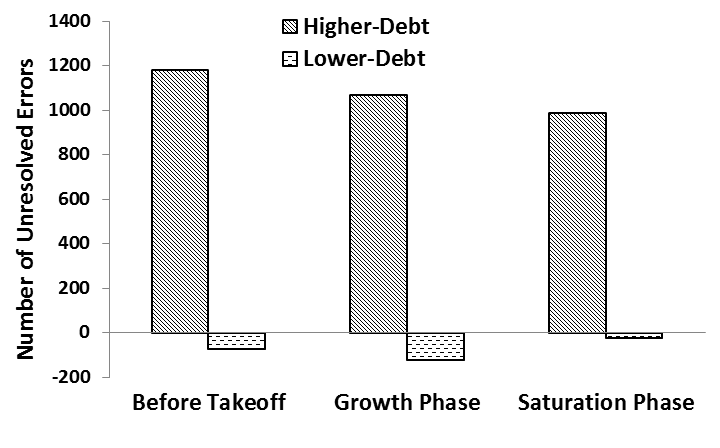


Fig. 10. Unresolved Errors

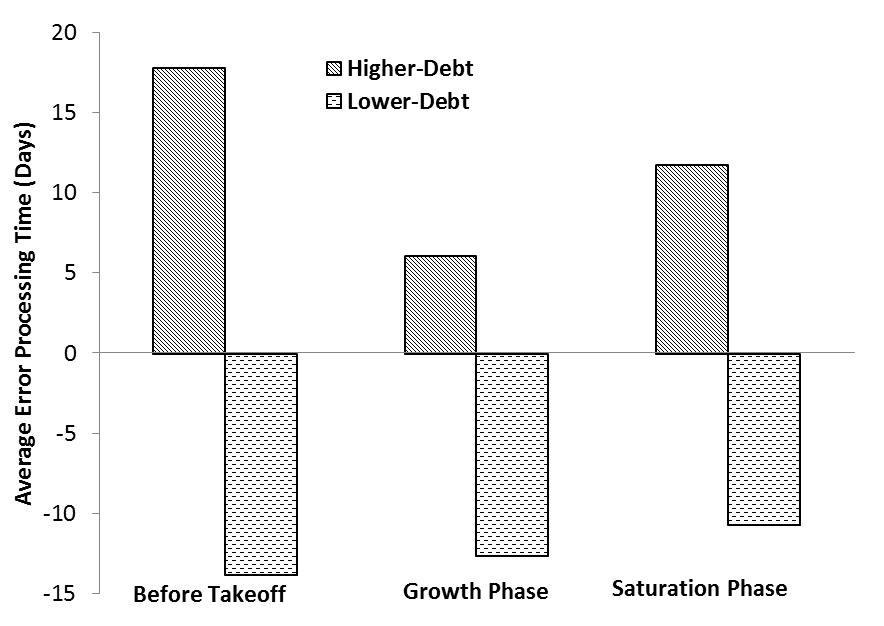


Fig. 12. Average Error Processing Time

To test our first set of hypotheses we compared the number of unresolved errors, average processing time of errors, and the percentage of errors exceeding the contractually agreed values found in the service level agreements (SLA) in the three phases of the package evolution. Referring to Table 3, we can see that the mean levels of all these metrics for higher-debt variants are higher than that of the base package, and the values for lower-debt variants are lower than that of the base package. Using t-tests of the comparison of means across the different groups we verified that these differences were statistically significant at p-values<5%. These quality differences between the higher-debt and lower-debt variants relative to that of the base package are illustrated in Figures 10, 11, and 12.

**4.2.2. Technical Debt and Customer Satisfaction**

We posit that the relationship between customer satisfaction and technical debt depends on the extent of technical debt accumulation, as well as the lifecycle stage of a software package. In the early stages of a package’s evolution, savvy early adopters tend to look for novel business functionality, and a rapid growth in functionality is likely to be relatively more satisfying [[41](#_ENREF_41), [52](#_ENREF_52)]. Hence, we expect that, all else being equal, the customer satisfaction of higher-debt package variants will be greater than that of the base package and lower-debt variants in this early stage. Since administrators of lower-debt 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](#_ENREF_17)]. Thus, in the evolutionary stage before takeoff, we expect that the customer satisfaction of lower-debt package variants will be lower than the base package provided by the vendor, all else being equal. Hence our second set of hypotheses is:

**Hypothesis 2a.** *In the phase before takeoff the customer satisfaction of higher-debt variants will be higher than that of the base package.*

**Hypothesis 2b.** *In the phase before takeoff the customer satisfaction of lower-debt variants will be lower than that of the base package.*

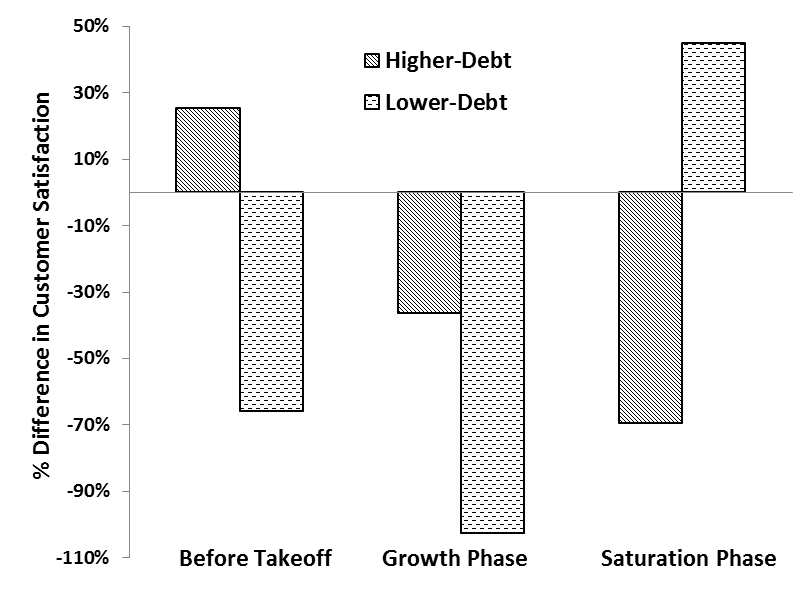


Fig. 13. Customer Satisfaction

The *growth phase* of evolution of a base package is defined as the time period between the takeoff and saturation points, which spans the inflection point of the cumulative functionality growth (refer to Figures 1 and 4). During this period the lower-debt package variants still lag in functionality (see Figure 8). However, the higher-debt package variants will have started to saturate, and the growth in functionality will be tending towards zero (Figure 7). In contrast to the higher-debt and lower-debt variants, functionality of the base package experiences rapid growth (Figure 6). Thus, we expect that during the growth phase of the base package, customer satisfaction of the base package will be higher than that of both the higher-debt and lower-debt variants, which leads to our third set of hypotheses:

**Hypothesis 3a.** *In the growth phase the customer satisfaction of the base package will be higher than that of the higher-debt variants.*

**Hypothesis 3b.** *In the growth phase the customer satisfaction of the base package will be higher than that of the lower-debt variants.*

At a time when the growth of the base version of a package saturates, the total functionality built in the package exceeds that of the higher-debt variant (Figure 7). However, lower-debt variants are now poised for their growth phase as clients wanting to extract benefits out of their package during the last stage of the technology lifespan are more likely to begin to experiment and add functionality [[42](#_ENREF_42)]. Because of the relatively increased levels of structural complexity present in the higher-debt and base packages, such late-stage experimentation is less likely [[17](#_ENREF_17)]. Thus, during the last stages of a product’s lifespan we expect that the customer satisfaction of higher-debt variants would be lower than that of the base package, and the customer satisfaction of lower-debt variants would be higher than that of the base package. This leads to our next set of hypotheses:

**Hypothesis 4a.** *In the saturation phase the customer satisfaction of higher-debt variants will be lower than that of the base package.*

**Hypothesis 4b.** *In the saturation phase the customer satisfaction of lower-debt variants will be higher than that of the base package.*

To test hypotheses 2a through 4b we performed a series of t-tests comparing the customer satisfaction of higher-debt, lower-debt, and base packages at the three stages of package evolution (before takeoff phase, growth phase, and saturation phase). The t-tests confirmed our hypotheses at a high level of statistical significance (p-value<1%). The mean differences in customer satisfaction between the variants are shown in Table 3 and illustrated in Figure 13. Our results show that before the takeoff of the base package customer satisfaction with the higher-debt variants averaged 25% higher than the base package. The customer satisfaction of the lower-debt variants during this stage is about 66% lower than that of the base package. During the growth phase the customer satisfaction of the base package is at its peak, and exceeds that of the lower-debt variants by more than 100% and that of the higher-debt variants by about 36%. In the final stages of the package’s evolution the lower-debt products depict the highest levels of customer satisfaction, which is about 44% higher than that of the base package. In contrast, the higher-debt variants are associated with the lowest customer satisfaction levels, which are about 70% lower than that of the base package. These results are consistent with all of the hypotheses pairs.

**4.2.3. Results Summary**

Our results on the software quality and customer satisfaction of the base, higher-debt, and lower-debt variants collectively illustrate the tradeoffs between the benefits and costs of the accumulating technical debt over the lifespan of a software package. On one hand, higher-debt variants are associated with increased customer satisfaction levels initially, but suffer from quality issues and saturate much earlier in the lifecycle. On the other hand, lower-debt variants are associated with initial poor customer satisfaction, but show rapid growth and improved customer satisfaction in the later stages of evolution when the base package and higher-debt variants suffer from lowered levels of customer satisfaction and quality. These results are consistent with what much of the technical debt literature has suggested with respect to costs, and also provide support for the notion of the business benefits of specific technical debt strategies.

## Package Variants in Transition

We now proceed to examine the 21 “transitional” candidate cases, which refer to situations where customers alter their debt-taking behavior at different stages of a software package’s evolution and therefore could not be appropriately examined as “pure” cases of high or low technical debt strategy. In such situations a package variant transitions from one of the canonical patterns (higher-debt, lower-debt, or base package) 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 7 cases with multiple transitions in their lifespan.

Any significant change in the debt-taking or debt-avoiding behavior of a customer alters the package growth trajectory in significant ways. For example, a customer owning a higher-debt package variant could invest in refactoring activities to stem code decay. This, in turn, decreases the rate of performance saturation, and tends to push the package variant’s growth towards that of the base version. We observed 3 cases in our data set where such a transition from higher-debt to base package behavior occurred before the takeoff milestone (Figure 14), and 4 cases with such a transition after the takeoff milestone (Figure 15).

A lower-debt 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 2 cases where lower-debt variants transitioned to the base package group before takeoff (Figure 16). In 5 other cases, package variants in the base group moved all the way to the higher-debt variant category before takeoff occurred (Figure 17). Apart from these one-time transitions, we observed 7 cases where customers transitioned across the base package growth trajectory multiple times in the 120 month lifespan of the package (Figure 18).

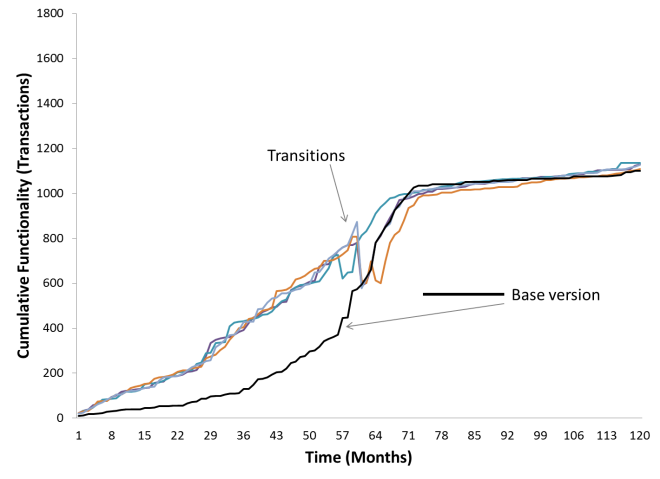


Fig. 15. Transitioning from Higher-debt to Base Package after Takeoff (N=4)

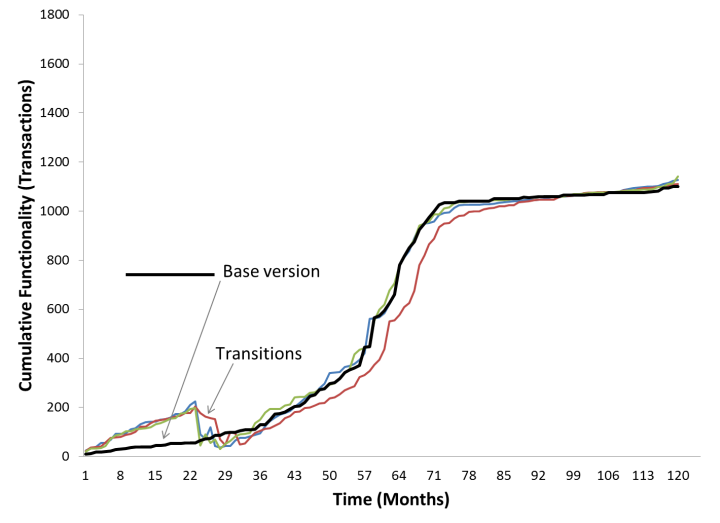


Fig. 14. Transitioning from Higher-debt to Base Package before Takeoff (N=3)

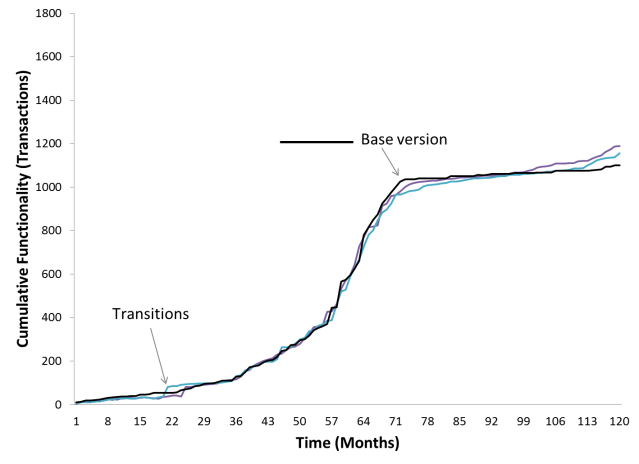


Fig. 16. Transitioning from Lower-debt to Base Package before Takeoff (N=2)

By considering a transition as the unit of analysis, we examined the impact of a shift in debt-taking 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 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 [[58](#_ENREF_58)]. For example, a transition from higher-debt towards the base package might alienate business customers who preferred the rapid expansion of functionality. Similarly, a transition from lower-debt 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 [[51](#_ENREF_51), [52](#_ENREF_52)]. Thus, we expect a drop in customer satisfaction when a transition happens in the growth trajectory of package variants that have taken off. Because transitions after takeoff deviate from customer expectations that lead to their initial adoption (and subsequent package takeoff), we posit that the drop in customer satisfaction is more pronounced for transitions after takeoff as compared to the transitions before takeoff. This leads to our next hypothesis:

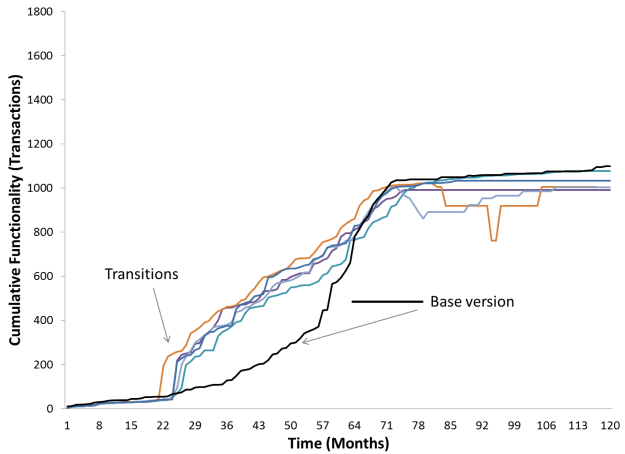


Fig. 17. Transitioning from Base Package to Higher-debt before Takeoff (N=5)

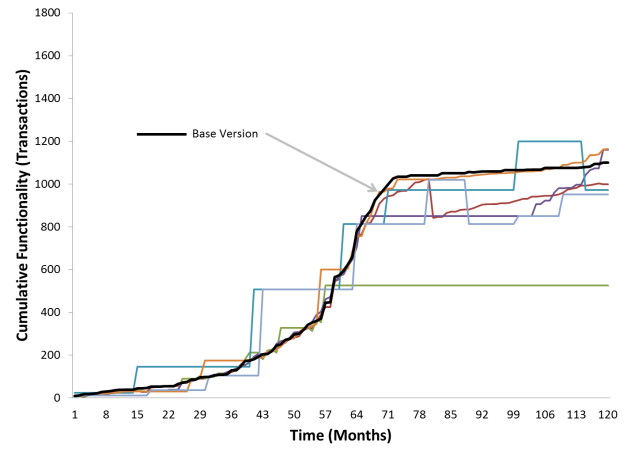


Fig. 18. Multiple Transitions (N=7)

**Hypothesis 5a.** *For transitions after takeoff, customer satisfaction decreases to a larger extent as compared to transitions before takeoff.*

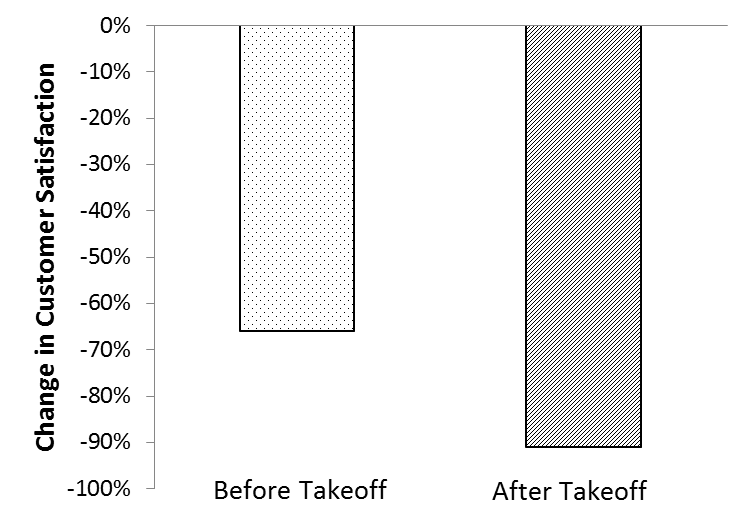


Fig. 19. Transitions and Change in Customer Satisfaction

To test hypothesis 5a we calculated the difference in customer satisfaction levels between the package groups experiencing transition before and after the takeoff milestone. Our results indicate that customer satisfaction levels dropped by more than 91% for transitions after takeoff, as compared to a decrease of 66% for transitions before takeoff. This is illustrated in Figure 19. T-tests showed that the 25% difference in the drop of customer satisfaction levels between transitions before and after takeoff was statistically significant at a p-value<1%, thus supporting H5a. 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 timing debt-reducing design investments judiciously, which we discuss further in Section 5.

Debt-reducing and debt-increasing transitions have opposing impacts on the business functionality and software quality of a package. Debt-reducing transitions tend to improve software quality, often at the expense of business functionality [[17](#_ENREF_17), [29](#_ENREF_29), [33](#_ENREF_33)]. In contrast, debt-increasing transitions tend to increase business functionality at the expense of software quality [[17](#_ENREF_17), [27](#_ENREF_27), [36](#_ENREF_36)]. These contrasting effects of debt-reducing and debt-increasing transitions could have different impacts on customer satisfaction beyond the timing of these transitions considered in hypothesis 5a. In order to empirically test the relative effects of debt-reducing and debt-increasing transitions on customer satisfaction, we propose the following hypothesis:

**Hypothesis 5b.** *Customer satisfaction of debt-reducing transitions is higher than customer satisfaction of debt-increasing transitions.*

To test hypothesis 5b we performed a t-test of means, comparing the customer satisfaction levels between the debt-reducing and debt-increasing transitions with an *n* of 14. Our results indicated that there was no statistically significant difference at the p<.05 level between the levels of customer satisfaction between the two groups of transitions. Even among the cases with multiple transitions (*n*=7) we did not notice significant difference in customer satisfaction levels between the debt-reducing and debt-increasing transitions. Thus, we did not find support for hypothesis 5b. We acknowledge the need to further investigate this result using a broader sample of transitional cases in future research, as the lack of statistical significance may be due to the very small sample size available.

## Summary of Analysis and Results

To summarize our empirical analysis and results we first utilized data from the ten year lifecycle of a large enterprise software package and 69 different implementations of the package to validate our evolutionary model of technical debt accumulation; our analysis confirmed that the base package supplied by the vendor followed a logistic functionality growth pattern, the lower-debt package variants followed an exponential growth pattern, and the higher-debt package variants followed an exponential saturation pattern of evolution. We then tested specific hypotheses on the business value derived by customers using the higher-debt and lower-debt package variants relative to the base package supplied by the vendor. We found strong support for nine of the ten hypotheses, which are summarized in Table 4. Overall, these results document the tradeoffs between the benefits and costs of the accumulating technical debt over the lifespan of a software package, and call for judicious management of technical debt at different phases of a package’s evolution. In the next section we discuss the opportunities for potential practical design decisions at important milestones in the evolution of an enterprise software package, along with some research implications of our results.

# Discussion

## Managerial Implications

Our empirical results have established the underlying tradeoffs between the costs and benefits of accumulating technical debt at different stages of a package’s evolution. The evolutionary model we developed and validated highlights several opportunities for design investments and transitions in functionality growth trajectories of the package variants throughout their lifespan. Based on our model and results, we highlight six key design decision points throughout the evolution of a software package— three on the higher-debt trajectory (HD1-3 in Figure 20) and three on the lower-debt trajectory (LD1-3 in Figure 20).

TABLE 4  
Results Summary

|  |  |  |
| --- | --- | --- |
| **No. (section)** | **Hypothesis** | **Result** |
| H1a (4.2.1) | Software quality of higher-debt variants < software quality of base package throughout the lifespan | Supported.  Higher-debt variants on average have 10xmore unresolved errors, 3x more errors that exceed the contractually agreed upon resolution times, and take 2x the time to be resolved |
| H1b (4.2.1) | Software quality of higher-debt variants < software quality of base package throughout the lifespan | Supported.  Lower-debt variants on average have 6x fewer unresolved errors, 4x lower errors that exceed the contractually agreed upon resolution times, and are resolved 5x more quickly. |
| H2a (4.2.2) | Customer satisfaction of higher-debt variants > customer satisfaction of base package in the phase before takeoff. | Supported.  Customer satisfaction of higher-debt variants higher by 25%. |
| H2b (4.2.2) | Customer satisfaction of lower-debt variants < customer satisfaction of base package in the phase before takeoff.. | Supported  Customer satisfaction of lower-debt variants lower by 66%. |
| H3a (4.2.2) | Customer satisfaction of higher-debt variants < customer satisfaction of base package in the growth phase. | Supported.  Customer satisfaction of higher-debt variants lower by 36%. |
| H3b (4.2.2) | Customer satisfaction of lower-debt variants < customer satisfaction of base package in the growth phase. | Supported.  Customer satisfaction of lower-debt variants lower by 102%. |
| H4a (4.2.2) | Customer satisfaction of higher-debt variants < customer satisfaction of base package in the saturation phase. | Supported.  Customer satisfaction of high-dent variants lower by 69%. |
| H4b (4.2.2) | Customer satisfaction of lower-debt variants > customer satisfaction of base package in the saturation phase. | Supported.  Customer satisfaction of lower-debt variants greater by 45%. |
| H5a (4.3) | Customer satisfaction for transitions after takeoff < customer satisfaction of transitions before takeoff. | Supported.  Customer satisfaction of transitions after takeoff lower by 25%. |
| H5b (4.3) | Customer satisfaction of debt-reducing transitions > customer satisfaction of debt-increasing transitions. | Not Supported.  No statistically significant difference at the p<=.05 level. |

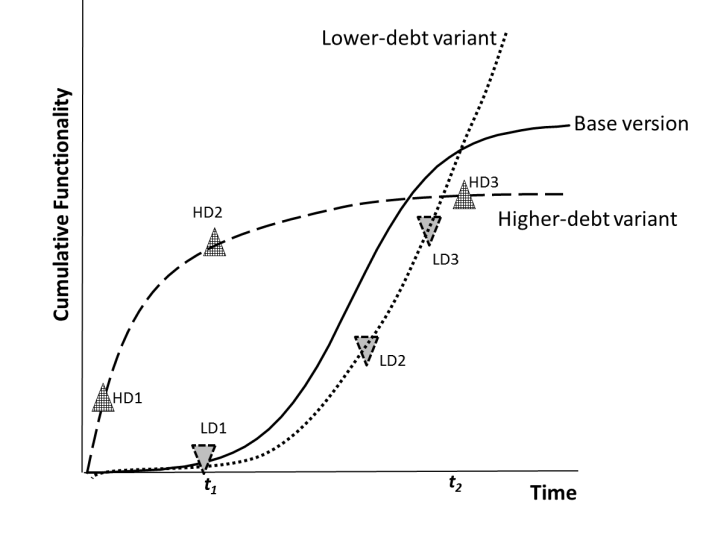


Fig. 20. Package Variants and Design Decisions

The first design 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 *t1*). 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 *t1*), 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.

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 design investment targeted at reducing the accumulated technical debt 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 firm 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 design 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 the 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 4.1 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 [[47](#_ENREF_47), [48](#_ENREF_48)]. The roadmap announcement helps the vendor to plan support and development activities. We recommend that customers invest in debt-reducing or functionality-increasing design 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 design 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 design 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 design 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.

## Research Contributions

In this study we developed and validated an evolutionary model of technical debt accumulation in a commercial enterprise software package that explicitly measures the impact of tradeoffs between functionality growth and software quality and customer satisfaction at different stages of the package’s multi-year lifespan.

This study makes several key contributions. First, the technical debt accumulation model advanced in this study adopts an evolutionary perspective, facilitating the analysis of costs and benefits 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 longitudinal 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 dataset used here. Second, building on recent efforts to address technical debt in software ecosystems [[23](#_ENREF_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. Third, 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 be expanded with a larger sample size. Finally, future research could compare and contrast the technical debt accumulation patterns in a wider range of types of software packages, which would aid the broader generalization of the results reported in this paper.

**Acknowledgment**

We thank J. Woodard, S. Daniels, N. Nagappan, 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 insightful comments on an earlier version of the paper.

**References**

[1] W. Cunningham, “The Wycash Portfolio Management System,” in Proceedings of Object-Oriented Programming Systems, Languages, and Applications, 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 Proceedings of FSE/SDP Workshop on Future of Software Engineering Research, Santa Fe, New Mexico, USA, 2010, pp. 47-52.

[3] 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, 2012.

[4] A. MacCormack, R. Verganti, and M. Iansiti, “Developing Product on "Internet Time": The Anatomy of a Flexible Development Process,” *Management Science,* vol. 47, no. 1, pp. 133-150, 2001.

[5] C. Seaman, and Y. Guo, "Measuring and Monitoring Technical Debt," *Advances in Computers*, M. V. Zelkowitz, ed., pp. 25-46, London, UK: Academic Press, 2011.

[6] 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, 2012.

[7] C. F. Kemerer, and S. Slaughter, “An Empirical Approach to Studying Software Evolution,” *IEEE Transactions on Software Engineering,* vol. 25, no. 4, pp. 493-509, 1999.

[8] D. Harter, C. F. Kemerer, and S. Slaughter, “Does Software Process Improvement Reduce the Severity of Defects? A Longitudinal Field Study,” *IEEE Transactions on Software Engineering,* vol. 38, no. 4, pp. 810-827, 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 Proceedings of Seventh Working IEEE/IFIP Conference on Software Architecture, Vancouver, Canada, 2008, pp. 83-92.

[10] N. Ramasubbu, and C. F. Kemerer, “Towards a Model for Optimizing Technical Debt in Software Products,” in Proceedings of Fourth International Workshop on Managing Technical Debt, San Francisco, CA, 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 Science,* 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 Science,* vol. 16, no. 3, pp. 256-270, 1997.

[13] M. Fowler. "Technical Debt," accessed on January 18, 2013; <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 Third International Workshop on Managing Technical Debt, Zurich, Switzerland, 2012.

[15] T. Klinger, P. Tarr, P. Wagstrom, and C. Williams, “An Enterprise Perspective on Technical Debt,” in Second International Workshop on Managing Technical Debt, Waikiki, Honolulu, Hawaii, USA, 2011.

[16] T. Theodoropoulos, M. Hofberg, and D. Kern, “Technical Debt from the Stakeholder Perspective,” in Second International Workshop on Managing Technical Debt, Waikiki, Honolulu, Hawaii, USA, 2011.

[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 Third International Workshop on Managing Technical Debt, Zurich, Switzerland, 2012.

[19] S. McConnell, “Managing Technical Debt,” in 4th International Workshop on Technical Debt, Keynote Presentation, San Francisco, CA, 2013.

[20] C. Baldwin, and A. MacCormack, "Finding Technical Debt in Platform and Network Architectures," Presented at The Mathworks, 2011.

[21] J. Brondum, and L. Zhu, “Visualising Architectural Dependencies,” in Third International Workshop on Managing Technical Debt, Zurich, Switzerland, 2012.

[22] J. D. Morgenthaler, M. Gridnev, R. Sauciuc, and S. Bhansali, “Searching for Build Debt,” in Third International Workshop on Managing Technical Debt, Zurich, Switzerland, 2012.

[23] J. D. McGregor, J. Y. Monteith, and J. Zhang, “Technical Debt Aggregation in Ecosystems,” in Third International Workshop on Managing Technical Debt, Zurich, Switzerland, 2012.

[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 Second International Workshop on Managing Technical Debt, Waikiki, Honolulu, Hawaii, USA, 2011.

[25] B. Boehm, “Assessing and Avoiding Technical Debt,” in Third International Workshop on Managing Technical Debt, Zurich, Switzerland, 2012.

[26] J. d. Groot, A. Nugroho, T. Back, and J. Visser, “What is the value of your Software?,” in Third International Workshop on Managing Technical Debt, Zurich, Switzerland, 2012.

[27] F. A. Fontana, V. Ferme, and S. Spinelli, “Investigating the Impact of Code Smells Debt on Quality Code Evaluation,” in Third International Workshop on Managing Technical Debt, Zurich, Switzerland, 2012.

[28] B. Curtis, J. Sappidi, and A. Szynkarski, “Estimating the Principal of an Application's Technical Debt,” *IEEE Software,* vol. 29, no. 6, pp. 34-42, 2012.

[29] N. Zazworka, C. Seaman, F. Shull, and M. Shaw, “Investigating the Impact of Design Debt on Software Quality,” in Second International Workshop on Managing Technical Debt, Waikiki, Honolulu, Hawaii, USA, 2011.

[30] A. Nugroho, J. Visser, and T. Kuipers, “An Empirical Model of Technical Debt and Interest,” in Second International Workshop on Managing Technical Debt, Waikiki, Honolulu, Hawaii, USA, 2011.

[31] J. Heintz, and I. Gat, “Investigating From Assessment to Reduction: Reining in Millions,” in Second International Workshop on Managing Technical Debt, Waikiki, 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,” in Second International Workshop on Managing Technical Debt, Waikiki, Honolulu, Hawaii, USA, 2011.

[33] J. Bohnet, and J. Döllner, “Monitoring Code Quality and Development Activity by Software Maps,” in Second International Workshop on Managing Technical Debt, Waikiki, Honolulu, Hawaii, USA, 2011.

[34] K. J. Sullivan, W. G. Griswold, Y. Cai, and B. Hallen, “The Structure and Value of Modularity in Software Design,” in Proceedings of 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Vienna, Austria, pp. 99-108.

[35] Y. Guo, and C. Seaman, “A Portfolio Approach to Technical Debt Management,” in Second International Workshop on Managing Technical Debt, Waikiki, Honolulu, Hawaii, USA, 2011.

[36] W. Nichols, “A Cost Model and Tool to Support Quality Economic Trade-off Decisions,” in Second International Workshop on Managing Technical Debt, Waikiki, Honolulu, Hawaii, USA, 2011.

[37] N. Zazworka, C. Seaman, F. Shull, and M. Shaw, “Prioritizing Design Debt Investment Opportunities,” in Second International Workshop on Managing Technical Debt, Waikiki, Honolulu, Hawaii, USA, 2011.

[38] J.-L. Letouzey, and M. Ilkiewicz, “Managing Techincal Debt with the SQUALE Method,” *IEEE Software,* vol. 29, no. 6, pp. 44-51, 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 Third International Workshop on Managing Technical Debt, Zurich, Switzerland, 2012.

[40] W. Snipes, and B. Robinson, “Defining the Decision Factors for Managing Defects: A Technical Debt Perspective,” in Third International Workshop on Managing Technical Debt, Zurich, Switzerland, 2012.

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

[42] C. M. Christensen, “Exploring the Limits of the Technology S-Curve: Component Technologies,” *Production and Operations Management,* vol. 1, no. 4, pp. 334-357, 1992.

[43] K. Hartig, "Tuning In: Communications Technologies Historically Have Had Broad Appeal for Consumers," *Wall Street Journal Classroom Edition,* 1998, <http://www.karlhartig.com/chart/techhouse.pdf>, [Accessed on 5/6/2013].

[44] F. Bass, “A New Product Growth Model for Consumer Durables,” *Management Science,* vol. 15, no. 5, pp. 215-227, 1969.

[45] R. Agarwal, and B. L. Bayus, “The Market Evolution and Sales Takeoff of Product Innovations,” *Management Science,* vol. 48, no. 8, pp. 1024-1041, 2002.

[46] 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: Springer, 2000.

[47] I. J. Patrick, and A. E. Echols, “Technology Roadmapping in Review: A Tool for Making Sustainable New Product Development Decisions,” *Technological Forecasting and Social Change,* vol. 71, no. 1-2, pp. 81-100, 2004.

[48] P. Groenveld, “Roadmapping Integrates Business and Technology,” *Research-Technology Management,* vol. 50, no. 6, pp. 49-58, 2007.

[49] 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 Transactions on Software Engineering,* vol. 27, no. 1, pp. 1-12, 2001.

[50] D. P. Darcy, C. F. Kemerer, S. A. Slaughter, and J. E. Tomayko, “The Structural Complexity of Software: An Experimental Test,” *IEEE Transactions on Software Engineering,* vol. 31, no. 11, pp. 982-995, 2005.

[51] 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,” *Decision Support Systems,* vol. 44, no. 2, pp. 509-523, 2008.

[52] S. Kekre, M. S. Krishnan, and K. Srinivasan, “Drivers of Customer Satisfaction for Software Products: Implications for Design and Service Support,” *Management Science,* vol. 41, no. 9, pp. 1456-1470, 1995.

[53] G. Succi, L. Benedicenti, and T. Vernazza, “Analysis of the Effects of Software Reuse on Customer Satisfaction in an RPG Environment,” *IEEE Transactions on Software Engineering,* vol. 27, no. 5, pp. 473-479, 2001.

[54] 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 Transactions on Software Engineering,* vol. 35, no. 4, pp. 534-550, 2009.

[55] C. F. Kemerer, and B. Porter, “Improving the Reliability of Function Point Measurement: An Empirical Study,” *IEEE Transactions on Software Engineering,* vol. 18, no. 10, pp. 1011-1024, 1992.

[56] N. C. Olsen, “Survival of the Fastest: Improving Service Velocity,” *IEEE Software,* vol. 12, no. 5, pp. 28-38.

[57] M. Lindvall, D. Muthig, A. Dagnino, C. Wallin, M. Stupperich, D. Kiefer, J. May, and T. Kähkönen, “Agile Software Development in Large Organizations,” *IEEE Software,* vol. 37, no. 12, pp. 26-34, 2004.

[58] E. Bridges, C. K. B. Yim, and R. A. Briesch, “A High-Tech Product Market Share Model with Customer Expectations,” *Marketing Science,* vol. 14, no. 1, pp. 61-81, 1995.

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]. [↑](#footnote-ref-1)
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 this paper we have expanded the model and report a comprehensive empirical evaluation of the model using a rich, longitudinal dataset drawn from commercial enterprise software package installations. [↑](#footnote-ref-2)
3. 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. [↑](#footnote-ref-3)
4. 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. Customers in the lower-debt package group typically omitted such modules with low test coverage and “beta” transactions. [↑](#footnote-ref-4)
5. 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% response rate. We signed a collective non-disclosure agreement with the vendor firm and the 75 customers. With the permission of the firms that declined to participate in our study, we did check their data for potential systematic non-responsiveness bias only. No major differences across the participating and non-participating firms were observed. [↑](#footnote-ref-5)
6. We tested the variations at multiple time intervals, spanning the evolution of the software package—before takeoff, during the growth phase, and during the saturation phase—and verified that the distribution of variance follows a normal curve pattern in all cases. The normal distribution of variance among the customer installations throughout the lifespan of the package allows us to utilize statistical tests that rely on normal distributional patterns, such as the t-tests of mean values of different groups. [↑](#footnote-ref-6)