Environmental Volatility, Development Decisions, and Software Volatility: A Longitudinal Analysis

Evelyn J. Barry
Department of Information and Operations Management, Mays Business School, Texas A&M University, 320J Wehner, College Station, Texas 77843, ebarry@mays.tamu.edu

Chris F. Kemerer
Katz Graduate School of Business, University of Pittsburgh, 278A Mervis Hall, Pittsburgh, Pennsylvania 15260, ckemerer@katz.pitt.edu

Sandra A. Slaughter
David A. Tepper School of Business, Carnegie Mellon University, 354 Posner, Pittsburgh, Pennsylvania 15213, sandras@andrew.cmu.edu

Although product development research often focuses on activities prior to product launch, for long-lived, adaptable products like software, development can continue over the entire product life cycle. For managers of these products, the challenges are to predict when and how much the products will change and to understand how their development decisions influence the timing and magnitude of future change activities. We develop a two-stage model that relates environmental volatility to product development decisions and product development decisions to software volatility. The model is evaluated using a data archive that captures changes over 20 years to a firm’s environment, its managers’ development choices, and its software products. In Stage 1 we find that higher environmental volatility leads to greater use of process technology and standard component designs but less team member rotation. Earlier development decisions strongly influence current development choices, especially for product design and process technology. In Stage 2 we find that increased use of standard component designs dampens future software volatility by decreasing the average rate and magnitude of change. Adding new team members increases product enhancements at a faster pace than more intense use of process technology but adds repairs at almost the same rate as enhancements.

Key words: product development decisions; product volatility; product life cycle; software volatility; environmental volatility; software development; software evolution; contingency theory; software maintenance; team management; software process; standard designs; organizational inertia

History: Accepted by Karl Ulrich, technological innovation, product development, and entrepreneurship; received July 15, 2003. This paper was with the authors 1 year, 2 months, for 2 revisions.

1. Introduction
Although one may think of product development as a set of events that takes place before the launch of a product, for long-lived, adaptable products like software, development activities continue to occur after the initial launch. Software products are often repaired and modified, and new features are added in an incremental development life cycle that can last for decades. In fact, the development activities occurring postlaunch (i.e., after initial implementation) can account for as much as 90% of the total software product life cycle (Bennett 1996).

As more products begin to include software, understanding the life-cycle behavior of software has become a more widely relevant issue. It is particularly difficult for managers to estimate when and how many resources will be needed to update software products that have varying degrees of volatility. Software volatility refers to both the length of the time between product modifications and the magnitude of these modifications. These modifications occur postlaunch as a result of life-cycle activities, including corrections, adaptations, and enhancements (including new features) to the product (Barry 2001). Volatility is neither inherently good nor inherently bad. Rather, it can reflect an important investment to keep products useful when, for example, new requirements emerge during use, the business environment changes, errors are discovered and must be repaired, new equipment or technology must be accommodated, or software performance or reliability must be improved. However, it can be very challenging to predict how many products will need to be changed and when. Some products are relatively stable over their life cycles, while others are frequently updated; still others may have periods of stability and instability (Kemerer and Slaughter 1997). Thus, the use of simple heuristics, such as fixed product-update schedules, can be significantly nonoptimal (Banker and Slaughter 1997).
Instead, we believe that managers will vary their development decisions in response to environmental conditions and that their choices will, in turn, influence the future frequency and magnitude of software change. The product development literature suggests that environmental characteristics can influence the choice of a development strategy (Brown and Eisenhardt 1995, Krishnan and Ulrich 2001). For example, managers operating in very uncertain and dynamic domains may choose more flexible development processes, and this flexibility allows the products to be changed often so as to stay in synch with their environment (MacCormack et al. 2001). However, there is very little knowledge about ongoing development choices and their impacts on product volatility: Do development choices change as environmental conditions vary over a product’s life cycle, and, if so, how? How do prior development choices impact product volatility in future periods? Given the long life cycles of software-intensive products, research on their ongoing development is essential to improving management practice.

Much of the product development literature has examined how to manage a firm’s product development efforts at a given point in time. However, relatively little research has focused on how a firm’s products and product lines should evolve over time (Ramdas 2003). By examining product development using a temporal lens, our study provides a richer understanding of product evolution and the implications of product development decisions, as opposed to more common nonlongitudinal analyses (Ancona et al. 2001). We leverage a unique data archive compiled from both primary and secondary data sources about changes to a firm’s environment, its managers’ development choices, and its software products over as many as 20 years in length. Our cross-sectional, longitudinal data panels afford us an unusual opportunity to examine how development choices and their impacts on volatility differ across products and over time as a firm’s environment changes. Such a perspective is increasingly important, because ongoing product development cycles characterize not only pure software products but also a growing number of manufactured products, such as automobiles, airplanes, and medical equipment, that have significant amounts of embedded software (Koopman 1996). Our study also extends the growing literature on contingency theory in product development by elaborating and empirically evaluating a two-stage model that associates environmental volatility with development decisions in one period, and then predicts the effects of those decisions on product volatility in a future period (Shenhar 2001). A predictive model that links development choices to the dynamics of the environment and to product volatility has the potential to significantly improve decision making for adaptable products like software. Researchers can use the model to broaden and deepen their understanding of the transforming processes and dynamic behavior observed during product evolution. Managers can use the model to improve their ability to anticipate change and design adaptable products, while retaining a lifecycle perspective for product development.

The paper is organized as follows: §2 develops our theoretical model and hypotheses, §3 describes the empirical evaluation of our model, and §4 presents the analysis and results. In the final sections we discuss our results and draw out their implications for product development research and practice.

2. Theoretical Framework

Using contingency theory as a foundation, we develop a two-stage model. Stage 1 associates environmental volatility with decisions on product design, process technology, and team composition. Stage 2 relates these development decisions to future software volatility.

2.1. Stage 1: Environmental Volatility and Product Development Decisions

The notion of a relationship between variations in a firm’s environment and its structures, strategies, and performance is fundamental in the strategy and organizational literatures (e.g., Hannan and Freeman 1984). In particular, environmental volatility is seen as a key characteristic of a firm’s context that influences its managers’ choices. Environmental volatility refers to the level of instability or unpredictability faced by an organizational unit (Dugal and Gopalakrishnan 2000, Dess and Beard 1984). According to contingency theory, firms are open systems faced by uncertainty but requiring clarity and certainty to function efficiently. Thus, management’s primary role is to reduce uncertainty by developing coping strategies that avoid, adjust to, reduce, or take advantage of uncertainties (Thompson 1967). A basic proposition of contingency theorists is that firms enhance their performance to the extent that they align their organizational structures and strategies to conditions in the external environment.

Similarly, the literatures on manufacturing and product development suggest that when firms experience increasing environmental volatility, managers will choose a more adaptable production approach to enhance their flexibility (Anand and Ward 2004, Mendelson and Pillai 1999). Flexibility offers managers the ability to increase the speed and ease with which they can respond and adapt to changing circumstances (Gerwin 1993). As noted by Krajewski and Ritzman (2002), the basic elements of a flexible strategy consist of decisions about product design,
process, technology, and people. In the next sections, we consider how such decisions would respond to environmental conditions.

2.1.1. Environmental Volatility and Product Design Choices. The extent to which a product can be easily adapted or tailored is an important aspect of flexibility in manufacturing, and the effort required to change a product is influenced by the product’s design. A modular design maps each product function to a component and specifies decoupled interfaces between components, while an integral design maps product functions to multiple components and/or has coupled component interfaces (Ulrich 1995). A modular design affords component standardization. Standard components involve the use of the same component in multiple products within a single firm or across multiple firms. Generally, products with a modular design and standard components are more easily adapted and tailored than products with an integral design because a modular design isolates changes to particular independent components, while an integral design also requires changes to related components (Ulrich 1995, Ramdas 2003). In addition, a modular design can facilitate product variety through a process of combining different components and focusing differentiation on specific components. Thus, in a volatile environment, a modular design with standard components should be preferred because this design is flexible and allows changes to be localized to the minimum possible number of components (Baldwin and Clark 2000, Sanchez and Mahoney 1996).

Consistent with the concepts of modular design, we would expect software products composed of a larger number of standard components to be more easily changed (Nidumolu and Knotts 1998). A “standard” software component refers to a particular design for a software module or program that is widely used across software products within a firm or in software products used by many firms (IEEE 1998). The standard software component design can be either developed internally or purchased. Standard software components can be updated or recombined with relatively small changes to the overall software product because the components are independent and the changes do not propagate across related components. An increased use of standard software component designs is thus likely to facilitate responsiveness to changes in information requirements arising from an unstable environment. This implies the following:

Hypothesis 1. A higher level of environmental volatility is associated with greater use of standard component designs to develop a software product.

2.1.2. Environmental Volatility and Process Technology Choices. Production technology choices have also been shown to have a strong influence on manufacturing flexibility and responsiveness. Although the goal of process technology has generally been to improve process efficiencies by achieving lower costs and higher reliability through dedicated and fixed production, certain kinds of automation facilitate more agile manufacturing approaches (Parthasarthy and Sethi 1992). In particular, programmable automation technologies increase flexibility by allowing more diversity in the variety and volume of components produced and by reducing the time needed to change product designs (Kelley 1994). In new product development efforts, flexible design technologies help developers more easily change product designs throughout the development process (Thomke and Reinertsen 1998).

In software product development, computer-aided software engineering (CASE) tools automate the analysis and development process by providing facilities for diagramming, requirements specification, design, quality management, documentation, and generation of models and code (Whitten et al. 2004). The tools make it possible to minimize effort even while increasing the changes occurring in the code through forward and reverse engineering, reuse, and adaptation of product designs. CASE tools are similar to the programmable automation technologies used in manufacturing because the tools create economies of scale and scope in software component creation and adaptation. With high environmental volatility, the use of process technology should become more attractive because CASE tools enable greater responsiveness to changes in the software product’s information requirements.

Hypothesis 2. A higher level of environmental volatility is associated with greater use of process technology in developing a software product.

2.1.3. Environmental Volatility and Development Team Choices. The team is central to the product development process, as team members accomplish the important tasks of articulating product specifications and transforming them into the design, development, and implementation of new products. In particular, team composition has been recognized as critical to the success of product development efforts (Clark and Wheelwright 1998). One important aspect of team composition is team tenure, as the performance of a product development team is strongly influenced by the extent to which the team members have worked together on prior related projects (Brown and Eisenhardt 1995). Researchers in organizational psychology have found through experimentation that teams where members are less familiar are consistently slower in completing tasks than
teams that have worked together before, as the prior experience of working together provides a basis for coordination and division of labor and reduces process losses in team interactions (Harrison et al. 2003). Researchers also suggest that team tenure has increasing importance for performance as the uncertainty of the task environment increases because familiar teams can respond more quickly to changing conditions due to their larger and better organized knowledge, better internal problem representations, and more automated responses to work stimuli (Goodman and Shah 1992, Allen 1977, Clark and Wheelwright 1998). Similarly, a software development team is composed of the developers who develop or update a software product over its life cycle, and whenever a new developer is assigned, the product team is less productive because any new team member is familiar with neither the product nor the other team members (Sacks 1994). We would expect that when environmental volatility is high, managers would be less likely to rotate new members onto the software product development team, as the ability of the team to respond quickly and effectively to changes would suffer.

Hypothesis 3. A higher level of environmental volatility is associated with less use of team rotation to develop a software product.

2.1.4. Organizational Inertia and Development Choices. Consistent with contingency theory and the literature on strategic decision making, we identified the level of environmental volatility as a primary determinant of development choices. However, firms do not always succeed in quickly adapting their strategies in response to changes in the environment, particularly when there is organizational inertia. Inertia occurs when the speed of change in the firm is slower than the rate at which environmental conditions change. Hannan and Freeman (1984) suggest that decisions relating to technology, personnel, work practices, and standards are particularly subject to inertia because they are at the “core” of the firm. Further, once inertia begins to build, it increases because of a self-reinforcing dynamic—when inertia is high, managers may be less able to recognize and respond to the need for change (Sastry 1997). In our study this implies that development decisions made in a prior period could influence current decisions. Thus, we include the lagged choice for each respective decision variable in our analysis to assess the extent to which current development choices are influenced by prior choices.

2.2. Stage 2: Development Decisions and Software Volatility

In theorizing how development decisions impact future software product volatility we distinguish between two basic types of activities: corrections and enhancements. Corrections include activities to repair software product defects. Corrective volatility refers to the length of time between software product repairs and to the magnitude of the repairs. A software product with high corrective volatility has a short time between repairs and/or a large number of components requiring repairs. Enhancements are activities that modify or extend the functionality of the software product, including the addition of new features. Enhancement volatility refers to the length of time between software product enhancements and to the magnitude of the enhancements. A software product with high enhancement volatility experiences a short time between enhancements and/or a large number of components added or modified. In the following sections we consider how each development choice would impact both future software product enhancement volatility and corrective volatility.

2.2.1. Product Design Choices and Future Software Volatility. Contingency theory suggests that a modular design with standard components is more likely to be selected for software products with volatile information requirements because the required changes can be localized to the minimum possible number of components. In addition, the use of modular design should facilitate adaptability, as standard components can be updated or recombined with smaller changes to the overall software product. Thus, a greater use of modular design should increase the subsequent enhancement volatility for those particular components that are subject to change. However, at the software product level, if only a few components are evolving, the average frequency and magnitude of product enhancement volatility will be lower. Although certain components could experience high volatility, the product as a whole is more stable. In contrast, for products with an integral design, a change is likely to require a change to many components. Therefore, a greater use of modular design with standard components should be associated with lower future enhancement volatility on average, at the product level. Further, because standard components have been used and tested across many products or firms, such components are, all else being equal, likely to have higher design integrity, validity, and quality than custom components developed for one particular product (Georgiadou 2003, Ulrich 1995). Thus, a greater use of standard component designs should help to preserve the integrity of the product’s structure, reducing the need for future corrections (Abrahamson 2004). This line of reasoning implies the following:

Hypothesis 4a. An increase in the use of standard component designs to develop a software product is associated with a decrease in the product’s future enhancement volatility.
Hypothesis 4b. An increase in the use of standard component designs to develop a software product is associated with a decrease in the product’s future corrective volatility.

2.2.2. Process Technology Choices and Future Software Volatility. Using contingency theory we proposed that process technology in the form of CASE tools is more likely to be selected for the development of software products that have volatile information requirements. This is because CASE tools can readily accommodate changes to the software design, code, and documentation (McMurtrey et al. 2000, Banker and Slaughter 2000, Whitten et al. 2004). Greater use of process technology should both facilitate a software product’s future adaptability and decrease its number of defects. The use of automated tools with their ability to consistently perform repeated functions should reduce human error in the creation of software that must meet precise syntax and other constraints. Although reliable quantitative data on the impact of CASE tools is scarce, studies of firms using these tools suggest that the use of CASE tools significantly improves software product quality and maintainability (e.g., Limayem et al. 2004, Finlay and Mitchell 1994). Thus,

Hypothesis 5a. An increase in the use of process technology to develop a software product is associated with an increase in the product’s future enhancement volatility.

Hypothesis 5b. An increase in the use of process technology to develop a software product is associated with a decrease in the product’s future corrective volatility.

2.2.3. Development Team Choices and Future Software Volatility. We have reasoned that a dedicated team is more likely to be selected when environmental volatility is high because product development teams without a history lack effective patterns of working together and have lower performance than teams whose members have worked together in the past (Brown and Eisenhardt 1995). However, Brown and Eisenhardt have also noted that when product development teams work together too long, their performance can suffer because members are too inwardly focused and neglect external communication. Thus, product development performance is highest when team tenure is moderate. In manufacturing, a team is considered more flexible when it is capable of doing many different tasks and can be shifted to jobs and projects as needed (Gerwin 1993). One way to create a flexible team is to rotate members across tasks and projects. In the context of software product development, rotating developers across products and teams could increase the firm’s ability to make future enhancements to its product portfolio, as more developers would become more familiar working with each other and with more products. All else being equal, managers will choose to make more enhancements to a product in the future with a flexible staff, because they can more easily assign developers to work on the product.

Hypothesis 6a. An increase in the rotation of a software product’s team members is associated with an increase in the product’s future enhancement volatility.

Although rotating workers across teams and projects can increase flexibility in staff assignment, the literature on team familiarity suggests that teams produce lower-quality output when some or all members don’t know each other well (Harrison et al. 2003). When new developers are assigned to work on the software product, they are initially less familiar with its design and code structure and may have little experience working with other team members. New team members are likely to make more errors (requiring future corrections) and are more likely to be less efficient when updating the software product because they are unfamiliar with it. Software developers who are unfamiliar with the source code are likely to change the code more often, change more code than necessary, and change more components than needed, all of which will contribute to future corrective volatility (Sacks 1994).

Hypothesis 6b. An increase in the rotation of a software product’s team members is associated with a decrease in the product’s future corrective volatility.

2.2.4. Product Characteristics and Future Software Volatility. In our model we control for four characteristics of software products that could influence corrective and enhancement volatility but that are less amenable to managerial action: product domain, complexity, size, and age. The product domain sets the boundary of the product’s task environment and includes the problem structure as well as the size and variety of the stakeholders. Some task domains are well structured, with consistent information requirements for all stakeholders, while others are illstructured, with uncertain and widely varying requirements from a variety of stakeholders (Prahalad and Krishnan 1999). In ill-structured domains, every time the task definition and structure change, the information requirements change. This requires the software product to adapt, and every such change has the potential to introduce errors. Thus, the average levels of corrective and enhancement volatility may vary by domain, and we control for differences in product domain in our analysis.

The inherent properties of software products are often reduced to measures of their complexity, size, and age. Both increased complexity and increased
size have been shown in prior research to be significant in predicting the future occurrence of software faults (Kemerer 1995). As the size and complexity of the software grow, the likelihood of making errors increases. This requires future work to repair and adapt the code, and therefore increases corrective volatility. Enhancement volatility is also likely to increase with size and complexity as the larger, more complex products are performing more functions and would thus have more functionality subject to change compared to the smaller, simpler products. Finally, the laws of software evolution as articulated by Lehman et al. (1997) suggest that as software products age there is likely to be an increasing divergence between the software and its technical and organizational environments. Resolution of these discrepancies requires modifications, and every modification has the potential to introduce errors, leading to both increased corrective volatility and increased enhancement volatility with age.

Figure 1 depicts the complete research model.

3. Method

3.1. Research Setting and Data

We evaluate our research model empirically analyzing cross-sectional, longitudinal panels of data collected from both primary and secondary sources. The research site is a large, publicly owned conglomerate of retail stores, and its software products include business-oriented information systems that are developed in house or purchased and adapted for the firm’s internal use. The firm has a portfolio of software products including more than 3,000 components. A product corresponds to a major business application in the firm. A component is a self-contained software module or program that accomplishes a specific task, such as printing a report or providing a screen for data entry. A centralized information technology (IT) group supports this large and varied software product portfolio.

Throughout the 20-year history of the software product portfolio the firm’s IT group maintained a comprehensive log of every update made to the software products from the 1970s to the 1990s, providing detailed data describing change events over that time period. Each change event includes textual data describing the original software component creation date and author, the function of the component, the product to which the component belongs, the developer changing the component, the date of the change, and a description of the change (Kemerer and Slaughter 1999). The change event data are complemented by archival data we collected from the firm’s software change control system, including source code as well as the technologies used to create the code. The source code was available for all components—whether developed internally or externally sourced from a vendor. We extracted the source code and analyzed it using a commercial code analysis tool to generate measures of software size and complexity. Because the firm is publicly held, financial reports to stockholders were available for the time frame of the study and were used to derive financial information about the firm (such as its sales, profits, inventory levels, the number of stores, and the

1The product portfolio includes 23 applications: advertising, accounts payable, accounts receivable (3 products), sales analysis (2 products), capital pricing management, fixed asset management, financial reporting, general ledger, shipping, merchandising (3 products), order processing, pricing (5 products), payroll, and inventory management.
3.2. Constructs and Measures

3.2.1. Software Volatility Measures. We assess two aspects of software volatility—the frequency of change and the magnitude of change—for two major types of changes—product corrections and product enhancements. To measure the frequency of changes to a software product, we calculate the average length of the time interval between changes to its components in each month of its life cycle. A product with an increase in corrective (enhancement) volatility will experience corrections (enhancements) at shorter, more frequent intervals, while a product with a decrease in corrective (enhancement) volatility will experience less frequent, longer intervals between corrections (enhancements). Specifically, the length of the average time interval between corrections is calculated by averaging the time since the previous correction (TSC) for each repair event for each component $c$ in software product $s$ during time period $t$:

$$AvgTSC_{st} = \frac{1}{C_{st}} \sum_{c=1}^{C_{st}} TSC_{cst}$$

where $C_{st} =$ # of corrections for product $s$ during time period $t$.

The calculation for enhancements ($AvgTSE_{st}$) is similar, with appropriate changes made.

To facilitate interpretation, we reverse score these measures by subtracting each from the number of months $M$ the product has been in existence at the end of time period $t$, so that a higher value indicates greater volatility, and a lower value indicates lower volatility. Thus, the corrective volatility, measured as the frequency of corrections to software product $s$ in month $t$, is defined as: $Corrective-Volatility[Frequency]_{st} = M_{st} - AvgTSC_{st}$. Enhancement (frequency) volatility is measured similarly.

We measure the magnitude of corrections (enhancements) to a software product by counting the number of its components that were corrected (enhanced or added) during the month. The corrective volatility, or magnitude of corrections, to software product $s$ in month $t$ is defined as: $Corrective-Volatility[Magnitude]_{st} =$ total # of components corrected for product $s$ during time period $t$. Enhancement (magnitude) volatility is measured similarly.

3.2.2. Environmental Volatility Measure. Following Dugal and Gopalakrishnan (2000), Dess and Beard (1984), and others, we view environmental volatility as the unpredictability in access to and utilization of key resources required by an organizational unit. In the context of our study the organization is a retailing firm. Thus, we obtained measures of important resource indicators for a retailer: sales, inventory levels, the number of stores, and the number of employees. Unpredictability in the levels of these indicators can influence the volatility that decision makers in the firm face by expanding or limiting the organizational resources available for their strategic actions. For example, unexpected levels of sales could influence the budget available for changes to the software product portfolio; the timing, nature, and magnitude of changes made to the software products; the usage and support requirements of the software products; and the implementation strategies for changes to the software products. Information on sales, inventories, stores, and employees was obtained for the time frame of the study from the firm’s 10K and 10Q financial reports to its stockholders. We measured sales in terms of the firm’s reported sales revenue in each month. The consumer price index (CPI) established by the U.S. Bureau of Labor Statistics was used to deflate sales to a common time period. Inventories were measured as the dollar value of the firm’s retail inventories for each month, adjusted for inflation using the CPI. Stores were measured as the number of retail stores in the firm for each month. Finally, we measured employees as the ratio of the number of employees to the number of months.

We also considered other variables, such as profits and market share, but these were highly correlated with many of the other financial measures. In addition, we obtained data on retail industry sales, profits, and inventory levels. Although senior executives may consider industry conditions in formulating the firm’s overall strategy, our paper is studying decision makers within a subunit of the firm (the IT unit), and these decision makers are more likely to be directly impacted by firm-level volatility than by industry factors. We thus decided to focus on firm-level measures. (Note that industry variables were highly correlated with the corresponding firm-level variables, and therefore this choice of variables does not materially affect the results.)

Data were not always available for all variables for each time period. Missing values within a reported quarter or year were calculated using a straight-line interpolation. If the missing values were for either sales or inventory, the straight-line interpolations for monthly values were adjusted to reflect seasonal patterns reported in the available monthly reports of sales; e.g., adjustments were made to reflect increased sales during the months in the fourth quarter. Missing values for the years prior to the earliest reported data were extrapolated from the earliest data available.
employees in the firm divided by the number of retail stores in the firm for each month.

Next we created a composite index of environmental volatility that incorporates the four resource indicators. Dess and Beard (1984) assert that the unpredictability of environmental change can be assessed by measuring environmental volatility in terms of the dispersion about the regression line obtained when a resource indicator is regressed on time. Following Dess and Beard, we first computed the average of each resource indicator (sales, inventory levels, stores, and employees) across the months in each product’s life cycle. We then regressed each resource indicator on a variable representing time (months) in each product’s life cycle and divided the standard error of each regression by the average of the resource indicator. This procedure generated a volatility measure for each resource indicator that varies by product. To allow for temporal variation in the volatility measures, we followed McNamara et al. (2003) and computed the average of each resource indicator across the months in each product’s life cycle, repeating the regressions year by year for each product. This procedure created a volatility measure for each resource indicator that varies by year for each product. We then used principal components analysis to reduce each pair of volatility measures (product-based and year-based) to four factors, one for each resource indicator. Finally, we used principal components analysis to reduce the four factors to one single factor, yielding a composite index of environmental volatility that varies by year and by product.5

3.2.3. Development Decision Measures. The choice of standard designs is measured using a ratio of the count of standard components to the total number of components in the product in each month. A standard component is a module that the firm developed or purchased from an external vendor and that is used many times in the firm or in many other firms. A custom component refers to the development of a unique module for the product within the firm that is not used by any other product or by any other firm. Information about whether each component is standard or custom is available from the software change control system (standard components have different naming conventions in the firm than custom components and so are easily identified). It is important to note that developers were not prevented from modifying either standard or custom components, and external vendors could also modify standard components that were purchased. Further, the change logs capture modifications to the standard and custom components used in this firm whether the changes were done in house or externally.

The use of process technology, i.e., CASE tool automation, is operationalized based on information in the IT group’s software change control system about the technology used to create each component in a software product. To calculate the measure, for every product and month, the number of components in the product that were created using CASE tools is divided by the total number of components in the product to create a proportional measure of the use of process technology.

Finally, the team rotation variable is assessed by counting how often updates to the software product are accomplished by a team member other than the team member who completed the previous software change. The change event logs identify the developers who created and modified the components in the software products, and this information allows us to recognize changes in developer-component assignments for a software product in each month over its life cycle. The number of changes in developer-component assignments is summed for each product each month to reflect the total number of changes made that month to the product team.

3.2.4. Product Characteristics Measures. Product domain is measured using a binary variable to distinguish between the primary business functions in the firm. In this firm, as in many others, there is a basic distinction between fiscal versus nonfiscal (i.e., operations) functions; our measure of domain is a binary variable where fiscal = “1” and nonfiscal = “0”. Product size is measured by counting the total number of components in the product in a particular month. Product complexity is assessed using information obtained from commercial code analysis tools. A complexity metric is calculated for each component by counting the total number of data elements referenced in the component. Total data complexity is a key indicator of a component’s inherent design complexity because it predicts the algorithmic complexity of the coded software (Al-Janabi and Aspinwall 1993). The total number of data elements referenced is summed across the components in each product to obtain a product-level measure of complexity.

We then calculate the relative change in product complexity by determining the difference between the total complexity in the product in one month and the total complexity in a prior month, divided by the total complexity in the prior month. Finally, product age is determined based on the age of the components in the

5 One composite index is extracted from the four resource indicator factors using the eigenvalue >1 criterion, and the index extracted explains 63.51% of the variation in the data. The loadings of the four resource indicators on the composite index range from 0.740 to 0.865, and the Cronbach’s alpha of reliability is 0.807, suggesting a cohesive and reliable index. The index is operationalized using factor scores that were computed by the regression method for each month in each year in each product.
product. The age of each component is obtained from the change event logs, and product age is measured by averaging the age (in months) for all components in the product in a particular month.

4. Analysis and Results

4.1. Model Specification

The data for evaluating our models were pooled into seven unbalanced, time series, cross-sectional panels. Three separate panels (one for each development decision variable: standard design, process technology, and team rotation) were created to relate measures of environmental volatility to development decisions in Stage 1. These data panels each contain 3,132 observations reflecting monthly measures of the firm’s environment and managers’ development decisions for the software products over the 20-year time frame of the study. Four other data panels of 3,132 observations reflecting monthly life cycle data for each of the firm’s 23 software products are used to relate measures of development decisions to measures of software volatility in Stage 2. The data panels are unbalanced because each software product has a different number of months in its life cycle. The shortest product life cycle is 62 months, and the longest is 246 months, with an average product life cycle of just under 164 months. All software products were in active use during the data-collection period.

The general simultaneous equations framework used for this analysis is of the form: $y_{it} = Y_{it} \gamma + X_{it} \beta + \epsilon_{it}$, where $y_{it}$ is the dependent variable (frequency or magnitude of corrective or enhancement volatility), $Y_{it}$ is a vector of observations for the endogenous variables (the development decision variables), $X_{it}$ is a vector of observations for the exogenous variables, and $\gamma$ and $\beta$ are vectors of coefficients. The Stage 1 model predicts the use of development choice $c$ ($c = 1$ to 3) for software product $s$ in time period $t - 1$ based on environmental volatility in time period $t - 2$:

$$\text{DEVELOPMENT-CHOICE}_{cst-1} = \lambda_0 + \lambda_1 * (\text{DEVELOPMENT-CHOICE}_{cst-2}) + \lambda_2 * (\text{ENVIRONMENTAL-VOLATILITY}_{s,t-2}) + \phi_{cst-2}. $$

We lag the explanatory variables in this specification because their impacts may not be immediate on software product development choices. Lagging the independent variables also helps to mitigate potential concerns with endogeneity (Kennedy 1998). Because the variables are measured using very different unit scales, to facilitate interpretation and comparison of the estimated coefficients we standardized each variable to its Z score before entering it into the regressions (Neter et al. 1990). The Stage 2 models predict the corrective and enhancement volatility (frequency and magnitude of change) of software product $s$ in time period $t$ from using development choice $c$ for software product $s$ in time period $t - 1$:

$$\text{SOFTWARE-VOLATILITY}_{st} = \eta_0 + \eta_1 * (\text{DOMAIN}_{s}) + \eta_2 * (\text{SIZE}_{st-1}) + \eta_3 * (\text{COMPLEXITY}_{s-1}) + \eta_4 * (\text{AGE}_{st-1}) + \eta_{5-7} * (\text{DEVELOPMENT-CHOICE}_{cst-1}) + \nu_{st}. $$

As in Stage 1, we lag our time-dependent explanatory variables in these specifications because their impacts on the different types of software volatility may not be immediate. We also standardized each variable to its Z score before entering it into the regressions.

4.2. Estimation of the Models

We estimated the equations using a two-step generalized least squares (GLS) regression procedure in which we first estimated the equations in Stage 1, and then used the fitted values for the development decision variables from these estimations as instruments in the Stage 2 GLS regression (Baltagi 2001, Greene 2003). Because we are estimating cross-sectional, time series panels we tested for cross-sectional heteroscedasticity as well as serial correlation. Lagrange multiplier tests suggested the presence of cross-sectional heteroscedasticity in the team rotation, corrective (frequency and magnitude) volatility, and enhancement (frequency and magnitude) volatility regressions (Greene 2003). Breusch-Godfrey tests indicated that there was autocorrelation in the standard design, process technology, team rotation, and corrective and enhancement (magnitude) volatility regressions and that the serial correlation parameter was different for each software product’s time series (Breusch and Godfrey 1981). Thus, we corrected for cross-sectional heteroscedasticity and panel-specific AR(1) correlation in these GLS regressions.\(^7\)

\(^6\)The duration of the lag was determined empirically. In this firm, corrections and enhancements to software products required approval and entry into the product development plan that was updated on a weekly basis. Thus, the effects of explanatory variables required at least one week to be manifested. We varied the length of the time period and found that less than one month was too short for an effect to be realized. At two months, the effects are considerably weaker, and beyond two months, the effects are not significant. Thus, we specified the model using a one-month lag for the time-dependent explanatory variables. A one-month lag is also consistent with much software management practice, where phenomena are often tracked monthly.

\(^7\)As a robustness check, we also estimated the coefficients for the reduced form of our models using the random effects two-stage least squares estimator (Baltagi and Chang 2000). The random effects estimator allows for a software product-specific disturbance ($\mu_s$) in addition to the “normal” disturbance ($\epsilon_{st}$). Results from these analyses are consistent with those from the two-step GLS procedure.
Hypothesis 2 predicted that higher levels of environmental volatility are associated with greater use of process technology. This hypothesis is supported, as an increase in the level of environmental volatility in the prior month is significantly associated with an increase in the use of CASE tools for the current month ($\lambda_{12} = 0.01$, $z = 3.07$, $p < 0.01$). Finally, Hypothesis 3 predicted that higher levels of environmental volatility are associated with less team rotation, and it is supported. As shown in Table 2, an increase in the level of environmental volatility in the prior month is significantly associated with fewer changes to the product team in the current month ($\lambda_{13} = -0.035$, $z = -3.83$, $p < 0.001$).

4.3. Results: Stage 1

The descriptive statistics and pairwise correlations for the variables in our Stage 1 model are displayed in Table 1, and the results from the GLS estimations of our Stage 1 model, after correcting for cross-sectional heteroscedasticity and panel-specific AR(1), are shown in Table 2.

As expected, the lagged dependent variable for each development decision is positive and significant, suggesting that organizational inertia is influencing future period development choices. Inertia is especially influential for choices of standard design ($\lambda_{11} = 1.00$, $z = 757.28$, $p < 0.001$) and process technology ($\lambda_{22} = 0.99$, $z = 403.29$, $p < 0.001$) but also influences the choice of team rotation ($\lambda_{23} = 0.62$, $z = 43.67$, $p < 0.001$). Controlling for organizational inertia, we find empirical support for all three of our Stage 1 hypotheses. Hypothesis 1 predicted that higher levels of environmental volatility are associated with greater use of standard component designs. This hypothesis is supported, as an increase in the level of environmental volatility in the prior month is significantly associated with an increase in standard components for the current month ($\lambda_{11} = 0.006$, $z = 6.11$, $p < 0.001$).

4.4. Results: Stage 2

The descriptive statistics and pairwise correlations for the variables in our Stage 2 model are displayed in Table 3.

The results from the two-step GLS estimations of our Stage 2 equations are shown in the second and fourth columns of Table 4 for the corrective (frequency and magnitude) volatility regressions and in the third and fifth columns of Table 4 for enhancement (frequency and magnitude) volatility regressions. The coefficient on the binary variable for product domain is negative and significant in the enhancement frequency volatility equation (suggesting that nonfiscal application products experience shorter intervals between product enhancements than fiscal application products). The coefficients on software product size, complexity, and age suggest that increases in these variables in a prior month are generally associated with increases in the frequency and magnitude of product corrections and enhancements for the current month, although the coefficient on product complexity is significant only in the corrective magnitude volatility regression. Overall, the results for the control variables are mostly as expected.

Controlling for product domain, size, complexity, and age, we find empirical support for five of our six hypotheses in Stage 2. Hypothesis 4a and Hypothesis 4b predicted that increases in the use of standard component designs in a prior month are associated with fewer product enhancements and corrections in the current month, and are supported ($\eta_0 = -0.16$, $z = -11.40$, $p < 0.001$ and $\eta_2 = -0.08$, $z = -9.40$, $p < 0.001$, for the frequency and magnitude of product enhancements; and $\eta_5 = -0.07$, $z = -9.23$, $p < 0.001$ and $\eta_5 = -0.03$, $z = -5.63$, $p < 0.001$, for the frequency and magnitude of product corrections).

Hypothesis 5a predicted that increased use of process technology in terms of CASE tool use in a prior month is significantly associated with an increase in product enhancements for the current month, and it is
Table 3  Descriptive Statistics and Correlations, Stage 2 Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean (s.d.)</th>
<th>(1)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
<th>(5)</th>
<th>(6)</th>
<th>(7)</th>
<th>(8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corrective frequency, magnitude</td>
<td>18.86 (45.33), 0.77 (1.74)</td>
<td>1.00</td>
<td>0.51***</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enhancement frequency, magnitude</td>
<td>44.15 (65.34), 3.47 (6.79)</td>
<td>0.52***</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard design</td>
<td>0.13 (0.33)</td>
<td>−0.16**</td>
<td>−0.24***</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process technology</td>
<td>0.17 (0.26)</td>
<td>0.22***</td>
<td>0.19**</td>
<td>−0.25***</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Team rotation</td>
<td>3.35 (6.31)</td>
<td>0.34***</td>
<td>0.41***</td>
<td>−0.19**</td>
<td>0.33***</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domain</td>
<td>0.63 (0.48)</td>
<td>−0.13**</td>
<td>−0.08**</td>
<td>0.31***</td>
<td>−0.68**</td>
<td>−0.20**</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>74.27 (77.78)</td>
<td>0.46***</td>
<td>0.63***</td>
<td>−0.09*</td>
<td>0.41***</td>
<td>0.45***</td>
<td>−0.20***</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Complexity</td>
<td>0.07 (0.11)</td>
<td>0.06*</td>
<td>0.06*</td>
<td>−0.01</td>
<td>0.24***</td>
<td>0.30***</td>
<td>−0.13**</td>
<td>0.11*</td>
<td>1.00</td>
</tr>
<tr>
<td>Age</td>
<td>83.03 (58.37)</td>
<td>0.40***</td>
<td>0.73***</td>
<td>−0.06*</td>
<td>−0.03</td>
<td>0.22***</td>
<td>0.12**</td>
<td>0.55***</td>
<td>−0.03</td>
</tr>
</tbody>
</table>

Notes.  n = 3,132.

*p < 0.05; **p < 0.01; ***p < 0.001.

Pairwise correlations are listed in the following order: corrective frequency and enhancement frequency; corrective frequency and enhancement magnitude; corrective magnitude and enhancement frequency; corrective magnitude and enhancement magnitude.

Table 4  Generalized 2SLS Estimates for Stage 2

<table>
<thead>
<tr>
<th>Dependent variable</th>
<th>Corrective volatility (frequency)</th>
<th>Enhancement volatility (frequency)</th>
<th>Corrective volatility (magnitude)</th>
<th>Enhancement volatility (magnitude)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Estimated coefficient (standard error)</td>
<td>Estimated coefficient (standard error)</td>
<td>Estimated coefficient (standard error)</td>
<td>Estimated coefficient (standard error)</td>
</tr>
<tr>
<td>Intercept (β0)</td>
<td>−0.0757*** (0.0108)</td>
<td>0.6079*** (0.0132)</td>
<td>−0.1226*** (0.0131)</td>
<td>−0.0767*** (0.0135)</td>
</tr>
<tr>
<td>Domain (β1)</td>
<td>0.0087 (0.0186)</td>
<td>−0.0714*** (0.0180)</td>
<td>0.0404 (0.0266)</td>
<td>−0.0345 (0.0262)</td>
</tr>
<tr>
<td>Size (β2)</td>
<td>0.2357*** (0.0168)</td>
<td>0.3100*** (0.0178)</td>
<td>0.1223*** (0.0184)</td>
<td>0.2969*** (0.0203)</td>
</tr>
<tr>
<td>Complexity (β3)</td>
<td>−0.0003 (0.0089)</td>
<td>0.0053 (0.0110)</td>
<td>0.0129† (0.0066)</td>
<td>0.0030 (0.0031)</td>
</tr>
<tr>
<td>Age (β4)</td>
<td>0.0602*** (0.0114)</td>
<td>0.7767*** (0.0189)</td>
<td>−0.0164† (0.0091)</td>
<td>0.0284† (0.0122)</td>
</tr>
<tr>
<td>Standard design (β5)</td>
<td>−0.0657*** (0.0071)</td>
<td>−0.1641*** (0.0144)</td>
<td>−0.0298*** (0.0053)</td>
<td>−0.0759*** (0.0081)</td>
</tr>
<tr>
<td>Process technology (β6)</td>
<td>0.0746*** (0.0212)</td>
<td>0.0362 (0.0160)</td>
<td>0.2369*** (0.0341)</td>
<td>0.1604*** (0.0338)</td>
</tr>
<tr>
<td>Team rotation (β7)</td>
<td>0.2178*** (0.0255)</td>
<td>0.1520*** (0.0211)</td>
<td>0.2558*** (0.0244)</td>
<td>0.1029*** (0.0184)</td>
</tr>
<tr>
<td>Wald χ² test</td>
<td>1,170.23*** (7,169.89***</td>
<td>668.43***</td>
<td>992.68***</td>
<td></td>
</tr>
</tbody>
</table>

Notes.  n = 3,132.

† p < 0.10; †* p < 0.05; †** p < 0.01; †*** p < 0.001. Betas are standardized.
supported \( (\eta_0 = 0.04, z = 2.27, p < 0.05 \text{ and } \eta_6 = 0.16, z = 4.74, p < 0.001 \), for frequency and magnitude, respectively). Hypothesis 5b predicted that increased use of process technology in terms of CASE tool use is significantly associated with a decrease in product corrections. This hypothesis is contradicted, as increased use of process technology in a prior month is associated with an increase (not decrease) in product corrections for the current month \( (\eta_7 = 0.07, z = 3.52, p < 0.001 \text{ and } \eta_6 = 0.24, z = 6.94, p < 0.001 \), for frequency and magnitude, respectively).

Finally, the empirical results support both Hypothesis 6a and Hypothesis 6b on the effects of team rotation, indicating that an increase in the number of new developer-component assignments in a prior month leads to an increase in product enhancements and corrections for the current month \( (\eta_9 = 0.15, z = 7.22, p < 0.001 \text{ and } \eta_7 = 0.10, z = 5.60, p < 0.001 \), for the frequency and magnitude of product enhancements; and \( \eta_6 = 0.22, z = 8.54, p < 0.001 \text{ and } \eta_6 = 0.26, z = 10.47, p < 0.001 \), for the frequency and magnitude of product corrections).

5. Discussion
In this section, we first evaluate and interpret the results from our two-stage models. We then conduct and discuss the results from a Granger causality analysis of both models.

5.1. Impact of Environmental Volatility on Product Development Decisions
The empirical results from our Stage 1 model support our first set of hypotheses. The differences in the estimated coefficient values and their significance levels suggest that environmental volatility differs in the nature and strength of its impact on development choices. As can be seen in Table 5, the interpretations of the estimated coefficients from the model are that higher environmental volatility is associated with greater use of standard component designs and process technology, and, as predicted, with less use of team rotation.

As we proposed in Hypothesis 1, the use of standard component designs is likely to improve responsiveness to changes in information requirements and is thus more likely to be selected when environmental volatility is high. Similarly, the use of process technology (Hypothesis 2) enables greater responsiveness to changes in the software product’s information requirements and is therefore more attractive to product managers when environmental volatility is high. Finally, when environmental volatility is high, product development managers will prefer more stability in team assignments and are thus less likely to rotate team members across software components (Hypothesis 3).

Note that these marginal effects of environmental volatility on the decision variables occur after controlling for organizational inertia in the form of prior development decisions, which have a substantial impact on current development decisions. Overall, our results suggest that product development decisions are more sensitive to inertial forces, i.e., prior product development decisions, than to environmental volatility. Further, inertia appears to have a greater impact than environmental volatility on decisions concerning the use of standard component designs and process technology, relative to choices on team composition. Inertia could be especially salient for standard design and process technology because these choices are subject to high switching costs. For example, changes or additions to products created using standard designs would likely be constrained to fit the design. Similarly, when a product has been created using a CASE tool, changes or new features would probably need to be made using that tool or the affected components would be incompatible with the other components in the product. In contrast, changes to developer-component assignments may be relatively easier to accomplish, albeit at the cost of the time it takes the new developers to learn about the component and become familiar with the rest of the product team.

5.2. Impact of Development Decisions on Software Volatility
The empirical results from our Stage 2 model in Table 4 indicate that product development decisions implemented in one time period have a significant impact on software volatility in a future period. The estimated coefficients from this model are interpreted in Table 6. For example, an increase in the use of standard designs in a prior month significantly reduces product enhancements (Hypothesis 4a) and product

<table>
<thead>
<tr>
<th>Decision variable</th>
<th>Average level of decision variable</th>
<th>New level of decision variable from a 1 std dev ↑ in environmental volatility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypothesis 1</td>
<td>Standard design</td>
<td>13.46% standard components</td>
</tr>
<tr>
<td>Hypothesis 2</td>
<td>Process technology</td>
<td>16.37% CASE generated components</td>
</tr>
<tr>
<td>Hypothesis 3</td>
<td>Team rotation</td>
<td>3.34 new team members</td>
</tr>
</tbody>
</table>
corrections (Hypothesis 4b) in the current month. Table 6 also suggests that the dampening effect of standard designs on product volatility is stronger for product enhancements than for product corrections. As we discussed earlier, modular products with standard component designs allow changes to be localized to the minimum possible number of components. Thus, although a modular product is more adaptable than an integral product, the changes are confined to certain components, so the product as a whole is more stable.

To further test this interpretation, we conducted an analysis of the distribution of changes across components for the different types of products. We separated the products into two groups: those using standard designs and those not. Using each product’s change history data, we then computed the average and standard deviation of the number of changes per component in each group and of the proportion of components changed in each group. If changes to products created using standard designs are concentrated within a few components, we would expect to see greater relative variation in the number of changes per component and in the proportion of components changed because of the wider dispersion of values from the average (i.e., only a few components are changed, and thus the intensity of changes to those components should be high with respect to the other components in the product). In contrast, in products not created using standard designs, changes will affect more components and should be more evenly distributed among components, and this should lead to a smaller dispersion of values from the average and thus a lower relative variation in both the average number of changes per component and the average proportion of components changed.

Our analysis of the product change history data reveals that the products created using standard designs did indeed have a higher variation in the number of changes per component relative to products not using standard designs (coefficient of variation of 236.64% for products created using standard designs versus 175.96% for products not using stan-
opers who are unfamiliar with a software component should be aware of the product enhancements. However, the results also supported in Table 6, as an increase in the use of CASE tools expands the volume of product enhancements. This is due to the increased growth in team rotation helps to increase managers’ flexibility in making product team assignments and enables them to accomplish more product enhancements. This is supported in Table 6, as an increase in the use of team rotation increases the frequency and magnitude of product enhancements. However, these results also suggest that, as we posited in Hypothesis 6b, developers who are unfamiliar with a software component or who do not have prior experience working with the product team may make more errors or may be more inefficient in making updates, thereby triggering future corrections to the product. Table 6 shows that, in terms of the frequency of change, adding new team members intensifies the pace of product updates at four times the rate from greater use of process technology, shortening the average time interval between enhancements by almost 10 months (compared to just over two months shorter between enhancements for process technology). However, adding new team members also shortens the average time interval between repairs by almost 10 months (compared to just over three months shorter between repairs for process technology).

In terms of the magnitude of change, an increase in team rotation increases product corrective volatility relative to enhancement volatility at twice the rate from increased use of process technology. This suggests that the use of team rotation to increase product development responsiveness comes at a higher “cost” than the use of process technology. Thus, team rotation may be somewhat of a double-edged sword, as rotating developers across products triggers product repairs as quickly and in almost the same volume, as enhancements.

5.3. Granger Causality Analysis

Our analyses have shown associations between environmental volatility in a prior period and product development decisions in a future period and between product development decisions in a prior period and software volatility in a future period. Although lagging the time periods of the variables helps to mitigate potential endogeneity problems, it does not provide evidence of causality. As our data are nonexperimental, it is, of course, not possible for us to determine cause-and-effect relationships in a strict sense. However, a concept of causality in the economics literature can be evaluated using econometric techniques, namely Granger causality analysis (Granger 1969, Dutta 2001). A Granger causality analysis of our Stage 1 model involves a two-step process: (1) estimating autoregressive models for each of the product development decision variables in which the relevant lagged development decision variable is the only variable in the regression, and (2) adding the environmental volatility variable and examining the extent to which it explains significant incremental variation in the product development decision variables. A similar process is used for our Stage 2 model: (1) estimating an autoregressive model for software volatility in which the only predictor is lagged volatility, and (2) adding the development decision variables to assess the extent to which they explain significant incremental variation in software volatility. Likeli-

---

8 We computed the coefficient of variation when comparing the two groups of products because the means are different, and thus the standard deviations alone do not directly reveal the relative variation. For the number of changes per component, products using standard designs averaged 1.16 changes per component (sd = 2.75), and products not using standard designs averaged 8.61 changes per component (sd = 15.15). In terms of number of different components changed, products using standard designs averaged 26% of their components changed (sd = 0.44), and products not using standard designs averaged 73% of their components changed (sd = 0.44).
hood ratio tests are used to evaluate the significance of the variables added in the second step for each model (Greene 2003).

For Stage 1, Granger causality is supported for all development decision variables, as environmental volatility explains significant incremental variation in the use of standard component designs ($\chi^2 = 20.76$, $p < 0.001$), process technology ($\chi^2 = 9.22$, $p < 0.01$), and team rotation ($\chi^2 = 28.56$, $p < 0.001$). For Stage 2, our results also provide strong empirical support that prior product development decisions do cause, in the Granger sense, software volatility in a future time period, as the development decision variables explain significant incremental variation in both the frequency and magnitude of change for product corrections ($\chi^2 = 295.59$, $p < 0.001$ and $\chi^2 = 494.28$, $p < 0.001$) and for product enhancements ($\chi^2 = 39.72$, $p < 0.001$ and $\chi^2 = 1.346.04$, $p < 0.001$).

Although earlier development choices significantly influence current development choices, the results of the Granger causality analyses of the Stage 1 models suggest that development decisions on product design, process technology, and team composition are also adjusted in response to changes in environmental conditions. For product development managers in dynamic environments requiring frequent shifts in strategy, our results suggest that it may be worthwhile to consider the flexibility of decisions made on product design, process technology, and team composition, because these decisions may influence their future ability to respond to change. This is particularly relevant because the Granger causality analysis of the Stage 2 model suggests that product development managers can, through their development decisions, significantly influence the future frequency and magnitude of change to the software products and can therefore anticipate the volatility consequences of these decisions.

6. Conclusions

Our study makes a new contribution to the product development literature by revealing how product development decisions respond to volatility in the environment and how the dynamic behavior observed during software product life cycles relates to these decisions. As business environments become increasingly uncertain, managers need to understand how the decisions they make during product development influence their ability to anticipate, influence, and respond to change in the future. Although much of the literature has focused on the prelaunch phases of product development, managers who manage long-lived, adaptable products like software or traditional products that have significant software elements need to understand the implications of their ongoing development decisions. At this research site more than 87% of the software components in the firm’s portfolio were created after the initial implementation of each product. Thus, managers continue to make product development decisions over the product life cycle.

The ability of managers to respond rapidly to changes in their firms’ competitive environment is increasingly essential to firm survival. If software product managers could forecast changes to the long-lived products they manage, they could more effectively plan, allocate their workforce, and manage change processes. However, signals in the form of past patterns of product volatility are not normally made available to managers due to the effort required to accumulate, report, and analyze detailed change request data. In addition, the ability to forecast changes to software products depends on whether there are predictable patterns in life-cycle activities. The results from our study suggest that managers can predictably influence the nature and timing of software change processes through their design and development decisions.

Using a decision model such as the one we have developed, product managers can anticipate the volatility consequences of their development decisions, improve their ability to forecast change, and make more informed decisions about investments in designing flexible and adaptable products, while retaining a life-cycle perspective for product support. As such, managers could more confidently plan for the allocation of scarce resources to development projects over the life cycle of software products. For example, managers could allocate fewer resources to support products developed using more standard component designs, or managers could choose to reduce the future volatility of software products (and reduce the future resource requirements for a product) by keeping the product’s development team more stable.

We have focused on examining the impact of product development decisions in a single firm. This research design has several strengths. Evaluation of this firm’s unusual and extensive longitudinal data has afforded a rare opportunity to investigate the dynamic behavior of software products over their life cycles. The focus on one firm and development effort has the advantage of naturally controlling for contextual factors that could differ across firms. We also added variables in our analysis to control for other factors (such as product domain, size, complexity, and age) that could contribute to product volatility over the period of the study. These controls strengthen the internal validity of our results. Although our focused research design enhances the internal validity of our findings, the external validity could be limited to organizational and product development contexts.
similar to the firm in our study. We would expect that our results would generalize to contexts where product managers can choose whether to make or purchase software components for the firm’s internal use. However, our results may also generalize to contexts where software products are developed for sale. As no single study is definitive, the external validity of our results can be strengthened by replication. Thus, it is important for future research to examine the volatility of products and development decisions in other domains and development settings.

Further, while our intent has been to understand the antecedents of product volatility, it is important for future work to examine how volatility relates to other product attributes, such as costs. It would also be interesting to examine the difference between planned volatility and the volatility actually experienced. All this research will be particularly salient for products, like software, that are developed, enhanced, and maintained over long life cycles.

Acknowledgments

Funded in part by National Science Foundation grants CCR-9988227 and CCR-9988315, a Research Proposal Award from the Center for Computational Analysis of Social and Organizational Systems, NSF IGERT at Carnegie Mellon University, the Software Industry Center at Carnegie Mellon University, and the Institute for Industrial Competitiveness at the University of Pittsburgh. The authors thank the associate editor, two anonymous reviewers, M. Cusumano, A. MacCormack, W. Richmond, and participants in seminars at Arizona State University, Ohio State University, Purdue University, Texas A&M University, the University of Calgary, the University of California at Irvine, and the University of Southern California for their valuable comments on prior versions of this manuscript.

References


McNamara, G., P. Vaaler, C. Deevers. 2003. Same as it ever was: The search for evidence of increasing hypercompetition. *Strategic Management J.* 24(3) 261–278.


