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

Evelyn Barry • Chris Kemerer • Sandra Slaughter

Mays Business School, Texas A&M University, College Station, TX
Joseph M. Katz Graduate School of Business, University of Pittsburgh, Pittsburgh, PA
David A. Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA
ebarry@mays.tamu.edu • ckemerer@katz.pitt.edu • 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 a challenge is 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 twenty years to a firm's environment, its managers' development choices, and its software products. In stage one 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 two 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.

(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.)

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 post-launch (*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 the length of the time between product modifications and to the magnitude of these modifications. These modifications occur post-launch 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, and 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 non-optimal (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 non-longitudinal analyses. 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 twenty 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 the 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 to design adaptable products, while retaining a life cycle 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 de-coupled 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 the creation of product variety through a process of combining different components together 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 either be 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 that:

H1: *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 to 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, re-use, 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. Thus,

H2: *A higher level of environmental volatility is associated with greater use of process technology to develop 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 the 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. Consequently,

H3*: A higher level of environmental volatility is associated with less use of team rotation to develop a software product***.**

2.1.5. 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 the 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 re-combined 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 of its 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 less 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 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 that:

H4a: *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*.

H4b: *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,

H5a: *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*.

H5b: *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 earlier 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 do more enhancements to a product in the future with a flexible staff, because they can more easily assign developers to work on the product. Thus,

H6a: *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 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). Therefore,

H6b: *An increase in the rotation of a software product's team members is associated with an increase in the product's future corrective volatility*.

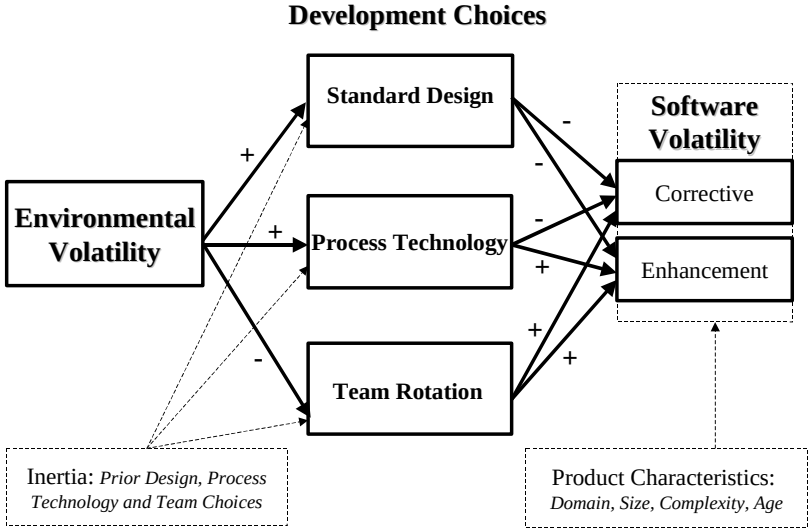
2.2.5 Product Characteristics and Future Software Volatility. In our model we control for four characteristics of software products that could influence corrective and enhancement volatility, but that are less amenable to managerial action: *product domain, complexity, size,* and *age*. The product domain 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 ill-structured 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.

Figure 1: 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 over three thousand 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.[1]

Throughout the twenty 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 1970's to the 1990's, providing us with 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 with archival data we collected from the firm's software change control system that included 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 number of employees) and to provide background information about the firm's strategies, markets, and activities. Finally, we obtained yearly reports from the IT group's internal library that described the firm's IT strategies, performance and policies.

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.[2] To measure the frequency of changes to a software product, we calculate the average

[1] The 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.

[2] As described earlier, corrections include activities to repair defects in the product, while enhancements include activities to modify or extend the functionality of the product, including the addition of new components.

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) occurring 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 *e* for each component *c* in software product *s* during time period *t*:

$$AvgTSC_{st} = \frac{1}{C_{st}} \sum_{e=1}^{Cst} TSC_{est} \quad \textbf{\textit{where}}\ C_{st} = \#\ of\ corrections\ for\ product\ \textbf{s}\ during\ time\ period\ \textbf{t.}$$

The calculation for enhancements *($AvgTSE_{st}$)* is similar, *mutatis mutandis*.

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.[3] Thus, the corrective volatility, measured as the frequency of corrections to software product *s* in month *t*, is defined as: $Corrective\text{-}Volatility[Frequency]_{st} = M_{st} - AvgTSC_{st}$. 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: $Corrective\text{-}Volatility[Magnitude]_{st}$ = *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,

[3] The maximum possible time between corrections or enhancements to a software product is its age in months at time *t;* thus, subtracting the average time between corrections or enhancements from the maximum reverse scores the respective measures. A product with components that do not experience corrections (enhancements) in a particular month has a corrective (enhancement) volatility value of zero (zero) for that month. Note that no products were retired during the time frame of the study.

inventory levels, the number of stores, and the number of employees.[4] 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.[5] 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 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) have asserted 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

[4] 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 sub-unit 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 (Mendelson and Pillai 1999). 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.)

[5] 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.

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 year 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.[6]

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 used 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 upon 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

[6] 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.

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 non-fiscal (*i.e.*, operations) functions; our measure of domain is a binary variable where fiscal = "1" and = "0" otherwise. *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 upon the age of the components in the product. The age of each component is obtained from the change event logs that record the date on which the component was created, 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 twenty-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 differing 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: $y_{it} = \mathbf{Y}_{it}\gamma + \mathbf{X}_{1it}\beta + \varepsilon_{it}$ where y_{it} is the dependent variable (frequency or magnitude of corrective or enhancement volatility), $\mathbf{Y}_{it}$ is a vector of observations for the endogenous variables (the development decision variables), $\mathbf{X}_{1it}$ is a vector of observations for the exogenous variables, and γ and β 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*: $\text{DEVELOPMENT-CHOICE}_{cst-1} = \lambda_{0c} + \lambda_{1c}*(\text{DEVELOPMENT-CHOICE}_{cst-2}) + \lambda_{2c}*(\text{ENVIRONMENTAL-VOLATILITY}_{t-2}) + \phi_{cst-2}$. 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*: SOFTWARE-

$VOLATILITY_{st} = \eta_0 + \eta_1*(DOMAIN_s) + \eta_2*(SIZE_{st-1}) + \eta_3*(COMPLEXITY_{st-1}) + \eta_4*(AGE_{st-1}) + \eta_{5-7}*(DEVELOPMENT\text{-}CHOICE_{cst-1}) + \nu_{st}$. 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.[7] 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.[8]

4.3 Results: Stage 1

The descriptive statistics and pair-wise 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.

[7] 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.

[8] 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 (μ_s) in addition to the "normal" disturbance (ν_{st}). Results from these analyses are consistent with those from the two-step GLS procedure.

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 (λ_{21} = 1.00, z = 757.28, p < 0.001) and process technology (λ_{22} = 0.99, z = 403.29, p < 0.001), but also influences the choice of team rotation (λ_{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. **H1** 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 (λ_{11} = 0.006, z = 6.11, p < 0.001).

Table 1: Descriptive Statistics and Correlations, Stage 1 Variables

Variable	Mean (s.d.)	(1)	(2)	(3)
(1) Standard Design	0.13 (0.33)	1.00		
(2) Process Technology	0.16 (0.26)	-0.25***	1.00	
(3). Team Rotation	3.34 (6.31)	-0.19**	0.33***	1.00
(4) Environmental Volatility	-0.01 (0.91)	-0.07*	0.19**	-0.04*

Notes: n = 3,132; * p < 0.05; ** p < 0.01; *** p < 0.001.

Table 2: GLS Estimates for Stage 1

	STANDARD DESIGN	PROCESS TECHNOLOGY	TEAM ROTATION
Independent Variables	**Estimated Coefficient** (standard error)	**Estimated Coefficient** (standard error)	**Estimated Coefficient** (standard error)
Intercept (λ_0)	**-0.0066*** (0.0034)	**0.0076**** (0.0027)	**-0.0508***** (0.0111)
Environmental Volatility (λ_1)	**0.0059***** (0.0010)	**0.0102**** (0.0033)	**-0.0346***** (0.0090)
Lagged Dependent Variable (λ_2)	**1.0045***** (0.0013)	**0.9878***** (0.0024)	**0.6182***** (0.0141)
Wald χ^2 Test	**573471.24*****	**225083.34*****	**1985.90*****

Notes: n = 3,132; † p < 0.10; * p < 0.05; ** p < 0.01; *** p < 0.001. Betas are standardized.

H2 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 (λ_{12} = 0.01, z = 3.07, p < 0.01). Finally, **H3** 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 (λ_{13} = -0.035, z = -3.83, p < 0.001).

4.4 Results: Stage 2

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

Table 3: Descriptive Statistics and Correlations, Stage 2 Variables

Variable	Mean (s.d.)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
(1) Corrective Frequency, Magnitude	18.86 (45.33), 0.77 (1.74)	1.00 0.51***							
(2) Enhancement Frequency, Magnitude	44.15 (65.34), 3.47 (6.79)	0.52*** [a] 0.42*** 0.30*** 0.52***	1.00 0.49***						
(3) Standard Design	0.13 (0.33)	-0.16** -0.15**	-0.24*** -0.19**	1.00					
(4) Process Technology	0.17 (0.26)	0.22*** 0.34***	0.19** 0.37***	-0.25***	1.00				
(5) Team Rotation	3.35 (6.31)	0.34*** 0.51**	0.41*** 0.54***	-0.19**	0.33***	1.00			
(6) Domain	0.63 (0.48)	-0.13** -0.26**	-0.08* -0.27***	0.31***	-0.68**	-0.20**	1.00		
(7) Size	74.27 (77.78)	0.46*** 0.34***	0.63*** 0.49***	-0.09*	0.41***	0.45***	-0.20***	1.00	
(8) Complexity	0.07 (0.11)	0.06* 0.21***	0.06* 0.17**	-0.01	0.24***	0.30***	-0.13**	0.11*	1.00
(9) Age	83.03 (58.37)	0.40*** 0.13**	0.73*** 0.27***	-0.06*	-0.03	0.22***	0.12**	0.55***	-0.03

Notes: n = 3,132; * p < 0.05; ** p < 0.01; *** p < 0.001. [a] pair-wise correlations are listed in the following order: corrective frequency & enhancement frequency; corrective frequency & enhancement magnitude; corrective magnitude & enhancement frequency; corrective magnitude & enhancement magnitude.

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 non-fiscal 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. **H4a** and **H4b** 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 (η_5 = -0.16, z = -11.40, p < 0.001 and η_5 = -0.08, z = -9.40, p < 0.001, for the frequency and magnitude of product enhancements; and η_5 = -0.07, z = -9.23, p < 0.001 and η_5 = -0.03, z = -5.63, p < 0.001, for the frequency and magnitude of product corrections).

Table 4: Generalized 2SLS Estimates for Stage 2

Dependent Variable	**Corrective Volatility (Frequency)**	**Enhancement Volatility (Frequency)**	**Corrective Volatility (Magnitude)**	**Enhancement Volatility (Magnitude)**
Independent Variables	**Estimated Coefficient** (standard error)	**Estimated Coefficient** (standard error)	**Estimated Coefficient** (standard error)	**Estimated Coefficient** (standard error)
Intercept (η_0)	**-0.0757***** (0.0108)	**0.6079***** (0.0132)	**-0.1226***** (0.0131)	**-0.0767***** (0.0135)
Domain (η_1)	**0.0087** (0.0186)	**-0.0714***** (0.0180)	**0.0404** (0.0266)	**-0.0345** (0.0262)
Size (η_2)	**0.2357***** (0.0168)	**0.3100***** (0.0178)	**0.1223***** (0.0184)	**0.2969***** (0.0203)
Complexity (η_3)	**-0.0038** (0.0089)	**0.0053** (0.0110)	**0.0129**† (0.0066)	**0.0030** (0.0031)
Age (η_4)	**0.0602***** (0.0114)	**0.7767***** (0.0189)	**-0.0164**† (0.0091)	**0.0284*** (0.0122)
Standard Design (η_5)	**-0.0657***** (0.0071)	**-0.1641***** (0.0144)	**-0.0298***** (0.0053)	**-0.0759***** (0.0081)
Process Technology (η_6)	**0.0746***** (0.0212)	**0.0362*** (0.0160)	**0.2369***** (0.0341)	**0.1604***** (0.0338)
Team-Rotation (η_7)	**0.2178***** (0.0255)	**0.1520***** (0.0211)	**0.2558***** (0.0244)	**0.1029***** (0.0184)
Wald χ^2 Test	**1170.23*****	**7169.89*****	**668.43*****	**992.68*****

Notes: n = 3,132; † p < 0.10; * p < 0.05; ** p < 0.01; *** p < 0.001. Betas are standardized.

H5a 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 supported (η_6 = 0.04, z = 2.27, p < 0.05 and η_6 = 0.16, z = 4.74, p < 0.001, for frequency and magnitude, respectively). **H5b** 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 ($\eta_6 = 0.07$, $z = 3.52$, $p < 0.001$ and $\eta_6 = 0.24$, $z = 6.94$, $p < 0.001$, for frequency and magnitude, respectively). Finally, the empirical results support both **H6a** and **H6b** 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 ($\eta_7 = 0.15$, $z = 7.22$, $p < 0.001$ and $\eta_7 = 0.10$, $z = 5.60$, $p < 0.001$, for the frequency and magnitude of product enhancements; and $\eta_7 = 0.22$, $z = 8.54$, $p < 0.001$ and $\eta_7 = 0.26$, $z = 10.47$, $p < 0.001$, 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.

Table 5: Interpretation of Stage 1 Model Coefficients

	Decision Variable	**Average Level of Decision Variable**	**New Level of Decision Variable from a 1 Std Dev ↑ in Environmental Volatility**
H1	Standard design	13.46% standard components	13.65% standard components (↑ of 1.48%)
H2	Process technology	16.37% CASE generated components	16.64% CASE generated components (↑ of 1.63%)
H3	Team rotation	3.34 new team members	3.12 new team members (↓ of 6.54%)

As we have proposed in **H1**, 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 (**H2**) 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 (**H3**).

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 time 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 (**H4a**) and product corrections (**H4b**) 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

have 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 that the product as a whole is more stable.

Table 6: Interpretation of Stage 2 Model Coefficients

	Decision Variable	**A 1 Std Dev ↑ in Prior Month Use is an:**	**Average Levels of Volatility** (frequency, magnitude)	**New Levels of Frequency Volatility** (*i.e., length of the average time interval between changes*)	**New Levels of Magnitude Volatility** (*i.e., # of components changed*)
H4a (*enhancement volatility*)	Standard Design	↑ of 34% in use of standard components	44 months and 5 days between enhancements, 3.47 components enhanced	54 months and 27 days between enhancements (↑ of 10 months and 22 days between enhancements)	2.95 components enhanced (↓ of 0.51 components enhanced)
H4b (*corrective volatility*)			18 months and 26 days between corrections, 0.77 components corrected	21 months and 26 days between corrections (↑ of 3 months between corrections)	0.72 components corrected (↓ of 0.05 components corrected)
H5a (*enhancement volatility*)	Process Technology	↑ of 26% in use of CASE tools	44 months and 5 days between enhancements, 3.47 components enhanced	41 months and 23 days between enhancements (↓ of 2 months and 12 days between enhancements)	4.56 components enhanced (↑ of 1.09 components enhanced
H5b (*corrective volatility*)			18 months and 26 days between corrections, 0.77 components corrected	15 months and 14 days between corrections (↓ of 3 months and 12 days between corrections)	1.18 components corrected (↑ of 0.41 components corrected)
H6a (*enhancement volatility*)	Team Rotation	↑ of 6 new team members assigned	44 months and 5 days between enhancements, 3.47 components enhanced	34 months and 7 days between enhancements (↓ of 9 months and 28 days between enhancements)	4.17 components enhanced (↑ of 0.70 components enhanced)
H6b (*corrective volatility*)			18 months and 26 days between corrections, 0.77 components corrected	8 months and 29 days between corrections (↓ of 9 months and 27 days between corrections)	1.21 components corrected (↑ of 0.44 components corrected)

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 - 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 standard designs).[9] In addition, products created using standard designs have a greater dispersion in the proportion of components changed (coefficient of variation of 168.46% for products created using standard designs versus 60.82% for products not using standard designs). Thus, changes are more localized to particular components when standard designs are used, as hypothesized.

The other development decision variables – process technology and team rotation - are both associated with increases in product volatility. CASE tools automate the development process and make it easier and faster for developers to create and adapt software products. Our results suggest that greater use of these tools to develop a software product increases the future rate and magnitude of enhancements to it (**H5a**). On the other hand, we had also theorized that the use of process technology should mitigate future corrective volatility (**H5b**) as the tools help to reduce human errors in coding. However, our empirical analysis suggests that the use of process technology is associated with *higher* corrective volatility. A closer consideration of Table 6 helps to interpret this seemingly contradictory finding. As can been seen in Table 6 although both enhancement and corrective volatility increase with greater use of CASE tools,

[9] 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).

enhancement volatility increases at more than twice the rate of corrective volatility (in terms of the magnitude of change). Thus, the increased use of CASE tools expands the volume of product enhancements more than repairs. One could conclude that, rather than contributing to poor product quality, process technology may, in fact, be helping product managers to keep correction rates lower than they would otherwise be, given the increased growth in enhancement activities and the number of components added and changed using the tools.

As we have posited in **H6a**, rotating developers 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 hypothesized in **H6b**, 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 ten 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 ten 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 non-experimental, 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. Likelihood ratio tests are used to evaluate the significance of the variables added in step 2 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 Granger-cause 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 $\chi^2 = 1346.04$, $p < 0.001$).

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 literature on product development 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 pre-launch 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 firm's 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 upon 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 a software product (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 time period of the study. These controls in our research design and model 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 of this research will be particularly salient for products, like software, that are developed, enhanced and maintained over long life cycles.[10]

7. References

Abrahamson, E. 2004. Managing change in a world of excessive change, *Ivey Business Journal Online*, January / February, www.iveybusinessjournal.com.

Al-Janabi, A., E. Aspinwall. 1993. An evaluation of software design using the Demeter tool. *Software Engineering Journal* 8(6): 319-324.

Allen, T. 1977. *Managing the Flow of Technology: Technology Transfer and the Dissemination of Technological Information within the R&D Organization*, Cambridge, MA: MIT Press.

Anand, G., P. Ward. 2004. Fit, flexibility and performance in manufacturing: Coping with dynamic environments. *Production and Operations Management*, 13(4): 369-383.

Baldwin, C. and Clark, K. 2000. *Design Rules*, Cambridge, MA: MIT Press.

Baltagi, B. 2001. *Econometric Analysis of Panel Data*, 2nd ed., New York: John Wiley and Sons.

Baltagi, B., Y. Chang, 2000. Simultaneous equations with incomplete panels. *Econometric Theory* 16: 269-279.

Banker, R., S. Slaughter. 2000. The moderating effects of software structure on volatility and complexity in software enhancement, *Information Systems Research* 11(3): 219-240.

Banker, R., S. Slaughter 1997. A field study of scale economies in software maintenance. *Management Science*, 43(12): 1709-1725.

Barry, E. 2001. *Software evolution, volatility, and life cycle maintenance patterns*. PhD Thesis. Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA.

Bennett, K. 1996. Software evolution: past, present and future. *Information and Software Technology*, 39(11): 673-680.

Breusch, T., L. Godfrey. 1981. A review of recent work on testing for autocorrelation in dynamic simultaneous models, in D. Currie, R. Nobay and D Peel, eds. *Macroeconomic Analysis: Essays in Macroeconomics and Econometrics*, Croon Helm, London, England, 63-105.

Brown, S., K. Eisenhardt. 1995. Product development: Past research, present findings, and future directions. *Academy of Management Review*, 20(2): 343-378.

Clark, K., S. Wheelwright. 1998. *Managing Product and Process Development*, New York: The Free Press.

Dess, G., D. Beard. 1984. Dimensions of organizational task environments. *Administrative Science Quarterly*, 29(1): 52-73.

[10] 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. We 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.

Dugal, M., S. Gopalakrishnan. 2000. Environmental volatility: A reassessment of the construct. *International Journal of Organizational Analysis*, 8(4) 401-424.

Dutta, A. 2001. Telecommunications and economic activity: An analysis of Granger causality. *Journal of Management Information Systems*, 17(4) 71-95.

Finlay, P., A. Mitchell. 1994. Perceptions of the benefits from introduction of CASE: An empirical study, *MIS Quarterly*, 18: 353-370.

Georgiadou, E. 2003. Software process and product improvement: A historical perspective. *Cybernetics and Systems Analysis*, 39(1) 125-142.

Gerwin, D. 1993. Manufacturing flexibility: A strategic perspective. *Management Science*, 39(4) 395-410.

Goodman, P., S. Shah. 1992. Familiarity and work group outcomes. In *Group Processes and Productivity*, Worchel, S., W. Wood, J. Simpson (Eds.), Newbury Park, CA: Sage Publications, 578-586.

Granger, C. 1969. Investigating causal relations by econometric models and cross spectral methods. Econometrica 37 424-438.

Greene, W. 2003. *Econometric Analysis*. 6th ed. Englewood Cliffs, NJ: Prentice-Hall, Inc.

Hannan, M.T., J. Freeman. 1984. Structural inertia and organizational change, *American Sociological Review*, 49 149-164.

Harrison, D., S. Mohamed, J. McGrath, A. Florey, S. Vanderstoep. 2003. Time matters in team performance: Effects of member familiarity, entrainment, and task discontinuity on speed and quality. *Personnel Psychology*, 56(3): 633-669.

IEEE. 1998. IEEE Standard #1016 for Software Design, New York: IEEE.

Kelley, M. 1994. Productivity and technology: The elusive connection, *Management Science*, 40(11) 1406-1426.

Kemerer, C. 1995. Software complexity and software maintenance: A survey of empirical research. *Annals of Software Engineering*, 1, 1-22.

Kemerer, C., S. Slaughter. 1999. An empirical approach to studying software evolution. *IEEE Transactions on Software Engineering*, 25(4), 1999, 1-17.

Kemerer, C., S. Slaughter. 1997. Determinants of software maintenance profiles: An empirical investigation. *Software Maintenance: Research and Practice*, 9, 235-251.

Kennedy, P. 1998. *A Guide to Econometrics*, 4th ed., Cambridge, MA: MIT Press.

Koopman, P. 1996. Embedded system design issues. *Proceedings of the International Conference on Computer Design*, Austin, TX, IEEE Computer Society Press, 310-317.

Krajewski, L., L. Ritzman. 2002. *Operations Management: Strategy and Analysis*, 6th ed., Prentice-Hall.

Krishnan, V., K. Ulrich. 2001. Product development decisions: A review of the literature. *Management Science*, 47(1) 1-21.

Lehman, M.M., J.F. Ramil, P.D. Wernick, D.E. Perry, and W.M. Turski. 1997. Metrics and laws of software evolution. Metrics '97, *The Fourth International Software Metrics Symposium*, Albuquerque, NM, 1997.

Limayem, M., M. Khalifa, W. Chin. 2004. CASE tools usage and impact on system development performance, *Journal of Organizational Computing and Electronic Commerce*, 14: 153-174.

MacCormack, A., R. Verganti, M. Iansiti. 2001. Developing products on 'Internet Time': The anatomy of a flexible development process. *Management Science*, 47(1) 133-150.

McMurtrey, M., Teng, J., Grover, V., H. Kher. 2000. Current utilization of CASE technology: Lessons from the field. *Industrial Management & Data Systems*, 100(1), 22-30.

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

Mendelson, H., R. Pillai. 1999. Industry clockspeed: Measurement and operational implications. *Manufacturing and Service Operations*, 1(1) 1-20.

Neter, J., W. Wasserman, M.H. Kutner, 1990. *Applied Liner Statistical Models Regression, Analysis of Variance, and Experimental Design*, 3rd Edition, Burr Ridge, IL: Richard D. Irwin, Inc.

Nidumolu, S., G. Knotts. 1998. Effects of customizability and reusability on perceived process and competitive performance of software firms. *MIS Quarterly*, 22(2) 105-137.

Parthasarthy, R., S.P. Sethi. 1992. The impact of flexible automation on business strategy and organizational structure. *The Academy of Management Review*, 17(1) 86-111.

Prahalad, C.K., M. Krishnan. 1999. The new meaning of quality in the information age. *Harvard Business Review*, 77(5) 109-118.

Ramdas, K. 2003. Managing product variety: An integrative review and research directions. *Production and Operations Management*, 12(1) 79-101.

Sacks, M. 1994. *On the Job Learning in the Software Industry: Corporate Culture and the Acquisition of Knowledge*, Westport, CT: Quorum Books.

Sanchez, R., J. Mahoney. 1996. Modularity, flexibility and knowledge management in product and organization design. *Strategic Management Journal*, 17: 63-76.

Sastry, M.A. 1997. Problems and paradoxes in a model of punctuated organizational change. *Administrative Science Quarterly*, 42(2), 237-275.

Shenhar, A. 2001. One size does not fit all projects: Exploring classical contingency domains. *Management Science*, 47(3) 394-414.

Thomke, S., D. Reinertsen. 1998. Agile product development: Managing development flexibility in uncertain environments. *California Management Review*, 41(1), 8-30.

Thompson, J.D. 1967. *Organizations in Action*, New York: McGraw-Hill.

Ulrich, K. 1995. The role of product architecture in the manufacturing firm, *Research Policy* 24(3) 419-220.

Whitten, J., Bentley, L., K. Dittman. 2004. *Systems Analysis and Design Methods*, 6th ed., NY: McGrawHill/Irwin.