ADAPTING TO A NEW ENVIRONMENT:
How A Legacy Software Organization Copes With Volatility and Change

NANCY STAUDENMAYER
The Fuqua School of Business
Duke University
Box 90120
Durham, NC 27708- 0120
Phone: (919) 660 – 7994
Email: nstauden@mail.duke.edu

TODD GRAVES
National Institute of Statistical Sciences

J.S. MARRON
Department of Statistics
University of North Carolina, Chapel Hill

AUDRIS MOCKUS
HARVEY SIY
LAWRENCE VOTTA
DEWAYNE PERRY
Lucent Technologies, Bell Laboratories

Research supported in part by NSF grants DMS—9208758 and SBR—9529926 to the National Institute of Statistical Sciences

Many of the key players in the telecommunications industry produce legacy products, software systems that were designed ten or even twenty years ago but also serve as the underlying basis for new products. A key challenge facing these firms is how to remain innovative and adaptive despite their ties with the past. This paper describes how one successful U.S. company coped with such a problem. Based on interview data, we identify three key transitions this organization underwent during the product's twenty year lifetime and test their effects using data from internal company databases. We found that environmental pressures to radically restructure the software system were inhibited due to certain inertial legacy factors. Instead, the organization coped by making more incremental adjustments to the product, organizational structure, coordination and control systems, and processes. A key proposition emerging from the study is that displacing change in this way increases the degree of technical and organizational interdependencies the firm is required to manage.

INTRODUCTION

Organizational change has become a pervasive topic among both practitioners and researchers of management, arguably capturing the attention of both groups more than any other topic in this decade. For managers of organizations, understanding how to cope with change is crucial for achieving competitive advantage, if not mere survival in today's business environment. The sheer importance and pervasiveness of the topic has in turn yielded a provocative set of research questions.

One stream of research focuses on the seemingly inherent forces, which serve to inhibit or constrain organizational adaptation. Since Hannan & Freeman [1, 2] first elaborated this idea in their theory of structural inertia, other researchers have catalogued various factors that affect the strength of inertia, including organizational characteristics [3-5], commitments to existing technology [6], the position of an organization in its institutional or competitive environment [7], and the extent or type of change in question [8].

More recently, some researchers have proposed that inertia is in fact a managerial variable. Particularly prominent in this line of reasoning are so-called multifaceted change models. According to this thinking, which clearly builds on the workings of structural inertia, incremental changes are inhibited because organizations are configurations of parts that form a logical whole. When such systems do change, they are therefore likely to do so in all-at-once jumps from one organizational configuration to another [5, 9]. The popularity of this model in part reflects the dominance of punctuated equilibrium models of change in current theories, which study change through the lens of a life cycle: abrupt and revolutionary transformations interrupt eras of incremental change [10].

Our article integrates and enlarges upon these growing streams of research in several important ways. We contribute first by exploring in greater depth how very real factors rooted in an organization's past impact the change process over time. We trace one organization over its twenty-year history and identify how decisions made early on and over time influence its ability to change now. We find that these inertial forces or ties to the past do not so much prevent change as constrain and shape the form and direction it must take.

We also examine how pressure for change on many fronts creates a complex mosaic of relationships inside the firm. While the resulting situation clearly corresponds to a multi-faceted change model, we focus on some of the complications and unintended consequences overlooked by previous scholars advocating this approach. In particular, our argument focuses on the disruption associated with coordinating multiple interdependent changes that may constrain and interfere with one another.

These results are captured in a conceptual model, which decomposes the environment this firm faces into distinct yet inter-related sub contexts—regulatory, competitive, customer, business community, and technological advances—and identifies change drivers in each. It then maps these change drivers to specific organizational domains and discusses the interaction of the resulting patterns.

Finally, we contribute methodologically by linking data on the subjective experience of change within this organization with more quantitative evidence on the type and extent of change that occurred. Grounding our more quantitative analysis in qualitative descriptions of key transitions increases the richness and validity of the findings.

The next section of the article provides some empirical motivation for our work and lays out the research questions. We then describe the research methodology, followed by a presentation of the results. The article concludes with discussion of the conceptual model and the implications of our results for theory and practice.

EMPIRICAL MOTIVATION

The original motivation for this study was to understand the phenomena of *"code decay"* in the software production process. Widely recognized in the software engineering community, code decay refers to the fact that as a software system ages and accumulates more and more changes, the cost to maintain and change the system typically increases [11]. One of the earliest empirical references to this phenomenon is based on study of IBM's OS 360, where researchers observed that, on average, each change to the system introduced a new fault, necessitating yet another change to the system to correct the fault [12].

Code decay is rapidly becoming a key management issue in firms today as software systems age and grow in size and complexity. In fact, many systems in existence today were designed and developed in the 1960's and are only now starting to show signs of maturity (i.e., the FAA Flight system). The company we study here, Lucent Technologies (formerly part of AT&T), is no exception. Managers at Lucent were particularly concerned about the implications of code decay for product development or cycle time.

To quantify this problem, we began by analyzing the extensive software change databases at Lucent to verify that it was in fact taking longer to make changes to the software. Defining the change interval as the lifetime of an individual change process in the database, the results showed a nearly 50% decrease—from a mean change interval of 10 days before 1990 to 6 days after 1992-- an apparently striking contrast to the concerns of management.

This unexpected result led us to expand the initial focus of our investigation, which yielded the results presented in this article. We conducted a number of interviews and gathered additional quantitative data on the software product, organization, and performance outcomes. The result is a framework, which explains how a software organization coped with a changing environment.

The study identified how certain legacy factors and coping mechanisms caused this striking discrepancy between the concerns of management and reality. The drive to shorten the change interval arose in the early 1990's after one major product release missed its delivery date. An extensive quality initiative was subsequently launched to complete unfinished work and also improve the quality of the code. This effort led to process and organizational changes that enabled more direct control of the change interval. Thus, the average change interval went down dramatically starting about 1991.

However, the managers' concerns were not entirely groundless; although the average interval to implement a change went down, the perceived complexity of the code went up, as judged by the number of different products and the amount of new data added to the system. In other words, developers were making smaller, more interdependent changes and had less control or knowledge of their potential impact. Certain organizational subsystems also became more interdependent, resulting in higher levels of coordination overhead in the company. Thus, when people referred to the code being "difficult to change," it meant not that it necessarily took longer but that it was cognitively more complex, required more extensive interaction with others, and was subject to greater potential error.

RESEARCH METHODOLOGY

The Setting. The telecommunications industry constitutes one of the most tumultuous environments of this decade [13]. One reason behind this extreme volatility is that multiple, diverse forces—including, but not limited to, technology, customers, and competitors—are evolving simultaneously [14]. On the demand side, the environment is becoming much more heterogeneous in terms of both the number and differentiation of customers. As a result, the design and delivery of products and services is now primarily customer-driven as opposed to the supplier-driven model of the past. Similarly, an industry that was once described as a "natural monopoly" now involves competition among rivals who span traditional industry boundaries. Underlying these market forces are changes in the core technology, which reflect the growing power and interconnectedness of software [15].

Many of the key players in this industry produce so-called legacy products, software systems that were designed ten or even twenty years ago and are currently embedded in social and technical infrastructures around the world. Such products require on-going maintenance and support in order to remain functional. More importantly, they must evolve and adapt to new conditions and serve as the underlying basis for new products [16, 17]. A key challenge facing these firms is how to remain innovative and adaptive in the new environment despite their ties with the past [18-20].

Lucent Technologies is one of the leading industry producers with approximately 60% of the U.S. sales and 13% of the global market for telecommunications products and services. The company develops telecommunications products including switching systems, wireless and optical hardware and software, network communications software, integrated business solutions, and communication chips. It employs more than 100,000 people worldwide, approximately 20% of whom work in its R&D unit, Bell Laboratories. Within the research and development community, the average tenure is well over 10 years.

Two factors characterize the culture at Lucent. First, the company has more than one hundred years of experience installing and developing telecommunications equipment. People, therefore, take great pride in the technical excellence of the products, particularly their reliability and robustness. Second, the culture of Lucent is rooted in a long tradition of monopoly. Before trivestiture, the primary customer for equipment was AT&T itself. Thus, competition was not a significant factor in product development until fairly recently.

Lucent's flagship product is a telecommunications switching system, which connects local and long distance calls involving voice and data and is installed in more than 50 countries. A legacy system, it was first released in 1984 and has since evolved to accommodate changes in telecommunications technology and customer requirements while continuing to preserve much of its original functionality. The software to run the switch is on the order of 100 million lines of code and is divided into several subsystems such as call processing, billing, operations, and so on. Products are typically composed of several features that can vary between a few to 10,000 lines of code each. The median feature size is about two hundred lines and has been decreasing over time.

We studied one software subsystem of this flagship product, which is a product in its own right. This product has been under continuous development since 1984 and was first installed in 1987 in Panama City, Florida. It is responsible for services such as directory assistance and credit card calling. The product currently stands at about 2 million lines of code and is in its eighth release. Over its lifetime, about 500 developers have been involved in its development and maintenance.

The Research Approach. Implicit in our approach is a belief that changes in an organization should be reflected in quantitative metrics on its products and processes. [21, 22]. There are also several practical reasons for integrating qualitative and quantitative data: (1) The amount of quantitative data on the software product is immense, increasing the statistical reliability of the qualitative results; (2) The quantitative data are recorded and stored longitudinally, enabling one to tease out cause and effect relationships among the associations reported retrospectively in interviews; (3) The quantitative data are collected automatically and recorded with extremely high precision, thus eliminating systematic biases and increasing the reliability of the results; and (4) Most large software projects keep similar types of databases, facilitating future replication of the study.

The Data. We began by interviewing five members of the organization, all of whom were guaranteed anonymity and confidentiality. (We also agreed to disguise the product name.) The sample was stratified by level of experience (all of the respondents had a minimum of ten years of experience working on this product), product (whether the subject primarily worked on the domestic or international versions of the product), and function (developer, tester, or feature manager).

Each subject was asked in advance of the interview to reflect upon what made the software difficult to change as well as key events in the product and organization's history. The interviews were recorded and later transcribed; each interview lasted approximately two hours. To ensure the validity of the results, we verified information at every step in the data collection. For example, we checked dates of events by referring to awards and annual performance review records. We also routinely fed information back to managers during the analysis.

The interview notes formed the basis for two qualitative data sets. The first contained approximately 300 events that were scored according to how frequently they were mentioned. Events were classified by product (applying to domestic, international, or all versions of the product) and type (strategy, product, organization, or process). The second database abstracted a set of transitions from the interviews. It is these transitions that form the core of this article.

We next generated a number of hypotheses about how the organization might cope with these transitions, which we subsequently tested with data from several product and management tracking databases. Two databases proved particularly important: the version control or product configuration system and the project planning system.

The version control system records each change to the software product so that every version of the system can be reconstructed as needed. It further enables large numbers of people to work on the software simultaneously. The Lucent system records changes done to each software file, the identity of the person performing the change, the date and time of the change, and the number of lines added and deleted. A related database contains textual descriptions of the change implemented—analyzing these abstracts enabled us to classify changes into three distinct types: adaptive (adding new functionality), corrective (repairing faults), and perfective (restructuring the code for ease of maintenance).

The project management database is used to plan and track feature development schedules and deliveries. The system was installed in 1991 and contains records from that time on. (Previous records were kept on paper.) From this database, we extracted information on product attributes, customer delivery dates, and customization information.

RESULTS

In this section we discuss the results of integrating the qualitative and quantitative data. In some cases the quantitative analysis offered quite conclusive support for the information in the interviews. But the data also frequently reflected a number of competing phenomena. In a small number of examples the subject's statements were so uniform that no quantitative confirmation seemed necessary.

The remainder of this section is divided into three "transitions" that emerged from the interviews: (1) changes in the product, (2) changes in the market, and (3) changes in product strategy. Each generated several hypotheses that we subsequently tested with quantitative data.

TRANSITION 1: "The software product code is getting harder to change over time."

A pervasive belief among the developers and managers we interviewed was that adding new features to the product and finding and removing faults in the software were more difficult today than they had ever been. In the interviews, the trend toward more difficult changes was judged to be a "major, major problem." One subject described the process as follows:

> "If a developer wants to change something, …, he has to go in and add a new case. Then he has to think whether that case is changing anything already there. In extreme cases, people end up writing 200 (lines) as 4 (lines) in 50 different modules rather than writing 200 (lines) as a function. … As a result, it takes longer to design and test and is more error prone…"

Code that is difficult to change is problematic for several reasons. First, making changes tends to take longer, which can result in missed market opportunities or a longer response time when a fault is discovered in the field. Second, even if changes are completed rapidly, code that is difficult to change carries more risk of introducing new faults. Third, the coordination costs associated with implementing a change—the need to consult experts, confer with colleagues, respond to questions, etc.—begin to seriously interfere with other productive activities. Finally, code which is difficult to change tends to lose its structural and conceptual integrity over time. Because the code is complex and difficult to understand, a developer may settle for a less than optimal solution— "a kludge"—to avoid changing modules he is unfamiliar with. We identified three hypotheses, which might account for the belief that the code is more difficult to change.

Hypothesis 1.1: Telecommunications features being developed today are more complex than those developed in the past.

The belief that the code got harder to change might reflect the fact that the tasks the developers had to perform to implement a feature grew inherently more difficult over time. This would be the case if telecommunications features were becoming larger or more complicated. We evaluated the extent to which this transition occurred by examining the size of the features as well as the type of changes implemented over time.

Figure 1 studies how feature complexity, as measured by the amount of new code required to develop a given feature, evolved over time. Each green dot in the top panel represents one feature and shows its size in terms of the number of lines of code changed. Structure in this scatterplot is revealed by a family of smooths (weighted running averages), shown as the blue curves. Depending on the amount of smoothing, these curves range from a nearly straight line to some quite wiggly ones representing high levels of sampling variability. The red curve is the "data based choice" among the family members, which represents a tradeoff between the extremes of under and over smoothing [23].

The bottom part of the figure is a SiZer map, which enables visual assessment of which features in the family of smooths are significant, as opposed to being mere artifacts of the sampling variability [24]. Colors in the SiZer map show which features are statistically significant: blue in locations where the corresponding smooth is significantly increasing, red where the smooth is significantly decreasing, purple where the change is not significant, and gray in regions where the data are too sparse for reliable inference.

In this case the color map is nearly all purple, which indicates that the wiggles seen in the smooths are sampling artifacts. In other words, feature size is fairly constant over time. The one exception is a significant increase near the beginning.

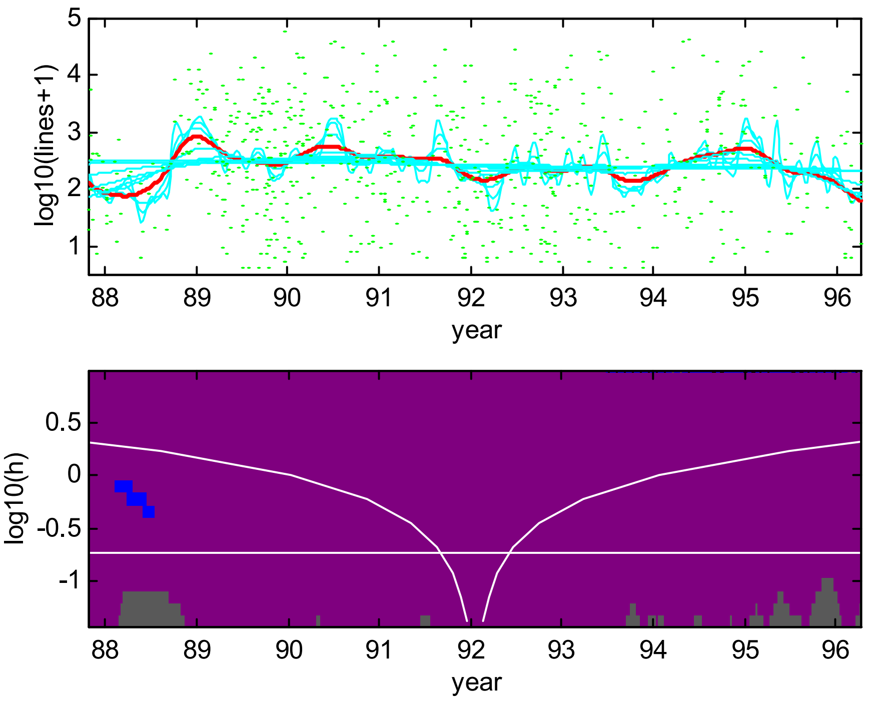

Figure 1: Top shows a scatterplot of log(lines changed) as a function of time, for features, together with a family of smooths. Bottom shows a SiZer map for assessment of which features in the smooths are statistically significant. The figure is represented on a log scale because the distribution is heavily skewed. Features are located at the midpoint of their lifetime.

The evolution of the type of change is illustrated in Figure 2 and shows a trend of increased amounts of new functionality being added up until the late 1980's with a subsequent tapering off and switch to more enhancement type development. We classified software changes or "Modification Requests" into three general categories: adaptive (additions of new functionality), corrective (fault repairs), and perfective (code restructuring). The data are the rate at which new code changes (as measured by Adaptive Modification Requests) are made to the system. The plot is a smooth histogram or more precisely a Gaussian kernel density estimate.

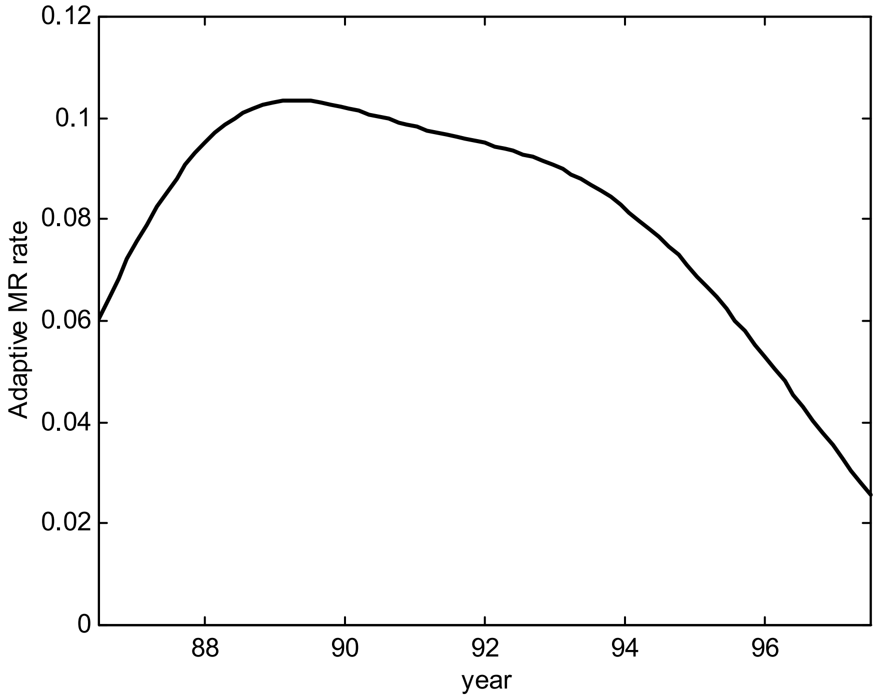

Figure 2: Rate of Adaptive MRs as a function of time.

Hence, this analysis suggests that the developers' sense that the code was becoming increasingly difficult to change did not arise from a change in the type of tasks they were performing. If anything, their tasks were seemingly becoming simpler in the sense that they involved making smaller, more incremental software changes than in the past. Thus we looked for alternative hypotheses that might account for transition one.

Hypothesis 1.2: The high-level product architecture became increasingly unstable or obsolete over time.

The purpose of a product architecture is to simplify the development and maintenance of the product by partitioning it into independent pieces as much as possible. For example, when designing a software product, architects group functionality in such a way that anticipated changes will be easy to implement by putting pieces of code that address related functionalities as physically close together as possible [25].

If the changes to the code did not themselves grow inherently more complex, an alternative hypothesis is that the code was getting harder to change because the architecture of the product was changing or no longer adequately supported the types of changes being implemented. In the first case, the developer's mental model of the product would no longer correspond to reality. Alternatively, the original architecture might become obsolete over time if customers demanded new types of features and functionality that cut across the original partitioning scheme.

Figure 3 shows the number of subsystems that needed to be changed for a given feature over time. Breakdown in the architecture would be reflected as an upward trend in this plot. However, the family of smooths (blue and red curves) appears to actually be trending downward. The downward trend is shown to be statistically significant by the red color in the SiZer plot at the bottom. In other words, these results suggest that the overall product architecture was virtually unchanged, a fact supported by examination of product schematics. Even more surprising, the plots indicate that over time more and more of the work required to implement a feature was concentrated within a given subsystem, suggesting that the original product decomposition was remarkably robust.

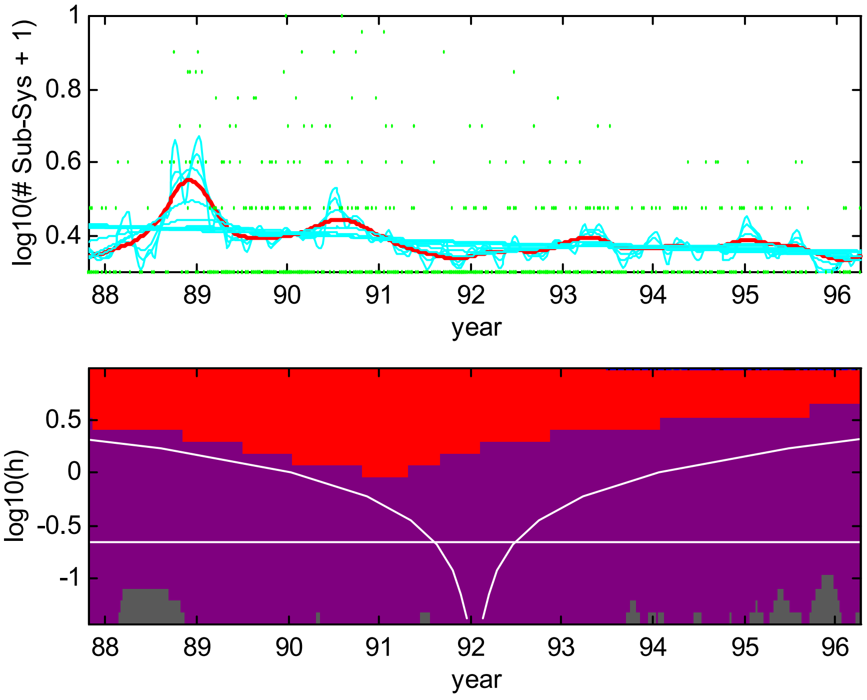

Figure 3: Top shows a scatterplot of log(number of subsystems touched) as a function of time, for features, together with a family of smooths. Bottom shows a SiZer map for assessment of which features in the smooths are statistically significant.

Product subsystems are actually a very coarse, global measure of the degree of stability of the architecture. While the product's high level design may remain quite stable, the code itself can still go through considerable churn within the subsystems. For example, there may be frequent changes in module partitioning schema, interfaces, line divisions, etc. This triggered a third hypothesis described below.

Hypothesis 1.3: The low level product architecture became increasingly unstable over time.

Evidence from the interviews seemed to lend support to this hypothesis. For example, several subjects reported that the number of options and interactions in the code had increased over time:

> "Every time you add a feature, either a new function or a new variation of an existing feature for a new customer, you add in another leg of code. Today, there are probably 100 different features in the product, some with 20-30 options. The interaction between options (within or across features) are particularly difficult to understand. Interactions across features are sometimes not detected until after the code gets to another customer who has a different set of features and options."

Interactions are most germane to this discussion, as they reflect interdependencies between separate pieces of the code. We quantified the degree of interdependence as the number of times two modules within a given subsystem were altered as part of implementing a given change. Figure 4 contains a network-style visualization of the modules in one subsystem. Each point represents a module, and modules are placed close together if they have been altered simultaneously a large number of times. Each of the three plots shows cumulative changes to the subsystem up until 1988, 1989, and 1996, respectively [26].

At the end of 1988, the modules clustered into two main groups, suggesting that developers working on a module in one of the groups were able to largely ignore the code in the other group. In 1989, however, these groups began to merge, a process that continued such that by 1996 there is no longer any evidence of two distinct groups existing. In other words, the interdependencies at the local level were getting more complicated over time.

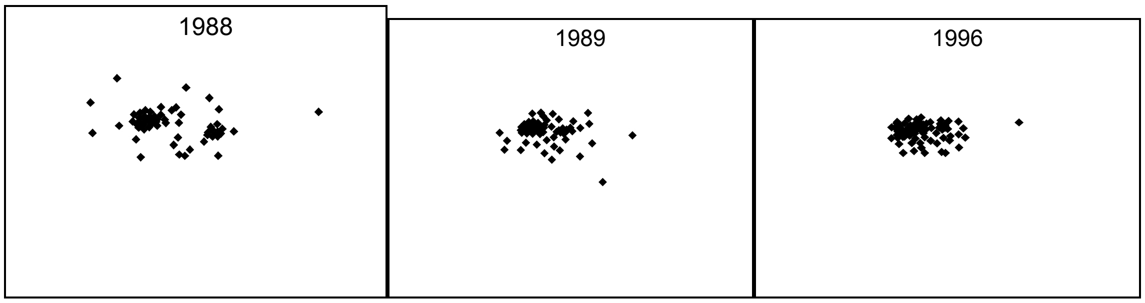

Figure 4: Network views of modules, using data up through 1988, 1989, and 1996 respectively.

In summary, we found some support for the first transition that the code was getting more difficult to change over time. Although difficulty did not appear to be represented in any of the usual aggregate metrics, data describing change processes at lower levels in the software product—where developers actually worked day-to-day—lent support to their statements.

TRANSITION 2: "The target product market changed from a few homogeneous customers to many diverse customers."

A second major theme emerging from the interviews concerned the nature of the marketplace. When originally designed in 1984, the product was intended to be sold as a comprehensive “black box” system, and the primary customer was AT&T. By the 1990’s, partly as a result of deregulation but also reflecting the exploding global telecommunications industry and changes in customer needs, the customer base had expanded to include many different customers. Each of these customers had unique calling requirements, a reflection of both historical differences in hardware and standards as well as cultural differences in telephone usage patterns. Furthermore, most customers purchased specific features such as credit card functionality or call waiting as opposed to the entire system product:

> “At first it was primarily an AT&T system. We were doing lots and lots of really big features for one customer. Then in 1991 or 1992 most of what we were doing was smaller deployments in international to many, many, many customers. This was when we got our best international geography lesson. A little sale to Guam, a little sale to the Philippines; you name it, some little country hither and yon. We got to find out where they all were—it was especially fun during the breakup of the Soviet Union.
>
> It amazed me. There was a time in 1992 where we did 9 deliveries. Some of it was pretty much reselling the stuff we had already done. So in Poland they wanted it in Polish and a little dialing thing was different. But there were also several very large scale new developments.”

One response to this transition might be to produce a series of completely independent products for each customer. But Lucent adopted more of a platform approach—one base of software supporting multiple, customized product versions. This decision reflected several factors. Most important was the legacy nature of the product and the need to maintain functionality and relationships with existing customers. A platform model also enabled the firm to produce a large, complex product with reasonable efficiency. The interviews also suggested that management at Lucent originally under-estimated the importance of the international market and therefore never anticipated the number of versions that would need to be supported on a single platform base.

Below we present three hypotheses about the implications of this transition. These hypotheses predict changes in the product delivery process, reward system, and organizational structure and composition, respectively.

Hypothesis 2.1: The existence of a more differentiated product market induced a shift in the delivery process from an annual delivery cycle to multiple deliveries on demand.

We hypothesized that the pressures for more differentiated products might be manifested in changes to the product delivery process. In particular, we should see an increase in the number of customized product versions being developed and delivered in parallel**.**

Figure 5 shows the number of Software Update (SU) streams over time. SU’s were originally designed as a mechanism for repairing critical faults found in the field. Gradually, and largely informally, they have come to be used for implementing more substantial changes and new features and therefore essentially represent customized streams of functionality. The advantage an SU stream offers over the official delivery process at Lucent is that it allows development upon demand. The disadvantage is that developers must either map the SU functionality addition back into the product base—thereby essentially delivering the feature twice—or support multiple variations of the product.

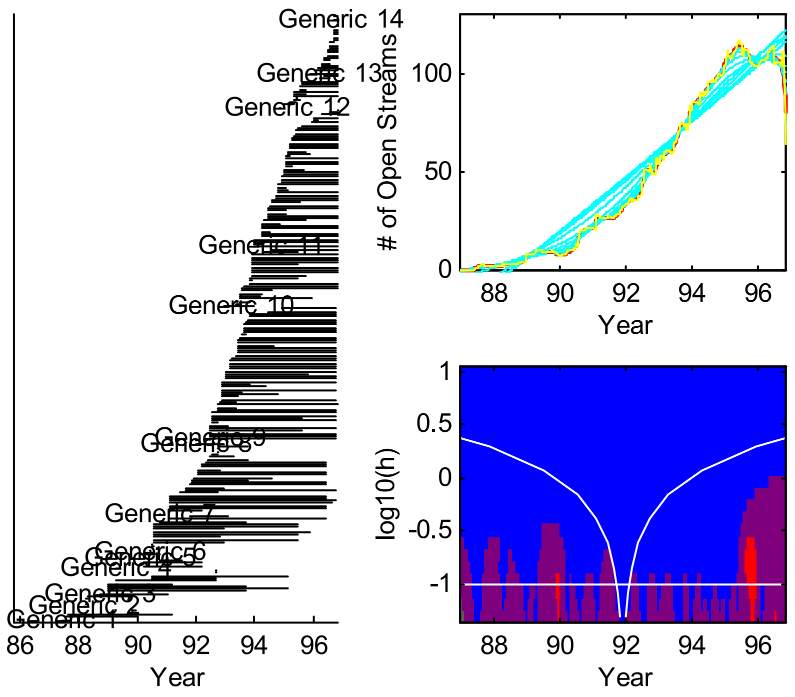

Figure 5: Time lines of software update streams. Each horizontal line corresponds to a stream, and its endpoints encode its open and close dates.

The left hand panel of Figure 5 depicts the open and close dates of each stream as horizontal lines. Clearly, the number of active streams in the mid-1990's was much larger than the analogous quantity a few years earlier. The right hand panel uses the smoothing and SiZer techniques to quantify the upward trend suggested in the left panel. The top part of the right hand panel displays smooth estimates of the number of streams active at a given time. The bottom shows a SiZer map, as in Figure 1. Both support a clear general upward trend through most years with perhaps some leveling off at the end. The predominance of blue areas indicates that the trend is very strongly significant. The few red areas, visible for small amounts of smoothing, show that there was a very slight, but significant, drop in the number of SU streams in late 1989 and a significant drop in late 1995.

Hypothesis 2.2: The existence of a more differentiated product market prompted a shift in the reward system.

We hypothesized that the changes in the competitive and customer market would trigger changes to the reward system with a shift from incentives aligned to optimize product reliability and call processing performance to a focus on shortening the development cycle. Interview subjects described a 1991 executive directive limiting the amount of time a developer "should" take to implement a change. Known informally as the "14 Day Rule," the directive was strictly enforced. If a developer took more than two weeks to implement a code change, his name was circulated daily to the executive management in the firm. Figure 6 shows another SiZer plot displaying the lifetime of, or time necessary to complete work on, code changes. The smooth curves in the top of Figure 6 suggest that since at least 1990, change lifetimes have tended to decrease, on average, showing a clear effect of management efforts to reduce cycle time. The large amount of red on the SiZer plot confirms this.

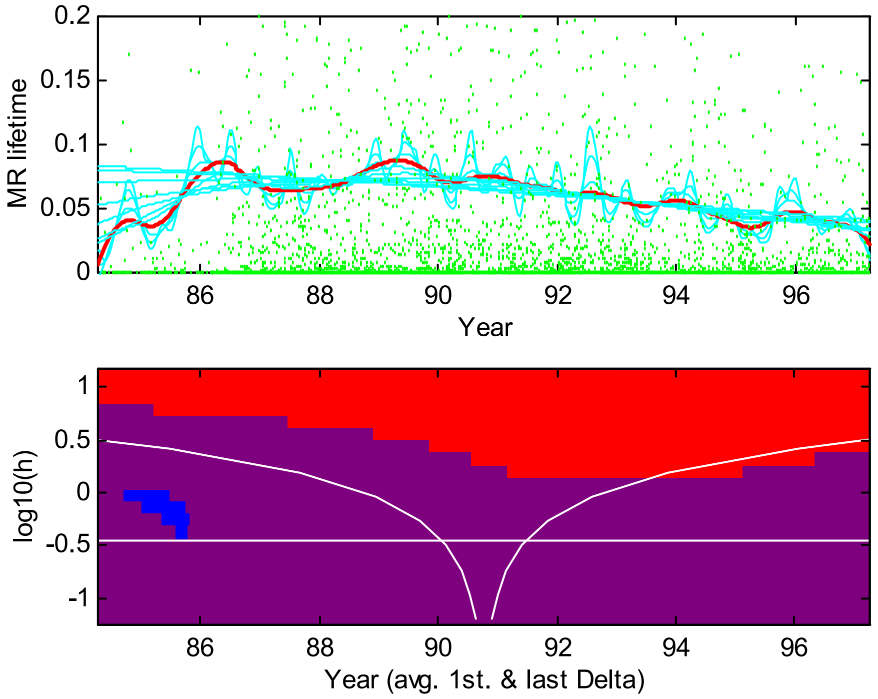

Figure 6: Top shows a scatterplot of MR lifetime as a function of its date (measured by the midpoint between its first and last changes), together with a family of smooths. Bottom shows a SiZer map for assessment of which features in the smooths are statistically significant.

Hypothesis 2.3: The existence of a more differentiated product market prompted changes in the organizational structure and composition.

Finally, we anticipated that there might be changes in the organizational structure, but we had no predictions about the specific form they might take. In order to focus our quantitative analysis, we conducted additional interviews with managers and gained access to archival records such as organizational charts and team reports. These sources concurred that the following changes had occurred.

First were changes in the structure and composition of product teams. Whereas previously product teams were organized by subsystems of code, now they were organized by customer feature requests. The organization also gradually devoted less effort to feature testing and more to field testing after a software update was completed. This was reflected in a one-third decrease in the number of people who were described as feature testers relative to the number of developers.

Lucent also altered the systems and processes it used to coordinate and control product changes. It discontinued its use of a code ownership model, in which a developer was designated to be the “owner” of one or more modules of code. The new model was one of change ownership, in which a developer was authorized to make all the changes necessary to implement a given feature or fix. The advantage of the latter model over the former was that it minimized formal coordination and communication costs whereby developers had to seek out module owners and get their “approval” or “permission” before implementing a change.

Another area of significant change was in individual task definition and assignment. Whereas once developers had single task assignments often extending over several months, they now tended to juggle many tasks simultaneously. This reflected alterations to the delivery model discussed earlier as well as a general downsizing in the organization. We hypothesized that one potential quantitative indicator of this transition would be an increase over time in the number of modules touched per developer.

Figure 7 plots the average yearly number of modules touched per change by a developer. The plus signs display the numbers of module touches as a function of the total number of modules changed by developers working in 1987, while the circles show the same measurements for developers working in 1996. A line between 1987 and 1996 depicts the simple linear regression between number of modules changed by a developer in that year and the total number of changes made by the same developer in that year. There is an apparent increase in the slope of the line (i.e., developers are touching more modules per unit change.)

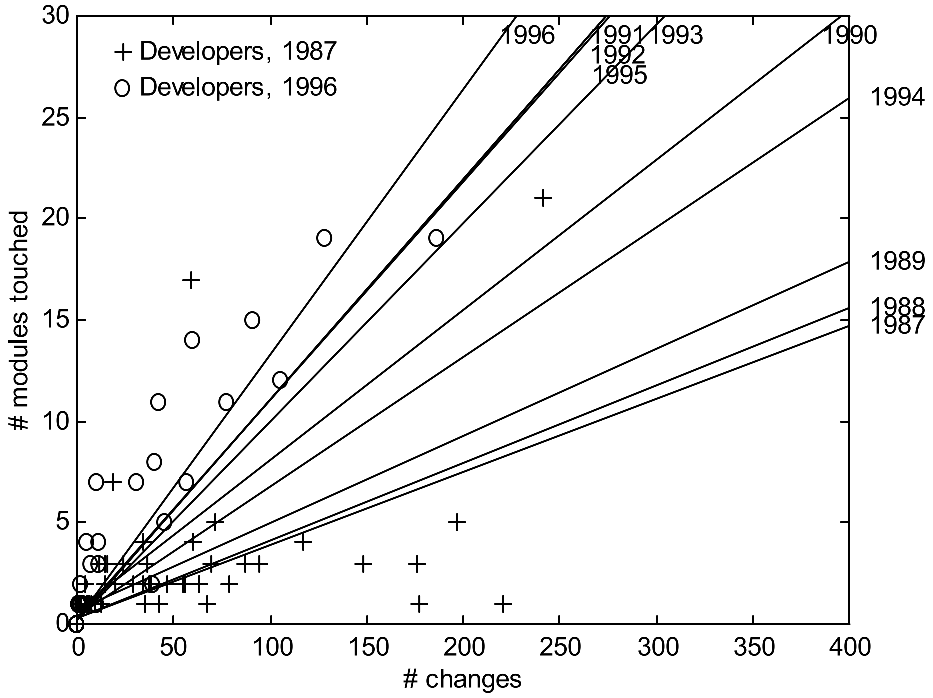

Figure 7: Numbers of modules touched by developers in a year’s time, as a function of their total number of changes. +’s show raw data for developers in 1987, o’s show 1996 data. Lines denote regression relationships for each year.

To assess the significance of this increase, we fit a generalized linear model, which predicted the number of modules a given developer would have to touch in a year as a function of the total number of changes made:

$E(\text{modules}) = \exp\{a + bt + c \log(n)\} = \alpha\, \beta^t\, n^\gamma$,

where E(modules) denotes the expected number of modules changed by that developer in that year, t denotes the number of years between October 1985 through September 1986, n denotes the total number of changes made by that developer in that year, and α, β, and γ are regression coefficients. In the model we treat the number of modules changed as having a gamma distribution with mean given by the above formula.

We estimated β to be 1.0764. This means that the expected number of modules that need to be changed in a year for a given number of total changes is increasing at a rate of 8% a year; in other words, in nine years, a developer's work is spread out over twice as many modules as previously. A 95% confidence interval for β is (1.0554, 1.0978), indicating that the increase in the expected number of modules is statistically significant at the 5% level.

TRANSITION 3: "Lucent's product strategy has evolved over time."

Finally, having established transitions in the product and market environment, we hypothesized that there might have been mediating changes in the company's product strategy.

Hypothesis 3.1: Product strategy shifted from gaining market share, to improving quality, to maximizing profitability.

The interviews suggested that Lucent's approach to its business had in fact undergone a distinct evolution with several well-defined stages. Originally the company focused on gaining market share, particularly in the international arena. As one subject described it, "We said 'yes' to virtually everything and probably underbid on most contracts." This was followed by a brief period devoted to improving product and process quality during the late 1980's. The 1990's have emphasized profitability with the company being somewhat "choosier about what we agree to develop."

Changes in strategy should be reflected in the types of activities people devote time and attention to. In order to test this hypothesis, we considered the rate at which different types of changes were made to the code. The black curves in the top of Figure 8 are kernel density estimates, which show the frequency of changes classified as perfective, as discussed in footnote 3. The metric is defined as the density, over time, of Modification Requests that have terms in their abstract fields (typed in by the developer at the time of performing the work) such as "clean up," "cleanup," and "unneeded." Also shown in Figure 8 are rates of requests with additional words in their abstracts indicating that they were part of specific perfective initiatives, including "streamlining," "lint cleanup,"and "flex names."

There is a very large peak in 1989, and smaller peaks near 1988, 1993, 1995 and 1996. A SiZer map in the bottom of Figure 8 shows that the large peak is strongly significant, the smaller peaks on the right side are somewhat significant, and the peak near 1988 cannot be distinguished from sampling variability. The red part on the right hand side of the SiZer map shows a general downward trend, which suggests less of an investment in enhancing the product.

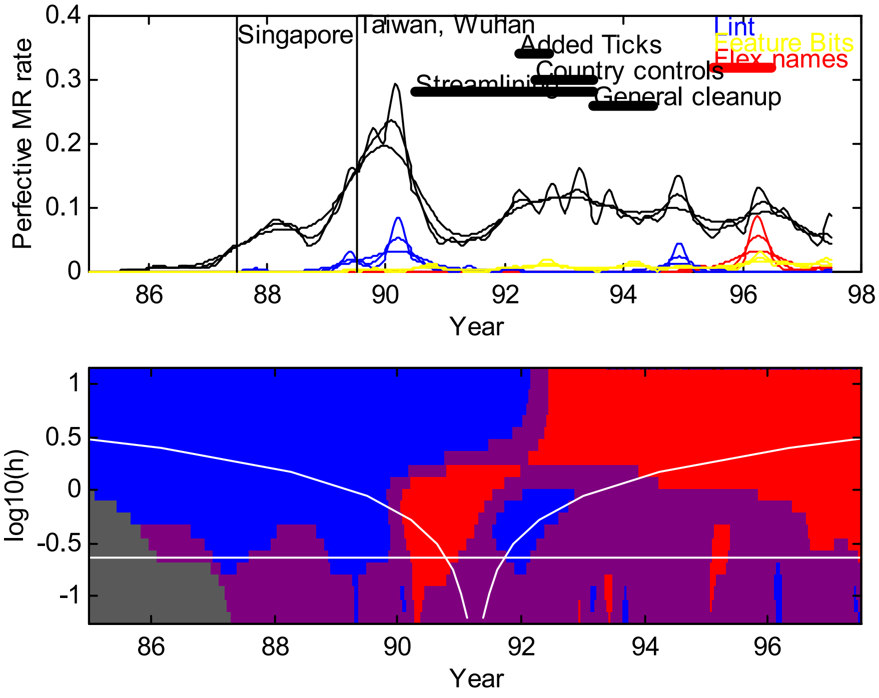

Figure 8: Top shows smooth estimates of the rate of the amount of "perfective" maintenance in time. Colored curves show numbers of change groups whose abstracts contain indications that they were part of specific perfective initiatives, including "Lint" and "Flex Names." These initiatives help to explain the reasons for some of the overall peaks in perfective rate. The bottom plot's SiZer map confirms the statistical significance of at least two substantial peaks.

These patterns fit well with evidence from the interview data. In particular, the large peak represents the "improving quality" phase of the 1980's. It occurred immediately after the release of a large international product, which suffered from major quality problems. The later smaller peaks show the "increasing profitability" phase, where perfective changes were made more incrementally, often in focused spurts as needed and allowed by available resources.

DISCUSSION

This investigation started with a puzzle. On the one hand, managers and developers at Lucent reported to us that the software products they produced were becoming increasingly difficult to modify. Managers were particularly concerned about the implications of this trend for cycle time. Developers tended to focus more on the increased cognitive demands and time required for coordination associated with their work.

Yet we initially found little quantitative evidence supporting these beliefs. For example, the data indicated that product development time was in fact decreasing (see Figure 7). Furthermore, when we conducted more extensive interviews in the firm, we started to hear some subtle contradictions. For example, although developers talked a lot about increased complexity, they also mentioned that the type of development had changed such that they were primarily making smaller incremental changes and enhancements.[1] Similarly, there were reports about high levels of churn in the code, yet inspection of the high-level product architecture indicated that it was virtually unchanged over time. These contradictions suggested a more complicated model was necessary to explain events and beliefs in this organization.

Figure 9 summarizes the conceptual model of change and adaptation, which emerged from our analysis. The model reflects how environmental forces for change confront various inertial factors in the product and subsequently displace change to other areas of the organization.

On the left-hand side, the organization's environment is decomposed into distinct sub contexts: regulatory, competitive, customers, business community, and technological advances. Each of these sub contexts exerts pressure for radical change in the product. Such changes are inhibited, however, by inertial forces such as the size and complexity, high availability requirements, and the fact that the product is embedded in social and technical infrastructures around the world. The right hand side of the model shows alternative coping mechanisms. It presents the organization as a multi-faceted system composed of distinct yet inter-related domains: an organizational structure domain, a coordination and control domain, and a process domain. In effect, radical technological changes are displaced or deflected to these domains.

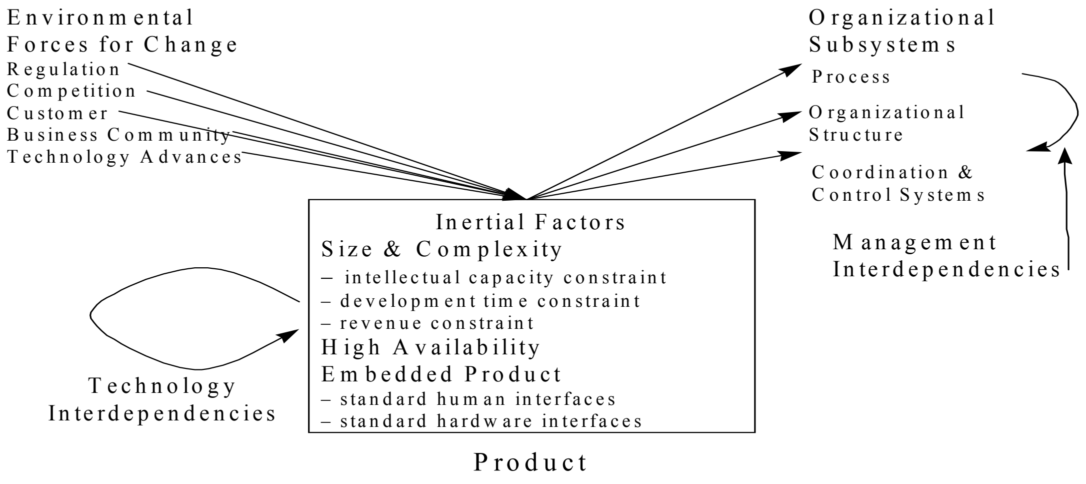

Figure 9: A Model of Change Deflection and Interdependencies

For example, one set of environmental forces facing this organization was deregulation, increased competition, and an expanding customer set. The result was a need for multiple, customized products. One reasonable response would be to create a unique stand-alone product for each customer, but this was not economically feasible

[1] Incremental changes are usually assumed to be smaller in scale and simpler to implement relative to new and more radical types of changes.

given the legacy nature of the system. Instead, the need was addressed by changes in the organizational domains. In particular, the existing process for implementing field fixes evolved into a feature delivery mechanism. This in turn necessitated more field testing capability—triggering subsequent adaptations in the organizational structure and composition. In other words, the technological change of constructing separate products was displaced into a combination of process and organizational changes.

Our model also clarifies the relationship between increasing competition and development time. In order to shorten the development time in a product of this type, the organization changed the unit of development and adapted its systems of coordination and control. In particular, features became smaller, and the code control system evolved from one where developers owned modules of the software to a system of change ownership. (Incentive systems were also aligned to support the new priority of reduced development time.)

But these parallel changes also resulted in more complex interactions in both the product and organization, represented as technical and managerial interdependencies in the model. Low level interdependencies among software modules increased. While formal coordination declined (since developers were no longer required to get "permission" to change modules), informal interdependencies between people also rose.

Such interdependencies have long been known to complicate organizational functioning and coordination with subsequent implications for performance [27]. While Lucent seems to have weathered considerable change in recent years, they continue to encounter disruptions, problems, and operating complexity. This is primarily due to the fact that when environmental change occurs it happens in stages and pockets over time.

Each change will in turn require an adjustment of various organizational subsystems. These different organizational adjustments, which may be simultaneous or sequential, can constrain, contradict, or interfere with one another in unexpected ways. As a result, adjustments made for one change can complicate adjustments made for the others [28]. For example, changes in product architecture in response to new customer needs drive subsequent modification of the delivery process; regulatory changes spur new strategies that aren't necessarily supported by simultaneous trends in product or process. Over time, each change has greater and greater costs in terms of potentially impacting other activities and subsystems and triggering yet more iterative changes. At the extreme, the organization risked becoming paralyzed by its own dynamics, so preoccupied with coping and responding to changes and unintended consequences that it subsequently fails to perform well.

CONCLUSION

This work has significant implications for management theory and practice. In terms of theoretical contribution, the results are consistent with a multi-faceted change model insofar as they depict pressures and responses to change as occurring along multiple fronts. They do not adhere to the punctuated equilibrium model, however, in that the organization is depicted as making small, incremental adjustments over time as opposed to massive reorientations. Most significantly, the model of displaced or deflected change introduces the role of interdependencies into the change process.

The work highlights the relationship between product and organizational complexity and suggests that managers need to pay close attention to both domains. Metrics of increasing complexity (and interdependence) would appear to be particularly important levers for monitoring the change process.

REFERENCES

1. Hannan, M.T. and J. Freeman, *The Population Ecology of Organizations.* American Journal of Sociology, 1977. **82**(pp. 929-964).
2. Hannan, M.T. and J. Freeman, *Structural Inertia and Organizational Change.* American Sociological Review, 1984. **49**(pp. 149-164).
3. Haverman, H., *Between a Rock and a Hard Place: Organizational Change and Performance Under Conditions of Fundamental Environmental Transformation.* Administrative Science Quarterly, 1992. **37**: p. pp. 48-75.
4. Nelson, R.R. and S.G. Winter, *An Evolutionary Theory of Economic Change*. 1982, Cambridge, MA: Harvard Business School Press.
5. Tushman, M.L. and E. Romanelli, eds. *Organizational Evolution: A Metamorphosis Model of Convergence and Reorientation*. Research in Organizational Behavior, ed. L.L. Cummings and B.M. Staw. Vol. 7. 1985, JAI Press: Greenwich. pp. 171-222.
6. Henderson, R.M. and K.B. Clark, *Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms.* Administrative Science Quarterly, 1990. **35**: p. pp. 9-30.
7. Singh, J.V., D.J. Tucker, and A.G. Meinhard, *Institutional Change and Ecological Dynamics*, in *The New Instiutionalism in Organizational Analysis*, W.W. Powell and P.J. DiMaggio, Editors. 1991, Chicago University Press: Chicago.
8. Anderson, P. and M.L. Tushman, *Technological Discontinuities and Dominant Designs: A Cyclical Model of Technological Change.* Administrative Science Quarterly, 1990. **35**: p. pp. 604-633.
9. Milgrom, P. and J. Roberts, *The Economics of Modern Manufacturing: Technology, Strategy and Organization.* The American Economic Review, June 1990. **80**(No. 3): p. pp. 511-528.
10. Gersick, C.J., *Time and Transition in Work Teams: Toward a New Model of Group Development.* Academy of Management Journal, 1988. **31**(No. 1): p. pp. 9-41.
11. Parnas, D.L. *Software Aging*. in *16th International Conference on Software Engineering*. May 1994. Sorrento, Italy.
12. Belady, L.A. and M.M. Lehman, *A Model of Large Program Development.* IBM Systems Journal, 1976. **15**(No. 3): p. pp. 225-252.
13. Davies, A., *Innovation in Large Technical Systems: The Case of Telecommunications.* Industrial and Corporate Change, 1996. **5**(No. 4): p. pp. 1143-1180.
14. Hughes, T.P., *The Evolution of Large Technological Systems*, in *The Social Construction of Technological Systems*, W.E. Bijker, T.P. Hughes, and T.J. Pinch, Editors. 1987, MIT Press: Cambridge, MA.
15. Cusumano, M.A. and R.W. Selby, *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets and Manages People*. 1995, New York: Free Press.
16. Hughes, T.P., *The Dynamics of Technological Change: Salients, Critical Problems, and Industrial Revolutions*, in *Technology and Enterprise in a Historical Perspective*, G. Dosi, R. Gianetti, and P.A. Toninelli, Editors. 1992, Clarendon Press: Oxford.
17. Rosenberg, N., *Technological Interdependence in the American Economy*. 1982, Cambridge, England: Cambridge University Press.
18. Brown, S.L. and K.M. Eisenhardt, *Leveraging Product Innovation: Innocent Traps, Adaptive Organizations and Strategic Evolution*, . May 1995, Stanford University Business School Working Paper: Palo Alto.
19. Lawless, M.W. and P.C. Anderson, *Generational Technological Change: Effects of Innovation and Local Rivalry on Performance.* Academy of Management Journal, 1996. **39**(No. 5): p. pp. 1185-1217.
20. Leonard Barton, D., *Core Capabilities and Core Rigidities.* Strategic Management Journal, Summer 1992: p. pp. 111-125.

21. Cain, B.G. and J.O. Coplien. *A Role-Based Empirical Process Modeling Environment*. in *Second International Conference on Software Process*. 1993. Berlin, Germany.
22. Grinter, R.E., *Understanding Dependencies: A Study of Coordination Challenges in Software Development*, . 1996, University of California, Irvine.
23. Jones, M.C., J.S. Marron, and S.J. Sheather, *A Brief Survey of Bandwidth Selection for Density Estimation.* Journal of the American Statistical Association, 1996. **91**: p. pp. 401-407.
24. Chaudhuri, P. and J.S. Marron, *SiZer for Exploration of Structures in Curves*, . 1998: Unpublished Manuscript, University of North Carolina, Chapel Hill.
25. Simon, H.A., *The Architecture of Complexity.* The American Philosophical Society, December 1962.
26. Eick, S.G., *et al.*, *Does Code Decay? Assessing the Evidence from Change Management Data*, . 1998.
27. Thompson, J.D., *Organizations in Action*. 1967, New York: McGraw Hill.
28. Barnett, W.P. and J. Freeman, *Too Much of a Good Thing? Product Proliferation and Organizational Failure*, . June 1997: Palo Alto, CA.