Resource Dependence in

Product Development Improvement Efforts

Nelson P. Repenning

Department of Operations Management/System Dynamics Group
Sloan School of Management
Massachusetts Institute of Technology
E53-339, 30 Wadsworth St.
Cambridge, MA USA 02142

Phone 617-258-6889; Fax: 617-258-7579; E-Mail: <nelsonr@mit.edu>

December 1999

version 1.0

Support has been provided by the National Science Foundation, grant SBR-9422228, The Center for Innovative Product Development at MIT, the Harley-Davidson Motor Company and the Ford Motor Company. Thanks to Drew Jones for providing useful comment on early versions of the model. Special thanks to Don Kieffer of Harley-Davidson for providing the catalyst to this study.

For more information on the research program that generated this paper, visit our World Wide Web site at http://web.mit.edu/nelsonr/www/.

Abstract

Managers and scholars have increasingly come to recognize the central role that design and engineering play in the overall process of delivering products to the final customer. Given this critical role, it is not surprising that design and engineering processes have increasingly become the focus of improvement and redesign efforts. In this paper one hypothesis for the difficulty of making the transition between existing and new development processes is proposed and analyzed. The starting point for the analysis is the observation that in an organization in which multiple products are being developed, scarce resources must be allocated between competing projects in *different* phases of the development process. The scarcity of resource creates interdependence; the performance of a given project is not independent of the outcomes of other projects. Resource dependence among projects leads to the two main ideas presented in this paper. First, the existence of resource dependence between projects, coupled with locally rational decision making, leads to an undesirable allocation of resources between competing activities; second, this error is self-reinforcing. Strategies for mitigating these undesirable dynamics are also considered.

1. Introduction

System Dynamics (SD) has a long tradition of using formal models to study research and development (R&D) processes. This focus emerged in the early days of the field and has grown to be both an active area of research and a domain in which SD has significantly influenced management practice. Initial work covered a range of topics. For example, in 1978 Roberts wrote "...work [in R&D] can be divided into three areas: (1) the dynamics associated with research and development projects; (2) phenomena associated with the whole of the R&D organization, especially resource allocation among projects or areas; and (3) interrelations between the R&D effort and the total corporation (or government agency)." That same volume (Roberts 1978) contains one or more papers covering each of these three topics. Since the publication of that volume, however, work in SD has increasingly focused on models of specific development projects and less has been written about multi-project management or the relationship between the R&D function and the rest of the enterprise.

The lack of interest in other approaches to understanding the R&D function may be attributed, in part, to the dramatic success of single project models. Single project dynamics were first treated in detail by Roberts (1964). The approach was subsequently used by the Pugh-Roberts consulting company to help the Ingalls shipyard settle a multi-million dollar claim against the United States Navy (Cooper 1980). Since then, Pugh-Roberts has become the industry leader in developing such models for litigation, bidding, and project management. Similarly, the scholarly literature on single project models has grown to include applications to software development (Abdel-Hamid and Madnick 1991), the construction of pulp and paper mills (Homer *et al*. 1993), and the design and development of semi-conductors (Ford and Sterman 1998). Measured in terms of dollar savings, single project models may have influenced management practice more than any other application of the SD method.

While the dynamics surrounding single development projects continue to be an important area of inquiry (see Ford and Sterman 1998 for recent developments), the second element of the research program outlined by Roberts (1978) presents an important and timely opportunity for SD modelers for at least two reasons. First, the dynamics surrounding managing multiple projects and the appropriate structure of the R&D enterprise are increasingly of interest to both product development practitioners and researchers. For example, Sanderson and Uzimeri (1997) study Sony's development of personal stereo systems and show how Sony has been able to target numerous markets at low incremental cost using a *product family* strategy. Wheelwright and Clark (1992, 1995), based on extensive interaction with industry, suggest that senior managers need to focus their attention on their *portfolio* of products rather than individual projects. They further argue that a single project focus can lead to low performance. More recently, Cusumano and Nobeoka (1998) show how Toyota has dispensed with its famous 'heavy weight project manager' system in favor of a multi-project approach. Based on this analysis they suggest that high performers are increasingly eliminating the single project focus in favor of *product platforms* that provide the basis for many projects.

Second, while interest in managing the R&D function (as opposed to specific projects) is on the rise, there are few formal models focused on understanding multi-project development environments. Further, though formal models of design and engineering processes are becoming increasingly popular in engineering and management science (Eppinger *et al.* 1997), few scholars have tried to model multi-project settings using the methods common in industrial engineering and operations research. One of the few papers that does contain such a model, Adler *et al.* (1995), highlights the difficulties of tackling this problem using a traditional management science approach. Adler *et al.* (1995) use a queuing formulation in which the product design and engineering function is modeled as a

stochastic processing network and "...engineering resources are workstations and projects are jobs that flow between workstations." A core assumption of this approach is that work moves but resources do not. Such an approach is appropriate in manufacturing where machines are typically attached to the shop floor and in engineering processes, like testing operations, that rely on dedicated people and machinery. Other resources like engineers, however, are more mobile and fungible. In some cases engineers follow the projects they work on through multiple phases, while in others engineers work on multiple projects in different phases of their development cycles. In contrast to the queuing based approach, the SD method, with its emphasis on operational representation, can comfortably handle the movement of resources. In fact the mobility of resources in R&D environments is recognized in the one paper in the SD literature that deals with a multi-project setting, Weil, Bergan and Roberts (1973). Thus the dynamics surrounding the management of multiple R&D projects are (1) increasingly of interest to both product development scholars and practitioners; and, (2) a research area in which the SD method is uniquely positioned to make important contributions.

To that end, in this paper I hope to accomplish three objectives: 1) re-establish models of multiple projects and the interaction of the R&D function with the rest of the organization as an open, active, and important line of research in SD; (2) present a specific model of a multi-project development system with interesting implications for both research and practice; and, (3) connect two disparate strands in System Dynamics modeling: R&D models and work on the dynamics of successful process improvement (e.g. Sterman, Repenning and Kofman 1997; Repenning and Sterman 1997).

The paper is organized as follows: in the remainder of this section I outline the problem towards which the modeling effort is directed and discuss my main hypothesis. In section two the basic model is developed and analyzed. In section three the possibility of

improving the process through the use of new tools is introduced. Section four contains policy analysis, and section five contains discussion and concluding thoughts.

The Problem: Why Don't People Use the Tools?

Managers and scholars have increasingly come to recognize the central role that design and engineering play in the overall process of delivering products to the final customer (Dertouszos, Lester and Solow 1989). Give this critical role, it should come as no surprise that design and engineering processes are increasingly the focus of improvement and redesign efforts. A variety of authors suggest ways to develop new products faster, cheaper, and with higher quality. Wheelwright and Clark (1992) focus on the role of general managers and Wheelwright and Clark (1995) re-frame their earlier work for more senior leaders. Ulrich and Eppinger (1995) provide a detailed, operational perspective, and Zangwill (1993) lists seven key factors that characterize the processes used by the top performers. None of these are purely theoretical efforts; instead, they are based on the detailed study of many firms and their attempts to improve their development processes. Many companies have, at one time or another, undertaken a serious effort to redesign and improve their product development processes (Repenning 1996, Krahmer and Oliva 1996, and Jones 1997 are a few examples).

Yet, while much has been written about the structure of the ideal development process, less has been said about how to implement these ideas. For example, Wheelwright and Clark (1992), Wheelwright and Clark (1995), and Zangwill (1993) each dedicate only one chapter to the challenges of implementing the structures, processes, and behaviors outlined in the rest of their books. Given the lack of knowledge about how to implement a new product development process, it is not surprising that many improvement efforts fail to produce substantial results. Repenning and Sterman (1997), Repenning (1996), and Jones and Repenning (1997) document cases in which firms invested substantial time, money and energy in developing a new product development process yet saw little benefit from their

efforts. While many people in each of the companies studied agreed that the new process constituted a better way to develop products, the usage of the new process was sporadic at best. In the introduction of their second book, Wheelwright and Clark (1995) discuss the difficulty that many organizations experienced trying to implement the suggestions from their first volume (Wheelwright and Clark 1992). Unfortunately, in many firms the new development process, which may have cost millions of dollars to develop, ends up being little more than a three ring binder sitting on a shelf gathering dust. One engineer summed up his experience in a process redesign effort studied in Repenning and Sterman (1997) by saying "The [new process] is a good one. Some day I'd like to work on a project that actually uses it."

Thus to take full advantage of the substantial progress made in process *design*, a better understanding of the dynamics surrounding process *implementation* is needed. To that end, this study is focused on the *transition problem* in product development: why is it so difficult to implement a new process in product development, and what actions are required to move an organization from its current process to that which it desires?

The Dynamic Hypothesis: Resource Dependence and Tilting

The starting point for my analysis is the observation that in an organization in which multiple products are being developed, scarce resources must be allocated between competing projects in *different* phases of the development process. The scarcity of resources creates interdependence; the performance of a given project is not independent of the outcomes of other projects. Instead, a decision to change the level of resources devoted to a given project affects every other project currently underway and, potentially, every project to come.

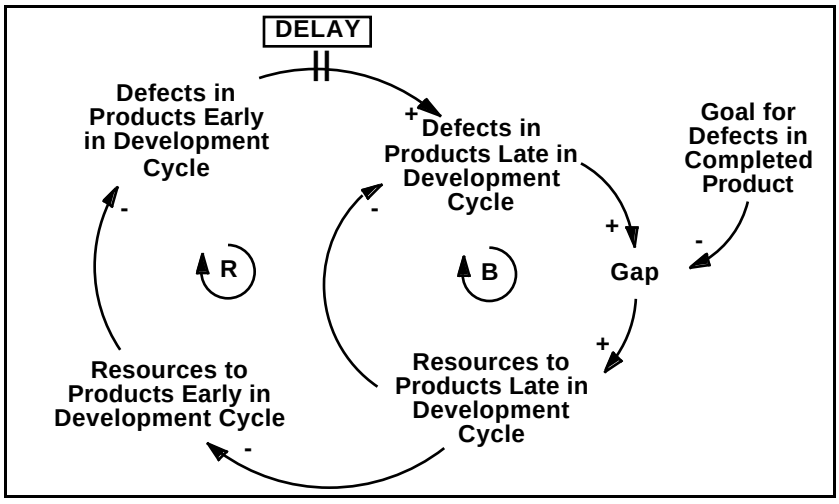


Figure 1

Building on the assumption of resource interdependence, the dynamic hypothesis is shown in figure 1. At the center of the diagram is a balancing loop that represents the resource allocation process. Managers are assumed to have a goal for the quality of each product introduced to the market. As a given development project approaches completion, managers assess the quality of that project via the number of defects remaining in the design. Here a defect includes not only parts of the product that don't work, but also missing features or customer requirements. Upon perceiving a gap, managers respond by allocating more resources to the specific project. Additional resources lead to a reduction in the number of defects, thus closing the gap.

The assumption of resource interdependence is captured by the negative link from *Resources to Products Late in the Development Cycle* to *Resources to Products Early in the Development Cycle*. As the level of resources dedicated to a product early in its development process declines, the rate of defect reduction in that project also declines. These links create a positive feedback loop because the products late in the development cycle are eventually introduced to the market and those formerly in early portions of the development cycle move to the later stages of the process. This positive loop is at the heart

of my analysis and leads to the two main ideas presented in the paper: the positive loop (1) dominates the behavior of many product development systems, and (2) constantly drives the resource allocation among projects to one of two extreme conditions. Either the loop works in a virtuous direction and the organization evolves to the point where most of its resources are dedicated to projects early in their development cycles and little time is spent fixing defects in the later stages of the development process. Alternatively, the loop works in a vicious direction and drives the system to the point where the organization spends all of its collective time on products late in their development cycles. This phenomenon is termed the 'tilting' hypothesis to capture the idea that once the distribution of resources begins to 'tilt' towards the projects closer to their launch into the market, the dominant positive loop amplifies the imbalance to the point where much of the organization's resources are focused on 'fire-fighting' and delivering current projects, and little time is invested in future projects.

The tilting hypothesis has important implications for the implementation of new tools and processes. Up-front activities such as documenting customer requirements and establishing technical feasibility play a critical role in many suggested process designs for product development (Wheelwright and Clark 1992, Ulrich and Eppinger 1995, Zangwill 1993). Organizations stuck in the 'fire fighting' mode allocate most of their resources to fixing problems in products late in their development cycle. Introducing tools that influence the early phases of the development cycle have little value if they do not get used. Prior to taking full advantage of many of these innovations, particularly those that focus on the early phases of the development process, the allocation of resources between current and advanced projects must be redressed. In fact, as will be discussed below, the naive introduction of new tools and processes to an existing organization may actually *worsen* the resource imbalance rather than improve it.

There is, however, a simple policy that greatly increases the organization's robustness to resource tilting. The policy is one example of the class of limiting strategies proposed in Repenning (*in progress*). The essence of limiting strategies is the progressive elimination of the system's ability to compensate for its root cause problems which, in turn, spurs efforts to make fundamental improvements. In this case, a limiting strategy can be implemented in the form of a resource allocation rule. Such a policy does entail important trade-offs between robustness and ideal state performance, but is quite desirable when the possibility of creating improvement via new tools or processes is introduced.

2. A Model of Resource Dependence

2.1 Model Structure

The model represents a simplified product development system in which a new product is introduced to the market every twelve months. Twenty-four months are required to develop a given product, so the organization always works on two projects at once. New development projects are introduced into the system at twelve-month intervals coinciding with the product launch date. The product in the second year of its development cycle (meaning its launch is less than twelve months away) is called the *current* project, and the product in the first year of the development cycle is called the *advanced* project. The timing is shown in figure 2.

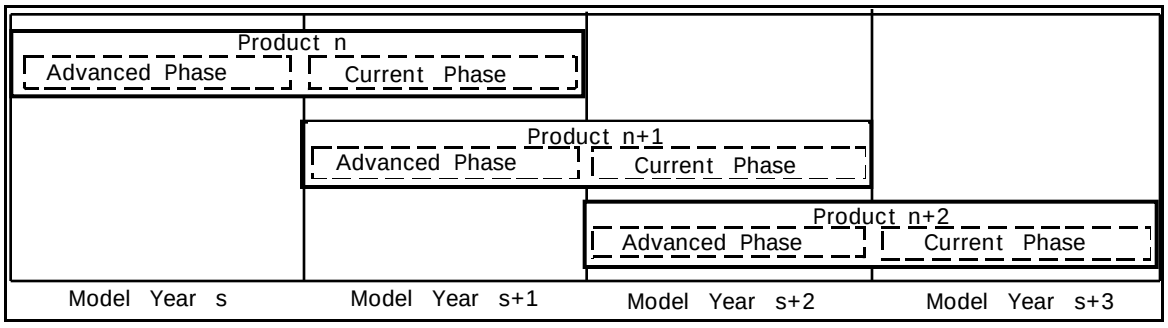


Figure 2

A development project is composed of tasks. Although in real product development systems there are a variety of different activities, in the model all of these are aggregated into one generic task. Similarly, there is only one type of resource needed to accomplish these tasks, engineering hours. These assumptions represent a substantial simplification of any real world process. As will be shown below, however, the model is still capable of generating quite complicated behavior.

There are four key assumptions embedded in the structure of the model. First, the product launch date is fixed. Every twelve months the current product is introduced to the market regardless of its state. There may be, for example, known defects in the design, but the product is still introduced. Such an assumption is appropriate in industries such as automobile and motorcycle manufacturing that have a fixed model year. It is less appropriate in other industries in which the launch date can be delayed until the product achieves a certain quality target. Relaxing the assumption of the fixed launch date remains for future work. Second, any time a development task is completed in either phase, there is some probability that it is done incorrectly. If so, then it can be corrected through re-work, but there is also some chance that re-work is done incorrectly, thus generating more re-work. Third, although engineering head-count is fixed, engineers can adjust for changes in the workload by reducing the number of hours they spend per task.

Fourth, and finally, the probability of doing a task incorrectly is *higher* for current work. The final assumption is critical to all the results developed so it requires additional discussion. The basic idea is that, although I assume the content of each task is identical, a task done in the advanced phase may be *executed* in quite a different way than that same task done in the current phase. Tasks completed early in a development cycle can be done following an established process and in the appropriate sequence. For example, in the advanced phase a particular design task may be accomplished using a CAD tool, and then

tested via simulation. Once the simulation is completed then the design may be converted to a physical manifestation. In contrast, when a task is done in the current phase, due to requirements imposed by the impending launch date, engineers do not have the luxury of testing the design prior to developing a physical prototype. Instead, the two must be done concurrently or the analytical work is skipped altogether. If the task is executed in hardware rather than software, the chance of introducing a defect is higher since testing is more costly and the physical prototyping methods are typically less reliable. Thus, although the resource requirement, in terms of engineering hours, might be the same under both settings, sequencing the tasks leads to a lower defect rate since the benefits of the analytical work precede the development of hardware. Repenning (1998) presents an alternative model of this phenomenon in which tasks are segmented into two different types.

While there is not sufficient space to give an equation by equation description of the model, the key structures and formulations are discussed below (full documentation is contained in the appendix). Readers interested in more detail can contact the author.[1]

Work Accumulations and Flows

The basic stock and flow structure of the model is shown in figure 3. There are four stocks in each phase. The solid flow arrows represent the entry of new tasks into the advanced phase and the exit of the current project to the market. These flows are only active at the annual model year transition. The solid arrow at the top left of the figure represents the introduction of new tasks into the development system which then accumulate in the stock of advanced tasks not completed.

[1] . The model is written using the VENSIM software produced by Ventanna Systems Inc. The model can be downloaded from <http://web.mit.edu/nelsonr/www>

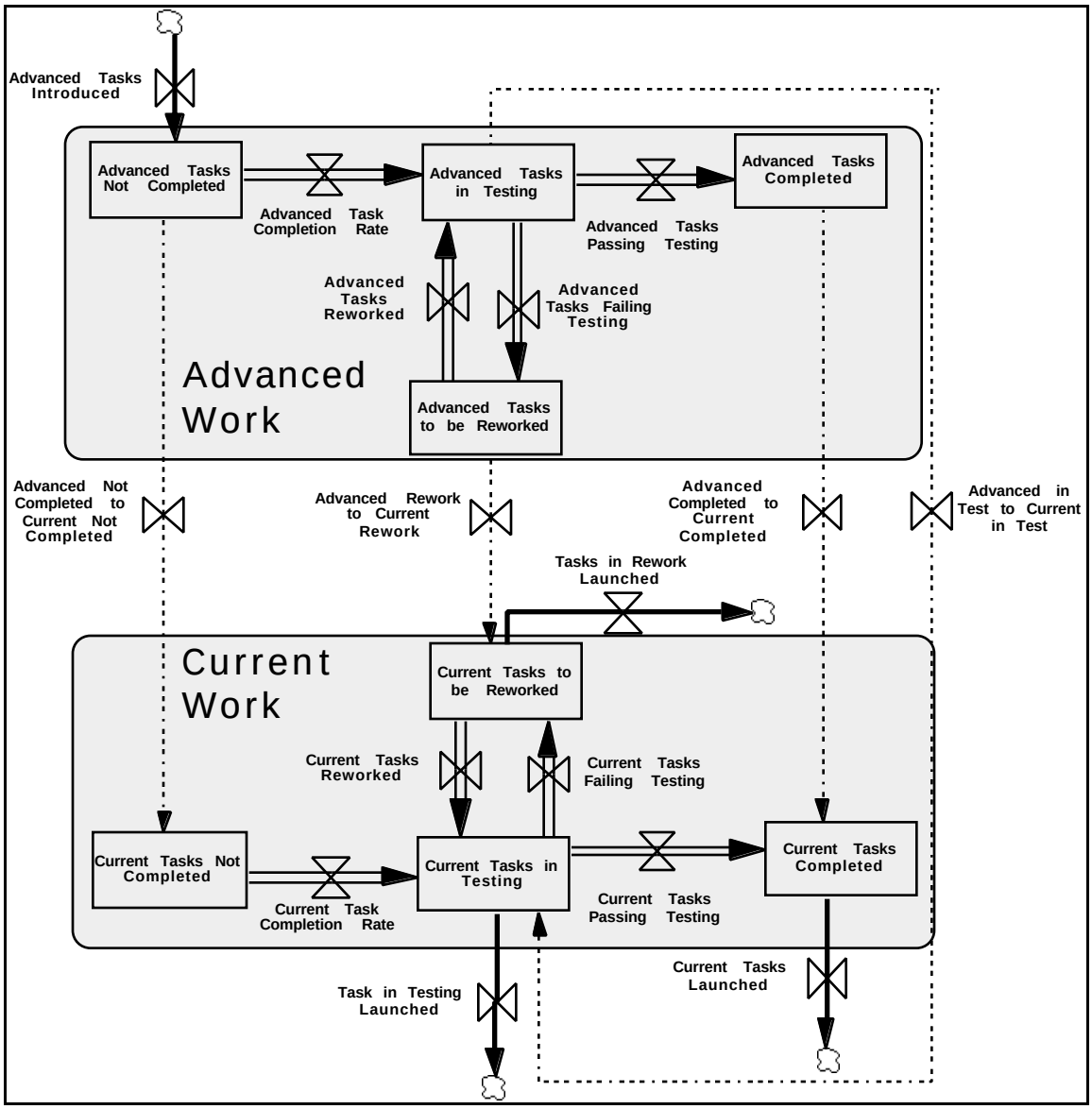


Figure 3: Arrows with valve symbols represent the flow of tasks in the system. Solid arrows represent the entry and exit of tasks to and from the product development process. Flows with double lines represent the continuous flow of work within a given model year. Dotted line flows represent the annual transition of advanced work to current work.

The standard flow symbols (those that are hollow) represent the continuous movement of work between the stocks during the given phase. The stock of advanced tasks not completed is drained by the advanced task completion rate. Once completed, tasks accumulate in the stock of advanced tasks in testing. If a task passes the test, meaning it was done correctly and is not defective, it flows to the stock of advanced tasks completed.

Tasks failing testing flow to the stock of advanced tasks requiring rework. Once reworked, tasks flow back to the stock of tasks awaiting testing.

The dotted flow arrows represent the annual transition of advanced work to current work. Each stock of advanced tasks has such an outflow and each stock of current tasks has such an inflow. At the transition between model years, the entire contents of each advanced work stock is transferred to the corresponding stock of current work in one time step.

The task flow in the current work phase is identical to that in the advanced phase. Completion of new tasks drains the stock of current tasks not completed and increases the stock of current tasks awaiting testing. Tasks either pass the test, in which case they flow to the stock of current tasks completed, or fail and move to the stock of current tasks awaiting rework. Once reworked, tasks return to the stock of those awaiting testing. At the end of the model year all current tasks completed are introduced to the market, regardless of their state. These outflows are represented by the solid black arrows. The measure of performance used to analyze the system is the quality of the current project at its launch date and is measured by the fraction of its constituent tasks that are defective.

Defects

The probability of doing a task incorrectly is determined by a slightly modified co-flow structure (Richardson and Pugh 1981, Sterman *forthcoming*). The structure is shown in figure 4. Each time a task is completed or re-worked there is some probability it is done incorrectly and is defective. The new task and rework completion flows are represented by the light gray flow icons. The number of defects introduced through new work and re-work is determined by multiplying the flow of tasks by the probability that a given task is done incorrectly. Once a task is completed or re-worked, its associated defect flows into the stock of defects in advanced testing. Dividing this stock by the total number of tasks currently in testing determines the fraction of defective tasks in testing, which, in turn,

determines the rate of tasks passing and failing. Tasks that fail testing remove one defect from the stock of defects in testing while tasks that pass testing remove no defects from the stock. In real systems there are both type I and type II errors, but in the model testing is assumed to be perfectly accurate. If the launch date is reached and some tasks remain untested, they are still moved to the current phase. An identical structure also exists to track the number of defects in the current project.

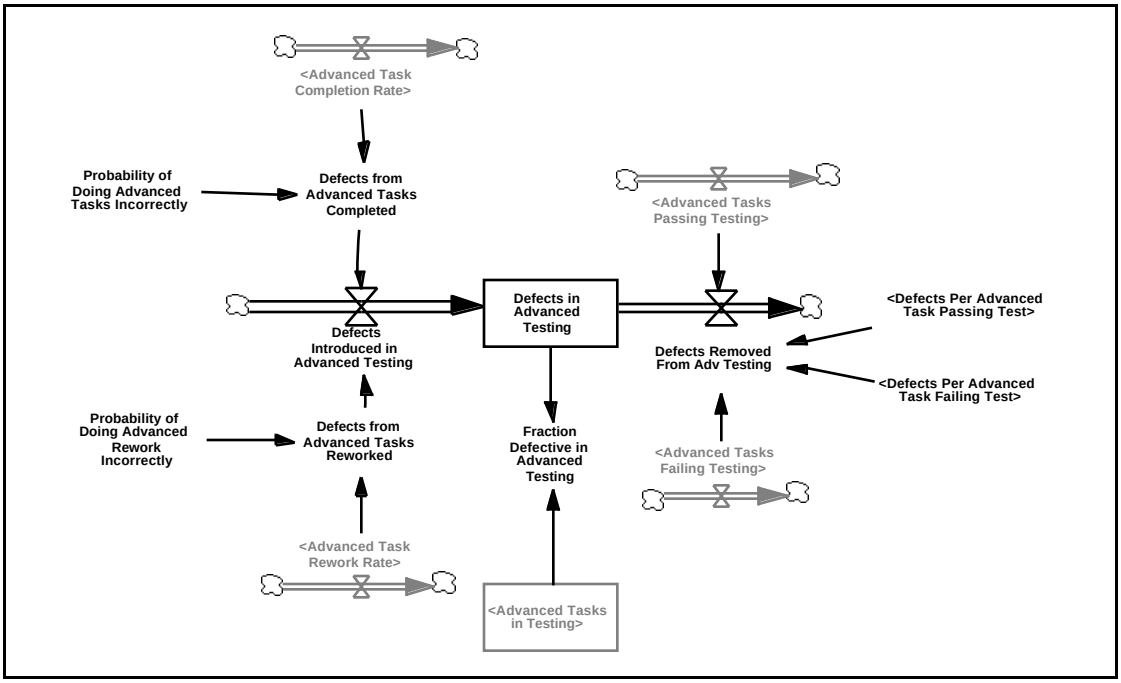


Figure 4

Allocation of Capacity

The critical decision function in the model is the allocation of engineering capacity between the four types of development work: 1) new current tasks (CW); 2) current re-work (CR); 3) new advanced tasks (AW); and 4) advanced re-work (AR). The allocation of effort between these categories is determined via a two step process. First, an desired completion rate (DCR_i, i {CW, CR, AW, AR}) is determined for each type of work. Following the formulation often used in single project models (e.g. Abdel-Hamid and Madnick 1991,

Roberts 1964), the desired rate of work completion is calculated by dividing the stock of work to be completed by the number of months remaining until product launch. Thus

$$DCR_i = \text{Work Remaining}/\#\text{ of months to product launch} \quad (1)$$

The formulation represents a decision process in which engineers look at the work to be completed and allocate it evenly over the time remaining.

In the simplest version of this formulation, engineers take no account of the chance that mistakes will be made thus creating additional rework. This assumption is somewhat unrealistic in product development since engineers often build prototypes in advance of the product's launch for the sole purpose of finding mistakes. To capture the idea that *some* rework may be anticipated, the four desired rates are formulated identically except for the rate of current work completion. In the formulation for the desired rate of current work completion the denominator, instead of being the number of months until launch, is the number of months until the prototype deadline (assumed to be six months prior to the product launch). Thus, in a simple way, engineers are assumed to factor in the possibility of making mistakes and plan their workload accordingly.

Second, once the desired completion rates are determined, the actual rates are calculated. Two rules are used: 1) new work takes priority over re-work; and, 2) current work takes priority over advanced work. The actual rate of current work completion (ACWR) is the minimum of the maximum development capacity (MDC) and the desired completion rate for current work (DCR_{CW}).

$$ACWR = \text{Min}(MDC, DCR_{CW}) \quad (2)$$

MDC is the maximum rate at which development work can be completed, and is equal to the total number of available engineer hours, AEH, divided by the minimum number of hours required per task (MHT):

$$MDC = AEH/MHT \quad (3)$$

MDC does not measure the rate at which development work can be done properly, but the rate at which work can be completed when engineers are working as quickly as possible and skipping all steps that do not directly contribute to finishing the design. Such skipped steps might include documenting work and testing new designs.

If, after completing the current new work, there is capacity remaining, then it is allocated to current rework. The actual rate of current re-work (ACRR) is equal to minimum of capacity remaining, MDC-ACWR, and the desired rate of current re-work (DCR_{CR}):

$$ACRR=Min(MDC-ACWR,DCR_{CR}) \quad (4)$$

Capacity allocated to the advanced phase is determined in a similar fashion. First, the capacity that *can* be allocated to the advanced phase (CAT) is determined by subtracting the capacity allocated to current tasks from the normal development capacity (NDC). NDC is the rate at which engineers can complete tasks when they are following the prescribed development process. Thus,

$$CAT=Max(NDC-(ACWR+ACCR),0) \quad (5)$$

The maximum function is important because the rate of work completion can exceed the normal development capacity. If the organization has more work to do than is possible under normal operations, engineers are assumed to complete that work by cutting corners and skipping development steps. If this is the case MDC is greater than NDC. If CAT is greater than zero, then the remaining development capacity is allocated to advanced work and then advanced re-work. Normal development capacity is equal to the available engineer hours (AEH) divided by the normal hours required per engineering tasks (NHT):

$$NDC=AEH/NHT \quad (6)$$

This formulation embodies an important assumption: advanced work does not get completed unless the current work rate is less than the normal development capacity. If the normal development capacity is not sufficient to complete all the outstanding current tasks,

then engineers will reduce their time per task to reach the required work rate. This does not happen, however, for advanced work. If the organization is working beyond normal capacity, advanced work is simply not done.

2.2 Model Behavior

This section describes the model's steady state behavior. In all runs the base task introduction rate is 1,500 per year.[2] All tasks are introduced in one time step at the model year transition.

Base Case

Figures 5 and 6 show the base case behavior for advanced and current tasks.

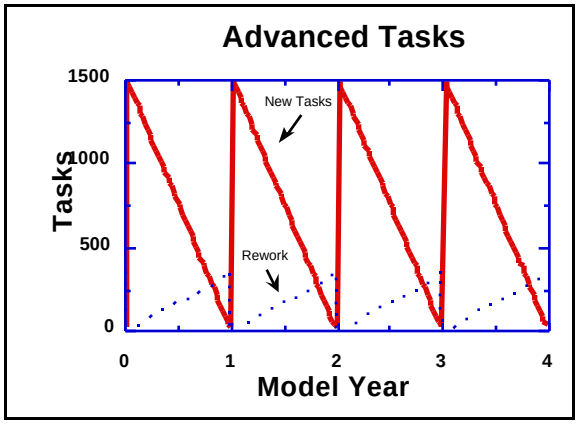


Figure 5

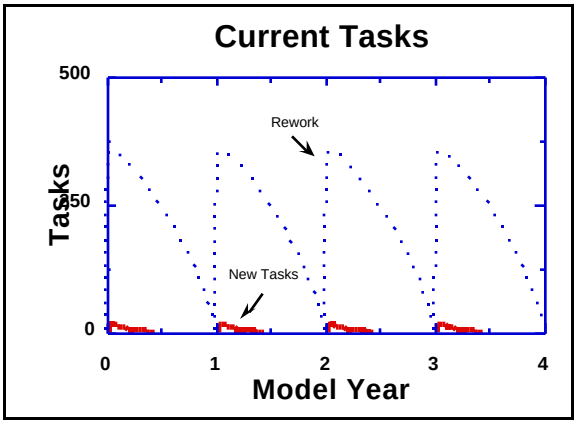


Figure 6

At each twelve-month interval 1500 new tasks, constituting a new development project, are introduced. Advanced work is completed at a constant rate throughout the year so the stock of advanced tasks to be completed declines approximately linearly. Most new tasks are completed in the advanced phase and only a small number (less than 50) are sent to the current phase uncompleted. The level of re-work to be completed grows during the advanced phase as tasks done incorrectly are discovered through testing.

Once the project reaches the current phase, the small number of tasks received uncompleted are quickly finished. Approximately 300 tasks requiring rework are also received from the advanced phase. The rate of re-work completion rises through the year as the product launch date approaches and the organization increasingly focuses on current work in the hope of fixing all the defective tasks before the product's launch.

Figure 7 shows the average defect level in the advanced and current phases. The defect fraction in advanced work starts at 40% (the probability of doing a new advanced task incorrectly) and declines for the first 9 months of the year. It then increases slightly as the rate of advanced re-work completion drops due to lack of resources. The fraction of defective tasks in the current phase drops dramatically throughout the year, due to multiple iterations through the rework cycle, falling from 30% to less than 5% and improving the quality of the finished product substantially.

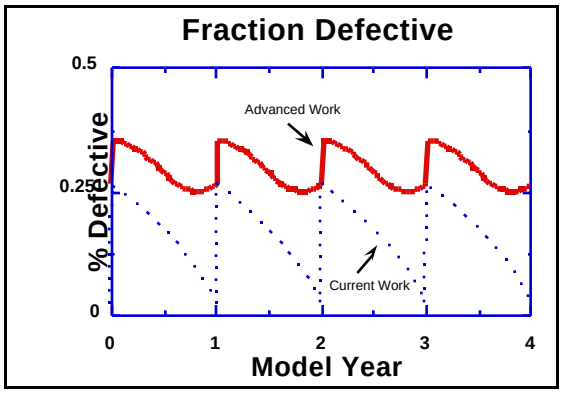


Figure 7

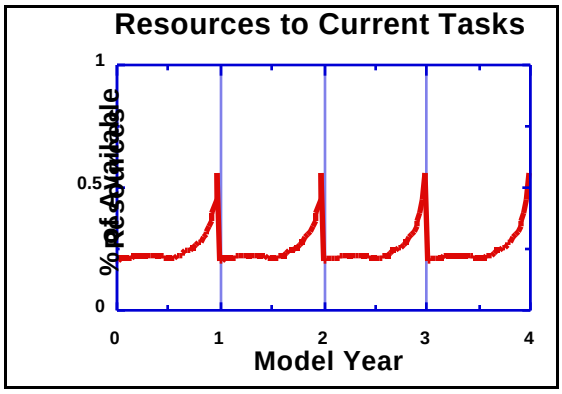


Figure 8

Figure 8 shows the allocation of resources between the two phases. At the beginning of the year approximately 25% of the total capacity is allocated to the current project. The allocation remains approximately constant for the first half of the year and then increases as the product launch date approaches. The shift in the allocation of resources leads to an increasing rate of defect reduction in the current phase (figure 7).

---

[2] . The model is simulated using the Euler integration method and runs in months with a time step of .25.

The base case depicts a development organization that does the vast majority of its initial development work while the product is in the advanced stage. However, due to low process capability, a significant fraction of that work is done incorrectly. During the current phase many of those defects are corrected through re-work, thus allowing the system to produce a product of relatively high quality.

Transient Behavior

Consistent with standard practice (e.g. Forrester and Senge 1980) a wide range of tests have been conducted to understand the model's behavior. In the interest of space, most of these will not be presented. A particularly instructive test of the model's transient behavior is, however, to introduce a one time increase in the amount of development work. In each of the runs presented in this section, the pulse increase takes place in model year one. Figure 9 shows the effect of different pulse sizes on the average fraction of work that is allocated to current programs and figure 10 shows the impact of the additional work load on the quality of the product introduced to the market.

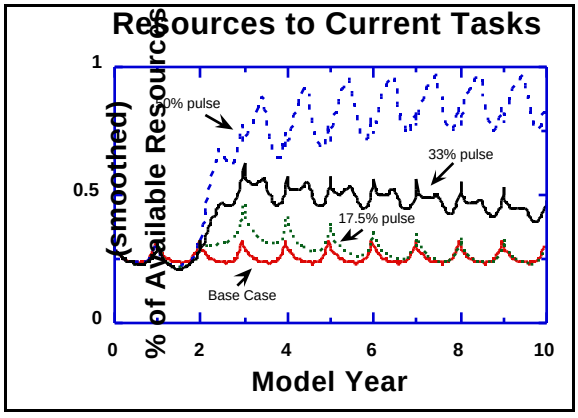


Figure 9

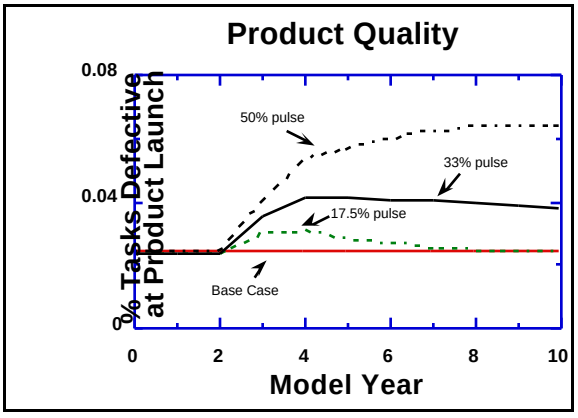


Figure 10

In the base case, approximately 25% of the total resources are allocated to the current phase. A one time increase in the work rate equal to a sixth of the base case value causes

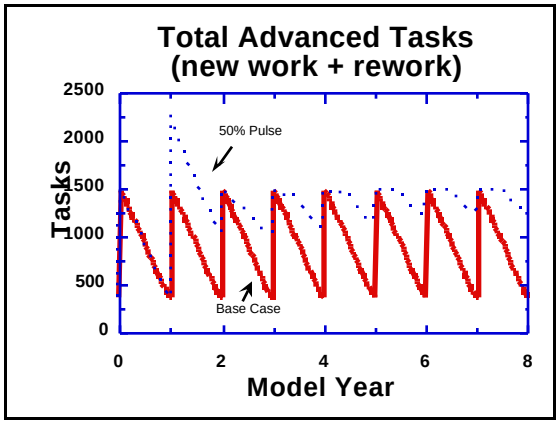
---

In each case, it is run for one hundred and eighty months to eliminate transients.

the fraction of resources allocated to current work to immediately grow and then slowly return to the initial steady state behavior. Similarly, a pulse equal to a third of the base value causes a greater increase in the fraction of work dedicated to the current phase and then a very slow recovery to steady state. With the larger pulse, product quality declines more and recovers more slowly. Thus, even though the pulse happens in only one model year, its impact can persist for a substantial amount of time. The larger the pulse, the more time is required to recover and the larger the impact on product quality.

If the pulse is sufficiently large, however, the system never returns to the base case behavior. When the pulse is equal to 50% of the base case, the fraction of capacity dedicated to current work immediately jumps and does not return to the pre-pulse behavior. Instead, the fraction of current work migrates towards 100%. Once resources are 'tilted' towards current work, the initial change is self-reinforcing and the system does not recover to its pre-pulse behavior.

Figures 11 and 12 help explain why this is the case.



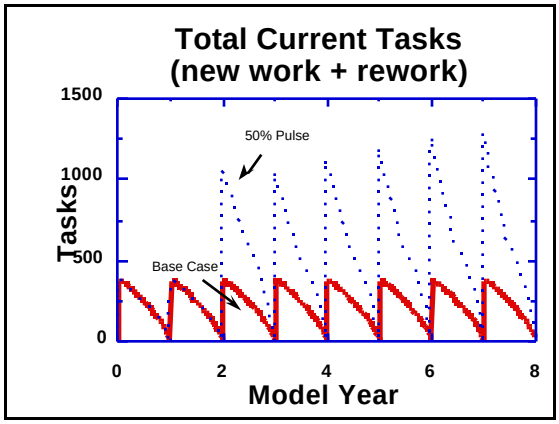


Figure 11 **Figure 12**

After a 50% pulse there are now 2250 changes to be completed rather than 1500 in the advanced phase. In the first year following the pulse, approximately the same number of new advanced tasks are completed, and there is no change in the completion rate of current

tasks. At the end of model year one, due to the increase in workload, 500 additional tasks move to the current stage uncompleted. In model year two, since there are now 500 additional tasks to complete in the current phase, fewer tasks are completed in the advanced phase. In subsequent years, as the advanced work completion rate declines, more work enters the current stage uncompleted and the effect of the initial pulse is amplified rather than attenuated.

Figure 13 shows the source of this persistence. The increase in the workload means that, in model year two, more new tasks are moved to the current phase uncompleted. Since the probability of introducing a defect is higher in the current phase, additional new tasks done in the current phase lead to a higher defect fraction in the current project (figure 13) thus creating more rework (figure 14). The extra rework requires additional resources that would not have been needed *had those tasks been done in the advanced phase*. The additional resource requirement in the current phase reduces the rate of task completion in the advanced phase which, in turn, causes the situation to be even worse in subsequent model years. In this case the pulse is large enough to cause the dominant positive feedback loop to work in an undesirable direction and drive the system to a state in which little work is done in the advanced phase.

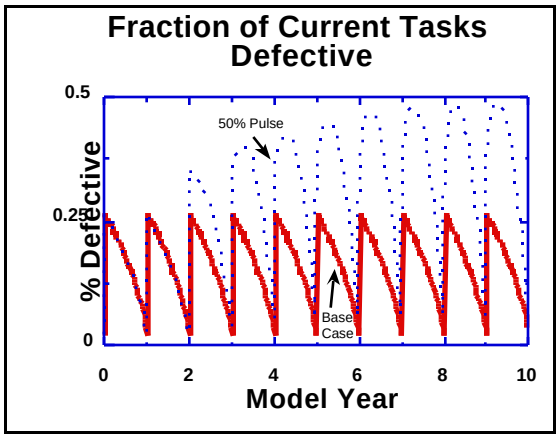


Figure 13

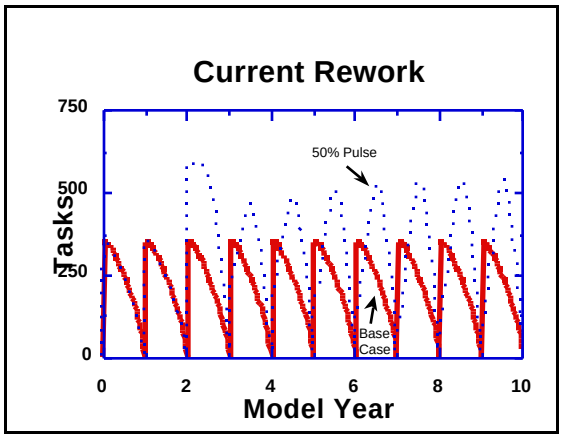


Figure 14

The pulse test shows an important property of the system. Although the pulse occurs only once, if it is large enough, it can push the system into a new steady state behavior in which the system's performance is permanently degraded. This key feature is due to resource dependence and the dominance of the positive tilting loop. Without resource dependence between the advanced and current phases, the impact of the pulse is only felt until the extra changes are completed and the product is introduced, then the system is re-set and returns to its steady state behavior. In this case, however, because additional work done in the current phase reduces the resources allocated to the advanced phase, the system does not re-set itself. Instead the pulse has a permanent, negative effect on the system's performance.

The existence of multiple steady state behavior modes (or 'equilibria') in the model leads to two important and related insights. First, it suggests that capacity planning is critically important in a product development system. Although the test just shown was a pulse, it should be fairly obvious that a permanent increase in workload (a step) of even a modest size is sufficient to cause the system to 'tilt' and dramatically degrades the system's ability to produce a quality product. The deleterious effects of overloading a product development system have been noted by a number of authors. Wheelwright and Clark (1992) describe the 'canary cage' problem: too many birds in a cage reduce the available resources, food, water and space, and none of the animals thrive. Similarly, in product development, they argue, too many development projects lead to a poor allocation of resources and mediocre products. The resource allocation rule implicit in their model allocates resources evenly across all projects. The formulation presented here is an alternative since resources are allocated to projects closer to their launch date. It also expands the 'canary cage' logic by introducing tilting. Increasing the number of projects in the system has two impacts: it reduces the resources allocated to any one project, and it changes the distribution of resources, favoring those closer to launch. Thus, not only are there too few resources, but

overloading, via the tilting loop, causes those resources to be allocated in an undesirable way.

Second, and more importantly, the analysis suggests that the system can operate in an undesirable mode, *even* when the resources and the workload are well matched. This possibility has not been discussed in the existing literature. The pulse test shows that a single shock can move the system to a state from which it will not return. Thus managers need to be careful to make resource allocation decisions based on the *current* state of their development processes, not the ideal state.

3. Introducing Process Improvement

The model provides one explanation for why organizations often operate in a high stress, high rework mode and allocate, few, if any, resources to up-front development activities. In addition, although the product is often reworked multiple times it fails to achieve many of the objectives set at the outset of the particular development project. A common response to such a mode of operation is to introduce some modifications to the engineering and design process. Such changes include new tools to reduce the probability of introducing a defect and to speed the re-work cycle, and specification of a modified sequence of development steps including appropriate checkpoints and reviews. Numerous firms have tried to make such process improvements in product development.

Unfortunately, as discussed in the introductory section, the existing empirical literature suggests that specifying a better process is not sufficient to insure that it is used effectively. The failure of such *process improvement* initiatives has recently become of a topic of interest in the SD literature (e.g. Sterman, Repenning, and Kofman 1997; Repenning and Sterman 1997; Krahmer and Oliva 1997; Oliva , Rockart and Sterman 1998; Repenning

1998a,1998b) the SD literature now contains a number of models that highlight the difficulties associated with trying to improve and change one or more processes within an organization. A common theme highlighted in all of these studies is the failure of managers to understand the dynamic implications of process change, and, as a consequence, programs often fail due to unintended side effects of well-intentioned actions. Such a finding should be of little surprise to SD modelers since process improvement and change are inherently dynamic, disequilibrium phenomena and the limited ability of humans to manage such dynamics lies at the foundation of the field (Forrester 1961).

A limitation of each of these studies in the present context is that they either focus specifically on manufacturing environments or a general improvement effort in a generic context. In each case, the models do not contain detail that is specific to the product development process (an important exception is that all of these recognize that improvement in product development may take longer). Thus, while one should expect that managers in product development settings are likely to fall prey to many of the failure modes identified in the other studies of process improvement, the domain-specific nature of these failure modes remain to be identified and analyzed. To that end, in this section, additional structure is added to the model to capture the introduction of new tools and processes.

Introducing a new tool or process is assumed to have two impacts on the system. First, the new technology reduces the probability of introducing a defect in all types of work. The full benefit of the tool is, however, not realized until the organization has developed experience using it; there is a delay between beginning to use a new tool and receiving its full benefit. Second, using the new technology makes completing each development task take *longer*. The increase in task duration captures the fact that new tools and processes generally require time to be learned and have additional documentation and reporting requirements. The model is parameterized in such a manner that the reduction in the

probability of introducing a defect more than compensates for the increase in task duration. Thus, if the firm can gain the necessary experience, adopting the new methodology increases the total capability of the development system.

3.1 Model Structure

The introduction of a process improvement initiative is captured in a simple way. First, there is assumed to be a threshold allocation of resources to the advanced phase required to achieve the full benefit of the new tools. The assumption is based on the fact that many of the current innovations in product development are focused on the up-front portion of the development process and have little impact if they are used late in the development cycle. For the purpose of this model the threshold is assumed to be 75% (the initial value in the base case). The organization's current adherence to the methodology (CAM) is then calculated by dividing the current fraction of resources (FRA) dedicated to the advanced phase by the threshold value (FRA*). Thus:

$$CAM=MIN(1,FRA/FRA^*) \quad (7)$$

The rate of defect introduction is then determined by the organization's average adherence to the methodology (AAM). AAM is an average of the current adherence weighted by the number of changes that have been done following the methodology in the past three model years. A co-flow structure, shown in figure 15, is used to calculate this weighted average.

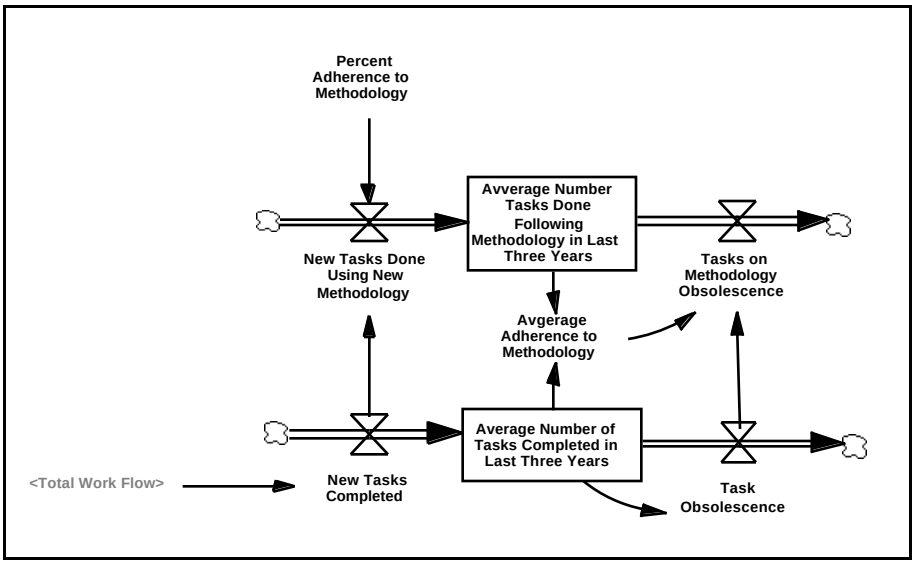


Figure 15

The impact of the new methodology on capacity is captured by changing the equation for normal development capacity (NDC). Once the new technology is introduced, NDC equals:

$$NDC=AEH/DHT \quad (8)$$

DHT represents the time required to complete a task while following the methodology.

3.2 Model Behavior

The new development methodology is introduced in the first model year. Figures 16 and 17 show the consequence of introducing a new methodology under three different assumptions about the necessary change in normal development capacity. In the first, the methodology requires no more time than the current operating procedure and thus normal capacity is not reduced–NHT=DHT. In the second, the methodology increases the required time per task by one sixth; in the third, the time required per task is increased by one third.

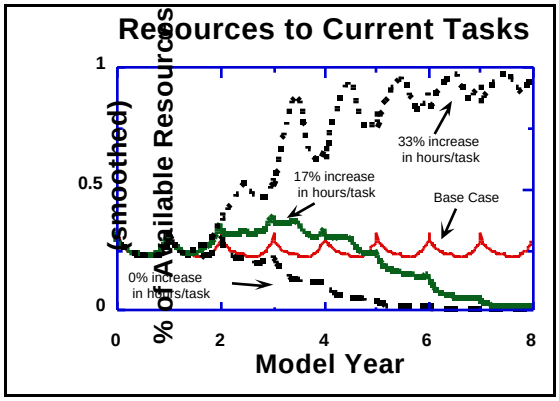


Figure 16

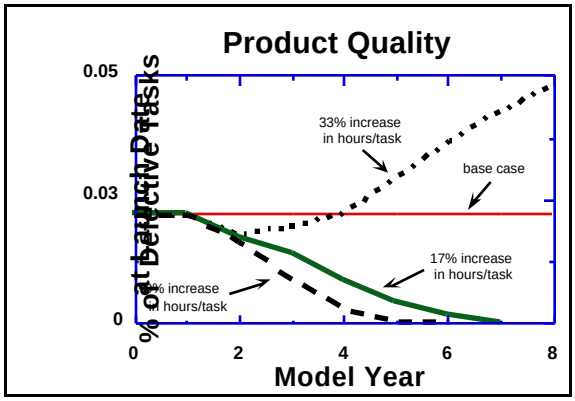


Figure 17

If the methodology entails no sacrifice in capacity (case #1), then the fraction of work devoted to current programs decreases immediately after the methodology is introduced. As more work is shifted to the advanced phase, fewer defects are introduced and less re-work is required, enabling an even larger fraction of work to be completed in the advanced phase. If the cycle works in this direction, the fraction of products introduced with defects declines significantly and the system's performance improves monotonically. In this case the positive loop is instrumental in shifting resources towards the upfront portion of the development process. In case #2, the decrease in development capacity caused by the new methodology initially causes the fraction of current work to increase. The change is, however, only temporary. After three model years, the fraction begins to decline and the new methodology again leads to a substantial reduction in the number of defects introduced.

Case #3 shows different behavior. Whereas in cases #1 and #2 the introduction of the new methodology leads to an improvement in system performance, in case #3 it leads to a substantial decline, even though the model is parameterized so that the new methodology, if fully adopted, leads to an improvement in capability. Figures 16 and 17 show that in case #3, the system evolves to the point where the vast majority of the work is done in the current phase and the defect fraction increases substantially.

Figures 18 and 19 help explain the different outcomes.

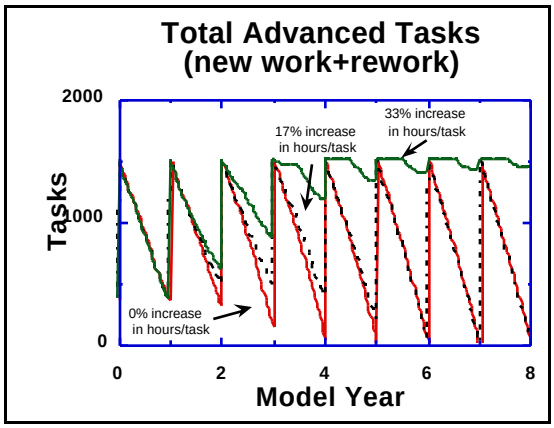


Figure 18

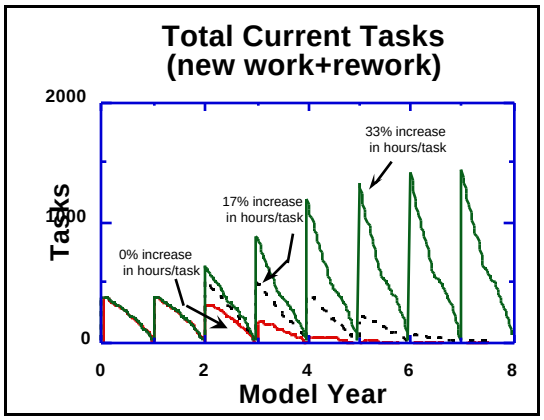


Figure 19

In both cases #2 and #3, due to the increase in the time required per task, fewer advanced tasks are completed in the years following the introduction of the new methodology. As a consequence, more uncompleted work is moved to the current phase. As in earlier cases, more work in the current phase leads to a higher defect introduction rate (figure 20). In case #2, this change is temporary. As the positive benefits of the new methodology kick in (figure 21), the defect introduction rate falls, fewer resources are needed for the current phase, and the advanced work completion rate begins to rise. As the advanced work rate rises, adherence to the methodology increases and system performance improves as the positive feedback loop works in the desired direction.

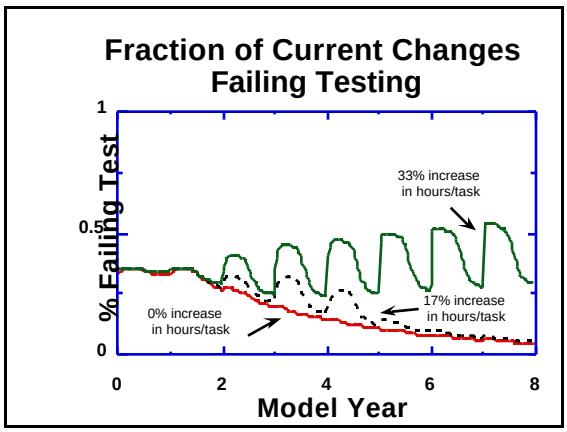


Figure 20

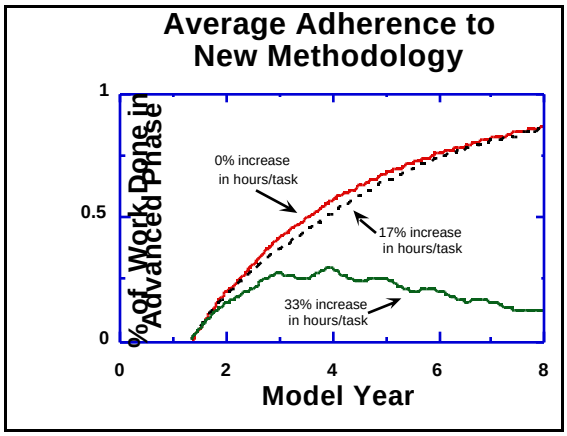


Figure 21

In contrast, in case #3 the advanced work completion rate does not recover. Instead, as more work is shifted to the current phase, the defect introduction fraction rises enough to offset the benefits of the methodology and creates even more current work. In this case, the initial transient is large enough that the system never recovers. The decline in performance is reinforced as an increasing fraction of resources are allocated to current programs and adherence to the methodology declines (figure 21). Here the positive loop works against the implementation of the new methodology and reduces adherence to the new process by pushing resources towards to the downstream portion of the development cycle.

Unlike cases #1 and #2, the system does not make the transition to the desired state. Instead the behavior is very similar to that of the pulse test described in section two. In case #3, the introduction of the new methodology actually makes performance *worse* rather than better. This happens because the cost of the methodology (the reduction in capacity due to increased time per task) is incurred immediately, while the benefits are only received with a delay. The initial reduction in capacity is sufficiently large to cause the dominant positive loops to work in the opposite direction and drives the system to a new steady state behavior in which the system's ability to produce a high quality product is dramatically degraded.

This simulation experiment makes a very important point. The naive introduction of a new set of tools in product development that has the *potential* to improve performance, can actually make performance worse rather than better. This dynamic played a significant role in Repenning and Sterman's (1997) analysis of an improvement effort in the product development function a major US auto manufacturer. In this setting the organization under consideration spent millions of dollars introducing a new computer aided design system, but did not provide the necessary release time and resources to allow engineers to actually

develop the appropriate skills with the tools. Initially, many engineers tried to use the tools, but unfamiliarity with the new tools slowed their progress and caused them to fall behind in their normal development work. Most eventually abandoned the tools in favor of the projects for which they were responsible, and many reported that the performance of their projects suffered for having tried to incorporate the new process technology. This story is unfortunately all too common, but rarely do those who promote new tools and methods worry that their introduction may make the system worse rather than better. As the analysis shows, however, in an environment with resource dependence this is a possibility.

4. Policy Analysis: Limiting Strategies

The simulation experiments show how introducing a beneficial new tool or process in an environment with resource dependence can actually make performance worse rather than better. In this section a policy for mitigating this dynamic is proposed. An organization has a number of ways to compensate for its low process capability, including fixing defects as they occur and investing in long run improvements that eliminate those defects in the before they occur (the classic prevent/correction distinction offered by Deming 1986). Frequently both paths cannot be pursued simultaneously and the organization must allocate scarce resources between fixing current problems and preventing future ones. Repenning and Sterman (1997) discuss in some detail why there is a strong bias in many organizations towards correction.

To counteract this bias, Repenning (*in progress*) introduces the concept of *limiting strategies.* Limiting strategies are based on a simple idea: effective implementation of new improvement methods focused on prevention requires progressively eliminating the organization's ability to use the short-cut correction methods. The archetypal example of such a strategy is the use of inventory reduction to drive fundamental improvements in

machine up-time and yield in manufacturing operations. As the amount of material released to the production system is reduced, operators and supervisors are less able to compensate for low yield and up-time by running machines longer and holding protective buffer inventories. In such a situation the only way to meet production objectives is through fundamental improvements.

The first step in using a limiting strategy is to identify the key mechanisms through which the organization compensates for its low underlying capability. In product development systems one way to compensate for low capability is to redo existing work to correct defects. In the model, this is represented by multiple iterations through the re-work cycle prior to product launch. To use a limiting strategy to implement the new methodology, the allocation of resources to the current phase re-work cycle must be constrained. In the model this is operationalized in a very simple way. At different points during the current phase, a design freeze is introduced and, from that date forward, resources are no longer allocated to re-work.

In the simulations shown below (figures 22 through 24), both the methodology and the freeze date policy are introduced in the first model year. Figure 22 shows the fraction of work dedicated to current programs for a variety of freeze date policies. The 'no-freeze'(freeze at 0) is identical to case #3 discussed above. Without a freeze date, the system never makes the required transition to a higher percentage of advanced work (figure 22), and never develops enough experience with the methodology to reap its benefits (figure 23). The system's performance is significantly worse than if the methodology had not been introduced (figure 24).

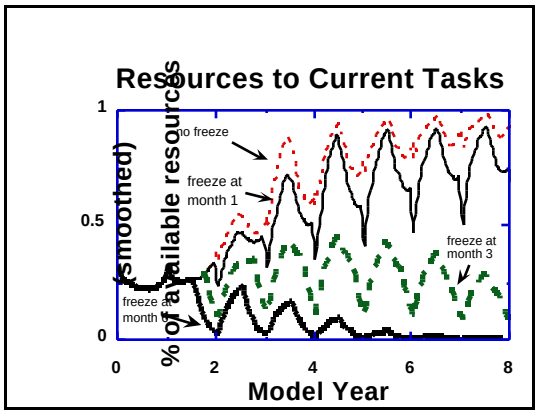


Figure 22

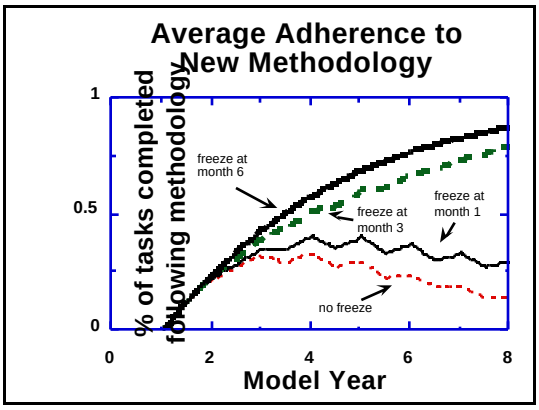


Figure 23

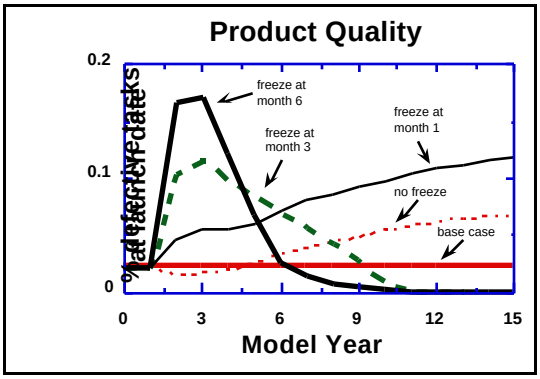


Figure 24

The results are even worse if the freeze date is one month prior to launch. The system fails to make the transition to an increased fraction of work done up-front (figure 22), adherence to the new methodology never reaches above 50% (figure 23) and an even higher fraction of defective products are introduced to the market (figure 24).

In contrast to the no-freeze and one month freeze policies, freezing at months three or six produces a different behavior pattern. In both cases the system makes the transition to the regime in which the vast majority of work is done in the advanced phase (figure 22). With a six month freeze date this happens quickly, while with a three month date there is an extended transition period in which the fraction of resource dedicated to the current project is increased. As a consequence, in both cases the adherence to the methodology increases

monotonically (figure 23). The limiting strategy is effective in ensuring the methodology is implemented.

The long run success does, however, come with a short run cost. The freeze policy creates a substantial 'worse-before-better' dynamic that is common in complex systems (Forrester 1971). In both the three and six month freeze cases the fraction of defective products introduced to the market increases substantially after the policy is put in place (figure 24). The three month date produces a smaller initial increase but also a slower transition to the higher capability system. The six month freeze date creates a substantial increase in the defect introduction rate, but leads to a faster transition to the new process.

The freeze policy works because it weakens the dominant positive loop that creates the tilting dynamic. It comes with a cost because it also weakens the re-work loop and, as a consequence, organizational performance suffers in the short run. Such "worse-before-better" behavior modes are a common feature of many complex systems (Forrester 1971). The decline in performance can, however, be avoided. The "worse" portion of the worse-before-better behavior mode can be incurred in a variety of ways. An alternative to a decline in product quality is to introduce fewer tasks and develop a simpler product. Of course, managers need to select the option that is best for their specific situation. The main insight of the model is that such a trade-off exists. Rarely do managers realize that successful process change often requires that performance declines before it improves, and managers often predetermine the failure of such initiatives by failing to account for such transient dynamics.

5. Conclusion

Many authors have recognized that managers who treat development projects as independent entities do so at their peril (e.g. Cusumano and Nobeoka 1998, Sanderson and Uzimeri 1997). One of the major thrusts of Wheelwright and Clark (1992, 1995) is that

senior leaders should focus on managing the portfolio of development efforts rather than individual projects. It has become an article of faith among business writers that managers should focus on whole systems and processes, not functions and specific pieces (Garvin 1995, Senge 1990, de Geuss 1988, Stata 1989). This shift in perspective is obviously important, but the literature contains less guidance on how to put such thoughts into action. Unfortunately, more is needed than simply recognizing that product development processes are complex systems with interdependent elements. A substantial body of research shows that humans perform poorly in decision making tasks in dynamic environments (Sterman 1989a, 1989b; Brehmer 1992; Funke 1991). In experiments with even modest levels of dynamic complexity, subjects are often outperformed by simple decision rules that account for a fraction of the available information (Paich and Sterman 1993, Diehl and Sterman 1995).

Decision making processes are the major focus of much of the literature on managing product development (Wheelwright and Clark 1992). An alternative approach to improvement is to focus not on the decisions, but on the structure of the process itself. Numerous authors have argued that senior managers have the most impact on a given project very early in its life-cycle (e.g. Gluck and Foster 1975). Similarly, the high leverage point in improving product development processes may lie in improving the structure of the process rather than the day-to-day decisions made within that structure.

The analysis presented here focuses on one decision: the allocation of resources between current and advanced projects. There is a growing body of case study evidence and theory that suggests that people behave myopically in these contexts, specifically over-investing in current projects and ignoring longer term investments (Repenning and Sterman 1997, Jones 1997). Combining resource dependence and such myopic behavior creates a system with a number of undesirable characteristics: the system is not robust to variations in work load, and new tools and processes are difficult to implement. Trying to improve the quality

of a manager's decision in this context may difficult. Repenning and Sterman (1997) suggest that there are a number of basic cognitive routines that bias managers towards short term throughput and against investing in long run improvement. A limiting strategy, which is operationalized as fixing resource allocation rather than leaving it to the discretion of the manager, is a simple solution to this problem that greatly increases the robustness of the process to variations in workload. The policy can also dramatically improve the effectiveness of new tools that are introduced.

Placing constraints on the allocation of resources between projects is only one small example of how processes can be redesigned to be more robust and produce consistently better results. The methodology of robust design has been widely propagated through the product development community with great success in many areas. Substantial improvements have been made in the quality and reliability of many projects through designs that minimize the sensitivity of the final item's performance to variations in the dimensions of its constituent components.

The core idea of robust design is applicable to the design of processes and organizations and has been a focus in the field of System Dynamics since its inception (Forrester 1961). In most cases it is, of course, prohibitively costly to run the needed experiments in actual organizations and, unlike developing new products, when developing new processes prototyping is often not an option. Thus the development of models to capture the dynamics of such processes is critical to understanding which policies are robust to changes in the environment and the limitations of the decision maker. System Dynamics provides an important means to generate useful models of organizations and processes and can contribute substantially to the emerging theory of designing and managing multi-project development processes.

References

Abdel-Hamid, T. and S. Madnick. 1991. *Software Project Dynamics: An Integrated Approach*, Englewood Cliffs, NJ: Prentice Hall.

Adler, P.S., A. Mandelbaum, V. Nguyen and E. Schwerer. 1995. From product to process management: an empirically-based framework for analyzing product development time, *Management Science*, 41, 3: 458-484.

Brehmer, B. 1992. Dynamic Decision Making: Human Control of Complex Systems, *Acta Psychologica*, 81: 211-241.

Carroll, J., J. Sterman, and A. Markus. 1997. Playing the Maintenance Game: How Mental Models Drive Organization Decisions. R. Stern and J. Halpern (eds.) Debating Rationality: Nonrational Elements of Organizational Decision Making. Ithaca, NY: ILR Press.

Cooper, K. G. 1980. Naval Ship Production: A Claim Settled and a Framework Built, *INTERFACES*, 10,6.

Cusumano, M. and K. Nobeoka. 1998. *Thinking Beyond Lean*, New York: Free Press.

de Geus, Arie .1988. Planning as Learning, *Harvard Business Review*, March-April: 70-74.

Deming, W. E. 1986. *Out of Crisis*, Cambridge, MA.: MIT Center for Advanced Engineering.

Dertouzos, M., R. Lester, and R. Solow. 1989. *Made in America: Regaining the Productive Edge*, Cambridge, MA. : The MIT Press.

Diehl, E. and J.D. Sterman. 1995. Effects of Feedback Complexity on Dynamic Decision Making, *Organizational Behavior and Human Decision Processes*, 62, 2:198-215.

Eppinger, S.D, M.V. Nukala, and D.E. Whitney. 1997. Generalized Models of Design Iteration Using Signal Flow Graphs, *Research in Engineering Design*, 9:112-123.

Ford, D. N. and J. D. Sterman. 1998. Dynamic modeling of product development processes, *System Dynamics Review*, 14, 1:31-68.

Forrester, J. W. 1961. *Industrial Dynamics*, Cambridge, MA: The MIT Press.

Forrester, J. W. 1971. Counterintuitive Behavior of Social Systems, *Technology Review*. 73,3: 52-68.

Forrester, J.W and P. Senge. 1980. Tests for Building Confidence in System Dynamics Models, *TIMS Studies in the Management Sciences*, 14: 209-228.

Funke, J. 1991. Solving Complex Problems: Exploration and Control of Complex Systems, in R. Sternberg and P. Frensch (eds.), *Complex Problem Solving: Principles and Mechanisms.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Gluck, F. and R. Foster. 1975. Managing Technological Changes: A Box of Cigars for Brad, *Harvard Business Review*, Sept.-Oct.: 139-150.

Homer, J., J. Sterman, B. Greenwood, and M. Perkola. 1993. Delivery Time Reduction in Pump and Paper Mill Construction Projects: A Dynamics Analysis of Alternatives, *Proceedings of the 1993 International System Dynamics Conference*, Monterey Institute of Technology, Cancun, Mexico.

Jones, A.P. 1997. Sustaining Process Improvement in Product Development: The Dynamics of Part Print Mismatches, unpublished MS Thesis, Department of Civil and Environmental Engineering, Massachusetts Institute of Technology, Cambridge, MA.

Krahmer, E. & R. Oliva. 1996. Improving Product Development Interval at AT&T Merrimack Valley Works. Case history available from authors, MIT Sloan School of Management, Cambridge, MA.

Oliva, R., S. Rockart, and J. Sterman. 1998. Managing Multiple Improvement Efforts. Advances, in the *Management of Organizational Quality*. D. B. Fedor and S. Goush. Stamford, CT, JAI Press. 3: 1-55.

Paich, M. and Sterman, J. 1993. Boom, Bust, and Failures to Learn in Experimental Markets, *Management Science*, 39, 12: 1439-1458.

Repenning, N. *in progress*. Towards a Theory of Process Improvement. Working Paper, Alfred P. Sloan School of Management, Cambridge, MA. (available at <http://web.mit.edu/nelsonr/www>.

Repenning, N. 1998a. "The Transition Problem in Product Development", Working Paper, Alfred P. Sloan School of Management, Cambridge, MA. (available at http://web.mit.edu/nelsonr/www/).

Repenning, N. 1998b. Successful change sometimes ends with results: Path dependence in participatory improvement efforts. Working Paper, Alfred P. Sloan School of Management, Cambridge, MA. (available at http://web.mit.edu/nelsor/www/).

Repenning N. and J. Sterman. 1997. Getting Quality the Old-Fashion Way: Self-Confirming Attributions in the Dynamics of Process Improvement, to appear in Improving Research in Total Quality Management, National Research Council Volume, Richard Scott and Robert Cole, eds. (available at http://web.mit.edu/nelsonr/www/).

Richardson, G. P. and Alexander Pugh. 1981. *Introduction to System Dynamics Modeling with DYNAMO*, Cambridge, MA: MIT Press,.

Roberts, E. B. 1978. *Managerial Applications of System Dynamics*, Cambridge, MA.: Productivity Press.

Roberts, E.B. 1964. *The Dynamics of Research and Development*, New York, NY: Harper and Row.

Sanderson, S., and M. Uzumeri. 1997. *Managing Product Families*, Chicago, IL: Irwin.

Senge, P. 1990. *The Fifth Discipline: The Art and Practice of the Learning Organizations*, New York, NY: Doubleday.

Stata, R. 1989. Organizational Learning: The Key to Management Innovation, *Sloan Management Review*, 30, 3: 63-74.

Sterman, J.D. *forthcoming. Business Dynamics: Systems Thinking and Modeling for a Complex World.* Chicago, IL, Irwin/McGraw Hill.

Sterman, J. D. 1989a. Misperceptions of Feedback in Dynamic Decision Making, *Organizational Behavior and Human Decision Processes*, 43, 3: 301-335.

Sterman, J. D. 1989b. Modeling Managerial Behavior: Misperceptions of Feedback in a Dynamic Decision Making Experiment, *Management Science* , 35, 3: 321-339.

Sterman, J., N. Repenning, and F. Kofman. 1997. Unanticipated Side Effects of Successful Quality Programs: Exploring a Paradox of Organizational Improvement, *Management Science*, April: 503-521.

Weil, H., T. Bergan and E.B. Roberts. 1978. The Dynamics of R&D Strategy, in Roberts, E.B. (ed), *Managerial Applications of System Dynamics*, Cambridge MA: Productivity Press.

Wheelwright, S. and K. Clark. 1993. *Revolutionizing Product Development: Quantum Leaps in Speed Efficiency and Quality*, New York, NY: The Free Press.

Wheelwright, S. and K. Clark. 1995. *Leading Product Development*, New York, NY: The Free Press.

Ulrich, K. and S. Eppinger. 1995. *Product Design and Development*, New York, NY: McGraw-Hill.

Zangwill, W. 1993. *Lightning Strategies for Innovation: how the world's best create new projects*, New York, NY: Lexington Books.