Why Firefighting Is Never Enough:

Preserving High-Quality Product Development

Laura J. Black[1]

Nelson P. Repenning[2]

Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA USA 02142

Version 1.0

Work reported here was supported by the MIT Center for Innovation in Product Development under NSF Cooperative Agreement Number EEC-9529140.

For more information on the research program that generated this paper, visit
http://web.mit.edu/nelsonr/www/.

1. MIT Sloan School of Management, E53-364**,** Cambridge, MA USA 02142. Phone 617-253-6638 Fax: 617-258-7579; <lblack@mit.edu>.

2. MIT Sloan School of Management, E53-339**,** Cambridge, MA USA 02142. Phone 617-258-6889; Fax: 617-258-7579; <nelsonr@mit.edu>.

Abstract

In this paper, we add to insights already developed in single-project models about insufficient resource allocation and the “firefighting” and last-minute rework that often result by asking why dysfunctional resource allocation persists from project to project. We draw on data collected from a field site concerned about its new product development process and its quality of output to construct a simple model that portrays resource allocation in a multi-project development environment. The main insight of the analysis is that under-allocating resources to the early phases of a given project in a multi-project environment can create a vicious cycle of increasing error rates, overworked engineers, and declining performance in all future projects. Policy analysis begins with those that were under consideration by the organization described in our data set. Those policies turn out to offer relatively low leverage in offsetting the problem. We then test a sequence of new policies, each designed to reveal a different feature of the system's structure and conclude with a strategy that we believe can significantly offset the dysfunctional dynamics we discuss. The paper concludes with a discussion of the challenges managers may face in implementing the strategy that can prevent persistent under-allocation of resources to projects.

1. Introduction

There are few images more common in current discussions of R&D management than that of the overworked engineering team putting in long hours to complete a project in its last days before launch. In venues ranging from the comics in the daily papers (e.g., “Dilbert,” Adams 1996) to scholarly studies of specific development projects (e.g. Walton 1997, Ford and Sterman 1998, Repenning and Sterman 2000), accounts of the execution of development projects almost universally depict the final stages as requiring heroic efforts by the associated engineering team. Such final "fire fights" often include discovery of errors that should have been identified months earlier, require significant departures from the desired product development project, and entail allocating significant additional engineering hours and other valuable resources. Despite these measures, projects experiencing these dynamics often fail to meet the customer's expectations.. Spending more time and more money to produce a product with fewer capabilities is a choice few people willingly make, but while this mode of project execution is roundly criticized by scholars, consultants, and practitioners, it remains one of the most persistent features of current product development practice.

The existence and persistence of this phenomenon remain as open and important questions facing both managers and students of R&D management. These questions also represent an arena in which theory inherent in current system dynamics models far exceeds that available in other domains. Starting with Roberts in 1964 and continuing with studies by Cooper (1980), Abdel-Hamid (1988), Homer *et al.* (1993), and most recently with Ford and Sterman (1998), system dynamics has developed an increasingly general and powerful framework for understanding how large-scale development projects often go awry and require the last-minute measures described above. Due in part to the immense success of the Pugh-Roberts consulting company in using this framework (e.g. Cooper 1980), the analysis of individual development projects may be one of the single biggest contributions of the system dynamics method to improved management practice.

While this literature identifies and explicates a wide range of dynamics and suggests numerous policies, one theme that runs through most analyses is the importance of beginning projects with realistic schedules and appropriate amounts of resources. Projects with overly ambitious deadlines and too few resources often fall prey to a host of self-reinforcing dynamics and become trapped in a quagmire of increasing works hours, fatigue, error rates, and deadline pressure. While many struggle to understand why projects so often fail to meet expectations and at the same time require additional engineering hours, in many ways system dynamics analysts have already developed significant insight: Starting a project with too few resources almost always ensures it will eventually require too many.

While the literature convincingly explains the features necessary for the *existence* of the phenomenon, in many environments it's harder to understand its *persistence*. In particular, if starting with an overly ambitious schedule and too few resources is so debilitating, why do organizations continue to do it? There are a number of plausible answers. Most prominently, as Sterman and others (Sterman 1989, Diehl and Sterman 1995, Moxnes 1999) have well documented, people just aren't very good at learning in dynamic environments, especially when there is little repetition in events and long delays between cause and effect. Thus, in organizations where development projects take many years to complete (e.g. ship building or large construction projects), managers are unlikely to learn that their own actions can create the difficulties they face. In these settings, a manager might oversee fewer than five projects in her career.

In other environments, however, development cycles are measured in months rather than years, and in these settings the inability of organizations to balance work and resources becomes more difficult to understand. The conditions for learning are improved, and experienced managers may have the opportunity to supervise dozens of projects. Thus, in many development environments, the insight from single-project models, while necessary, may not be sufficient to explain the persistence of the flurry of activity that often accompanies the completion of projects, particularly in environments characterized by relatively short development cycles and multiple projects in progress.

In this paper, we add to the insights already developed in the single-project context by focusing on the question of why resources are often allocated to projects in such a dysfunctional manner. We begin our analysis with an existing model (developed in Repenning 2000) that captures the dynamics of resource allocation in multi-project development environments. The main insight of this analysis is that under-allocating resources to the early phases of a given project in a multi-project environment can create a vicious cycle of increasing error rates, overworked engineers, and declining performance in all future projects. While this analysis begins to answer some of the questions posed above, two features limit its applicability. First, the original paper contains no policy analysis. Thus, while the model offers an explanation for the phenomenon of interest, it is silent on the question of what managers might do about it. Second, the model is highly stylized. Most important, in the original model testing is represented as an uncapacitated delay. This assumption precludes analysis of a key popular policy lever, alternative strategies for allocating resources to testing.

To overcome these limitations, in this paper we do the following: First, working from a data set collected by the first author, we propose an enhanced model grounded in the experience of one organization. Second, we spend relatively little time on the analyzing the basic behavior of the model, since it replicates that of the Repenning's paper. Instead, we focus attention on policy analysis. We begin with the policies that were under consideration by the organization described in our data. Consistent with one of the most commonly observed features of management practice (e.g. Forrester 1971), those policies turn out to have relatively low leverage in offsetting the problem. Following this, we test a sequence of other policies, each designed to reveal a different feature of the system's structure and conclude with a strategy that we believe can significantly offset the dysfunctional dynamics we discuss. Finally, we conclude with a discussion of the challenges managers may face in implementing this strategy.

To that end, the paper is organized as follows. In the next section we briefly describe the effort to collect and analyze data at the selected field site. Following that, in section two we present the main model and briefly discuss its base case behavior. In section three we present policy analysis. Section four contains discussion and concluding thoughts.

2. Overview of the Field Site

In 1998 the first author spent three months at a midsize Midwest manufacturing company, conducting interviews and observing production activities and meetings in order to prepare at the company's request an assessment of the product development process and launch. The report summarized 33 interviews, including 10 project teams and 15 process teams in six facilities. Interview participants represented a variety of departments, including marketing, developmental and operational purchasing groups, design engineering, data processing, cost accounting, manufacturing engineering, and assembly. Among the top 15 themes identified as "what went wrong" during development for the model year launched, design instability ranked fourth, and late testing and insufficient testing resources ranked ninth. This particular manufacturing company has during the last decade established a solid development method, introduced more rigorous project reviews, and improved in myriad ways its new product development process. Despite these advances, and despite increasing its engineering staff and giving heightened managerial attention to new product efforts, the problem of overworked engineers working feverishly to meet the project deadline persists.

One program manager at the research site said, "We don't know what we don't know until it's *real* late," elaborating that critical issues in product development projects consistently surface in the last months before the scheduled market introduction[1]. Design changes to address the critical problems then throw into furious activity development engineers, component suppliers, manufacturing engineers responsible for tooling, and people working on dependent projects, straining communications and jeopardizing product quality. By the time the product hits the assembly line on the company's annual launch date, according to one wry comment, everyone involved is holding up "the company's victory symbol--crossed fingers."[2]

[1] "Launch Assessment: 1999 Model Year," 1998, company report (name withheld in accordance with researchers' confidentiality agreement).

The data collection also included reviewing problem logs kept during development and the number of parts officially authorized by engineering at various points during development. By comparing these with previous years' records, here, too, a consistent theme emerged: The organization identifies problems in new product efforts later in the development cycle rather than earlier, and even the few problems identified fairly early in the development process remain unresolved until shortly before launch. As a project nearing launch accumulates an alarming number of unresolved problems, it absorbs resources otherwise devoted to projects scheduled for later launch dates. Solving the urgent problems necessitates changes to product design--both design and manufacturing engineers consistently decry "late changes," design alterations that take place after a key prototype build occurring about eight months before launch. This prototype is, according to the company's product development methodology, to signal the *end* of the design phase. As a more recent launch assessment (prepared by another researcher) of the company's development and launch activities points out, "Builds serve as targets and launch is the only milestone. Problems are neither resolved nor expected to be resolved until launch."[3]

This story of late problem identification and still later changes and of indirect but inevitable disruption to other projects led us to investigate more thoroughly the role that testing and prototype builds can play in an ongoing stream of product development activity. While the

[2] Ibid.

report raised a number of interesting issues, the modeling effort reported in this paper was targeted specifically at explaining why major problems were so often discovered and resolved late in the development cycle.

3. Model Structure

To develop the model, we began with observations of the development activities of the company in question. First, the company typically launches products on an annual cycle. Due to industry norms, product launch is rarely, if ever, delayed. Products that are not ready for a given model year must be postponed for another twelve months. Second, while the company has a "rolling" product plan that outlines which products will be launched when over the next five years, the majority of actual development work is focused on the two model years nearest their launch dates. Thus, for both simplicity and tractability, and to avoid revealing proprietary details, we assume that the product development organization in our model works on projects in two main phases during a period of two years. There is an up-front or *early* phase that occurs between twenty-four and twelve month prior to launch, and there is a downstream or *current* phase that takes place in the twelve months before the product's introduction date. Figure 1 depicts the scheme of overlapping product development efforts.

[3] "How the Rubber Met the Road: Product Development Process Assessment, Model Year 2000," 1999, company

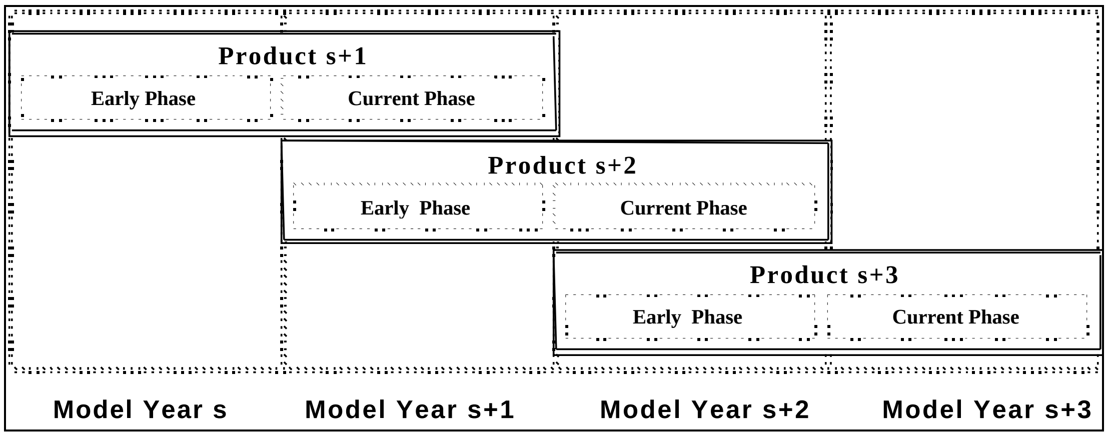

Figure 1: Overlapping Product Development Efforts

The unit of work in our analysis is the part. Each model year begins in the early phase with a certain number of parts that must be designed, tested, and possibly revised in order to assure high quality. We assume that all parts contribute equally to the performance of the final product and that there is no possibility of system or sub-system level defects (i.e. a problem created when two parts are each designed correctly but do not fit together). When the company launches Model Year *s* (market introduction takes place at a fixed time each year), then the next model year's work-in-process transitions from the early phase to the current phase, and also early phase work begins on the model year to launch two years after Model Year *s*.

report (name withheld in accordance with researchers' confidentiality agreement).

The Flow of Work

The model portrays essentially the same activities and sequence of work in each of the two development phases. In each phase, parts can exist in one of four states: parts to be designed; prototyped parts; tested parts; and revised parts. Parts move from one state to another based on the engineering work rate, and the work rate depends on the number of engineers assigned to each activity (designing, testing, or problem-solving/revising) and the average productivity per engineer in that endeavor. A simplified diagram of a single phase, Figure 2, shows the accumulation of parts in various states and the rates that move parts from one state to another.

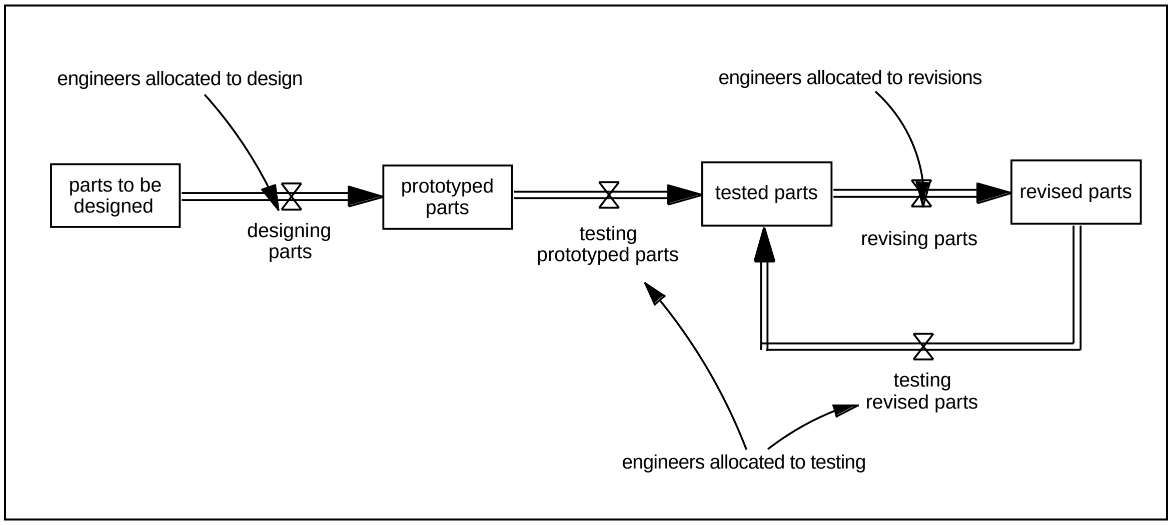

Figure 2: The Flow of Work Within a Development Phase

At each product launch, work in the current phase enters the market; work in the early phase becomes current work, and the early phase begins design of new parts for the model year to launch in 24 months. At this annual transition, parts in each state of the early phase move directly to the corresponding state in the current phase. Thus parts that were tested in the early phase do not need to be tested again in the current phase, unless testing discovered a problem that necessitates revision.

In contrast to Repenning's (2000, forthcoming) work, which portrayed testing as an uncapacitated delay, here testing takes place under one of two conditions: engineers are allocated to testing or a prototype "build" takes place during the current phase. When testing is resource-based, testing doesn't occur unless there are parts whose design or revision is complete but untried *and* engineers are allocated to testing those parts. When a prototype build occurs, all the designed parts are effectively tested within a two-week period with no allocation of engineering resources.

Elaborating on Key Assumptions

The distinction between the early and current phases is critical for two reasons. First, in the early phase, the probability of a design error is one-third that if the part is designed during current phase. The difference between the two defect rates reflects benefits accruing both from fewer complicating interdependencies among parts when the design is in an earlier stage and from a longer lead time for choosing materials and processes (although dependencies, material selection, and process design are not explicitly represented in the model). For simplicity's sake, only one kind of "error" is represented in the model.

Second, the distinction between the two phases is also important in determining resource allocation. Priority for allotting engineers to the three main activities of designing, testing, and problem-solving/revising is influenced by how near a project is to its scheduled launch date. This model represents an organization in which resources are scarce and the slate of new products is ambitious. There is little slack in the system. The model depicts the company's bias for allocating more resources to the project nearest to launch by continuously estimating the total amount of work to do and allocating as many engineers as needed (or as many as are available) to the following activities in descending priority:

- Designing current phase parts;
- Testing current phase parts;
- Revising current phase parts;
- Designing early phase parts;
- Testing early phase parts; and
- Revising early phase parts.

Since this way of prioritizing is similar to those in the phases outlined in numerous product development textbooks (e.g. Ulrich and Eppinger 1995, Wheelwright and Clark 1992) we label it the "By the Book" allocation scheme. It is a rational, robust method for allocating engineers to work. If there is no work to be done in a particular activity, no engineers are assigned to it. If the organization is feeling pressure from an overload of incomplete or problematic parts when launch is imminent, it pulls engineers from other activities onto the more urgent efforts.

The "base" workload in these simulations is 750 parts, comparable to what the field site studied actually undertakes. Diagrams and an equation listing of the model are attached as an appendix, and the model can be download from the internet.[4]   To study the dynamics that this system generates, we primarily rely on one-time increases in a single year’s workload as test inputs. In each of the simulations below, the one-time (i.e., one model year's) increase in workload takes place at the beginning of the second model year in the simulation. Because the development cycle requires two years, that higher-content model year launches at year three.

We rely on one-time increases (or pulses) as test inputs because we are primarily interested in understanding how the system behaves when subjected to variations in workload driven by both changing product and market requirements and the discovery of unanticipated problems that require significant resources for correction. While in some analysis we used more complicated test inputs (e.g. pink noise), pulse tests simplify the process of presenting and explicating the system's dynamics

Given the assumption of a fixed development cycle, we assess the performance of the development system with the variable *quality*, measured as the percentage of the new model year parts designed and launched correctly. Parts that are incompletely or incorrectly designed (in other words, parts with errors) detract from quality. This measure of the system is rooted in our field data. One interview yielded the assertions that "quality is the default variable that moves" and that the team "can't move any [other] variables [including] time, people, or dollars." Another project team suggested that expectations for low-cost, on-time delivery came "at the expense of quality."

4. Base Case

We now briefly summarize the base case behavior of the model. Figures 3 through 6 show the behavior of the major stocks in the early phase while figures 7 through 10 highlight the evolution of the major states in the current phase. In each graph we show two simulations, the base case, in which the annual workload is steady at 750 parts, and a pulse test, in which the second model year's workload is temporarily increased by 75 percent over the base level.

[4] . Please see http://web.mit.edu/nelsonr/www.

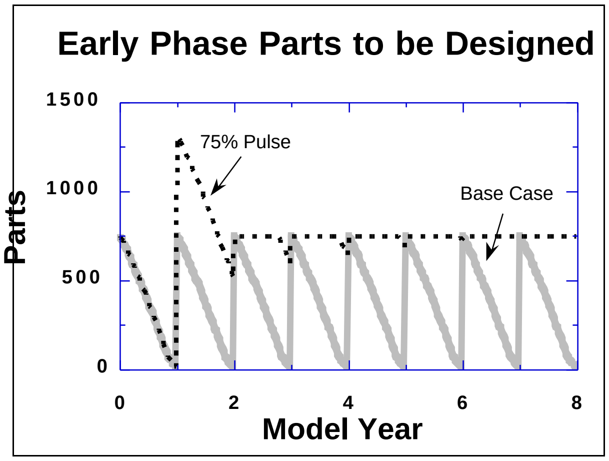

Figure 3: Base Case—Early Phase Parts to be Designed

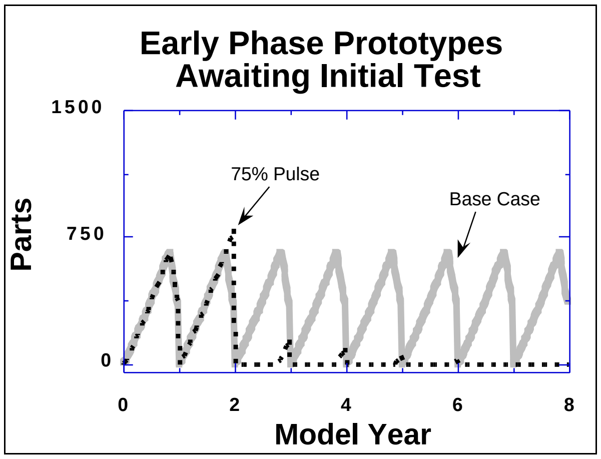

Figure 4: Base Case—Early Phase Parts Prototyped

As Figure 3 shows, in the base case, almost every part is designed during the early phase. These parts then accumulate in the stock of prototyped parts awaiting testing (Figure 4), and then a relatively small number of them are tested (Figure 5). Of those parts tested, some of the defective ones are revised (Figure 6). Thus, in the base case, while there are enough resources to design each part once during the early phase, most of the rework remains to be done in the current phase.

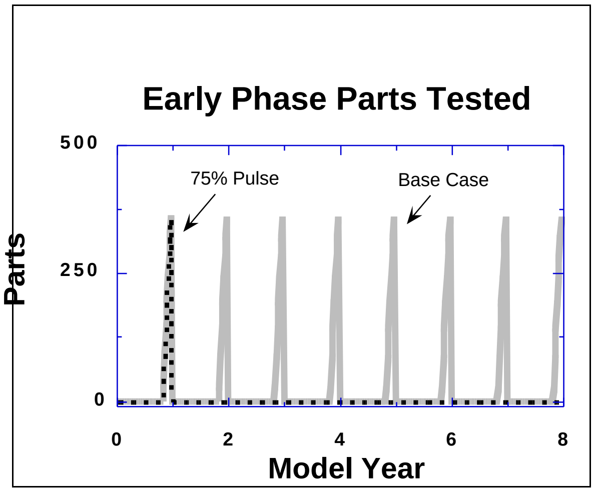

Figure 5: Base Case—Early Phase Parts Tested

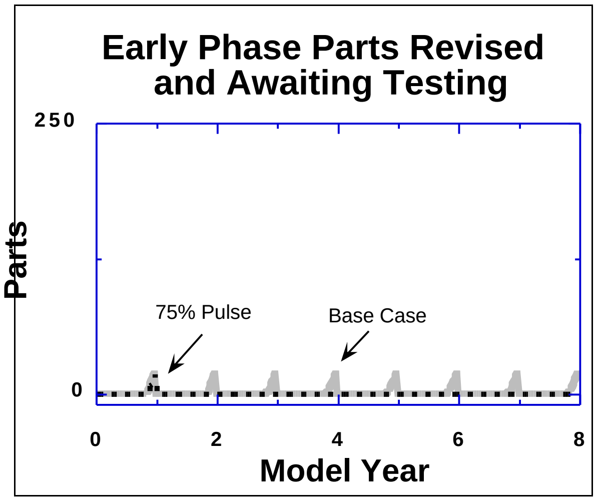

Figure 6: Base Case—Early Phase Parts Revised

While almost all parts have been designed by the time the project reaches the current phase (Figure 7), an unknown amount of rework remains because not all prototyped parts are tested during the early phase. Thus at the model year transition nearly 400 prototyped parts remain to be tested (Figure 8). Once the current phase starts, these tests are rapidly completed, and the stock of tested parts grows rapidly (Figure 9). Further, due to the existence of a prototype “build” that takes place eight months before launch, the remaining stock of untested prototypes is drawn to zero following the build (Figure 8).

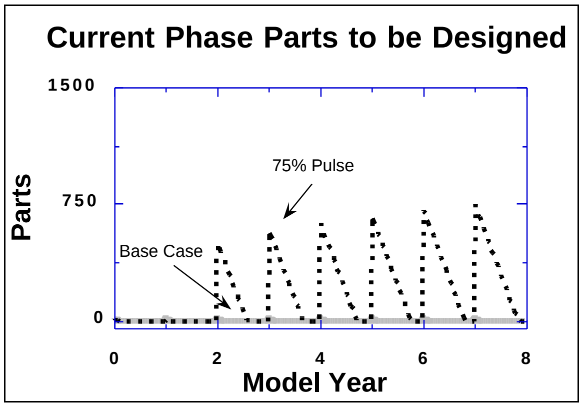

Figure 7: Base Case—Current Phase Parts to be Designed

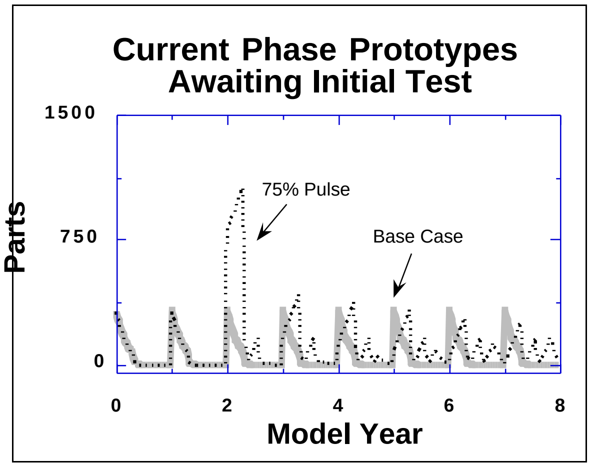

Figure 8: Base Case—Current Phase Parts Prototyped

Once all the parts are tested, resources are dedicated to revisions, causing the stock of revised parts to grow late in the model year (Figure 10). This stock is then drawn down by the second prototype build, five months before launch. The model is parameterized so that in the base case the final product launches with essentially zero defects.

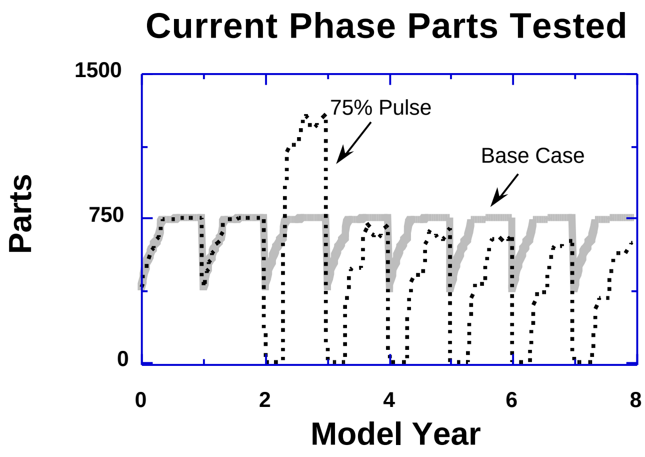

Figure 9: Base Case—Current Phase Parts Tested

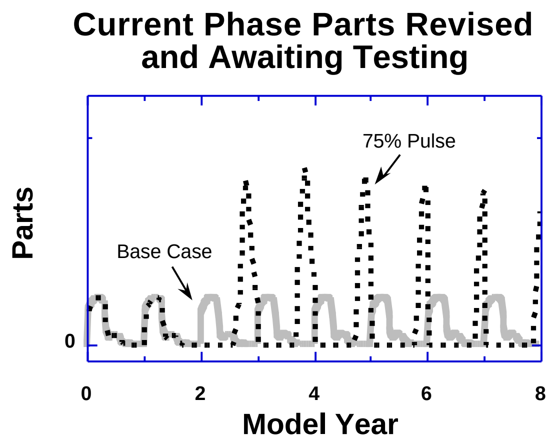

Figure 10: Base Case—Current Phase Parts Revised

In contrast to the desirable state of affairs shown in the base case, the simulation with the pulse shows different behavior. The extra workload in model year one first manifests itself as an increase in the number of parts to be designed in the early phase (Figure 3). In the early phase, approximately the same amount of work is accomplished, resulting in almost all the extra work being transferred the current phase (Figure 7). Importantly, although the increase happens in only one model year, once it occurs, engineering activity in the early phase essentially ceases for all following model years. Few parts are designed in subsequent model years' early phase (Figure 4), and absolutely no early phase testing or revising is accomplished (figures 5 and 6).

The cause of this dramatic shift is shown in the behavior of the current phase stocks. The probability of introducing a defect when completing a design is higher in the current phase, thus causing more parts to fail testing and require revision (Figure 10). Allocating resources to revise parts scheduled to launch imminently precludes devoting resources to the early phase activities, thus perpetuating the cycle, despite the fact that the workload returns to its pre-pulse base level in the third model year and for every year thereafter. Interestingly, the prototype builds play a more important role following the pulse. Whereas before the pulse, relatively few errors are discovered in the "builds" (their existence is almost imperceptible in the graphs), following the increase in one year's workload, the builds become an obvious source of testing. Their role is easiest to see in Figure 9 where the number of tested parts shoots up following each build. The central role of the builds is consistent with observations at the research site, including one project manager's assertion that many engineers rely on the first prototype build to mark the beginning of serious design rather than the completion of the design phase.[5]

We summarize the dynamics of this system with the surface plot shown in Figure 11.

[5] "Launch Assessment: 1999 Model Year," 1998, company report (name withheld in accordance with researchers' confidentiality agreement).

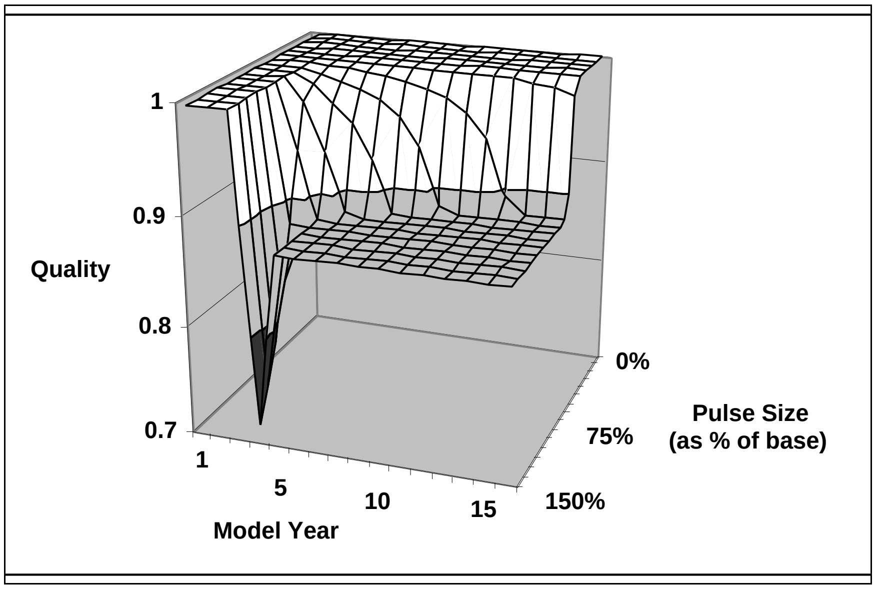

Figure 11: 3-Dimensional Surface Showing Quality in a Series of Simulations

for a Variety of One-Time Increases in Workload

To generate the figure, we ran the model for a variety of pulse sizes, ranging from 0 to 150 percent of the base case. As the figure shows, given the selected parameters, the system is able to accommodate a shock less than 40 percent of the base case without significant degradation in long-term performance. For larger shocks, however, the situation changes. Immediately following the one-time increase in workload, quality falls dramatically. This is not particularly surprising, given that engineering resources have not changed. More significantly, however, performance never recovers, even though the workload returns to the original level. Instead, the system becomes stuck in a second equilibrium in which performance has fallen by approximately 15 percent.

The feedback processes that generate this behavior are highlighted in Figure 12. Beginning at the lower left of Figure 12, we see that more current phase work to do leads to an increase in resources allocated to current phase. This increases the current phase work rate, which reduces the accumulation of undesigned, untested, or unrevised parts in current phase. This process of balancing resources to work is referred to as the Current Phase Work Completion Loop. A similar process exists for early phase.

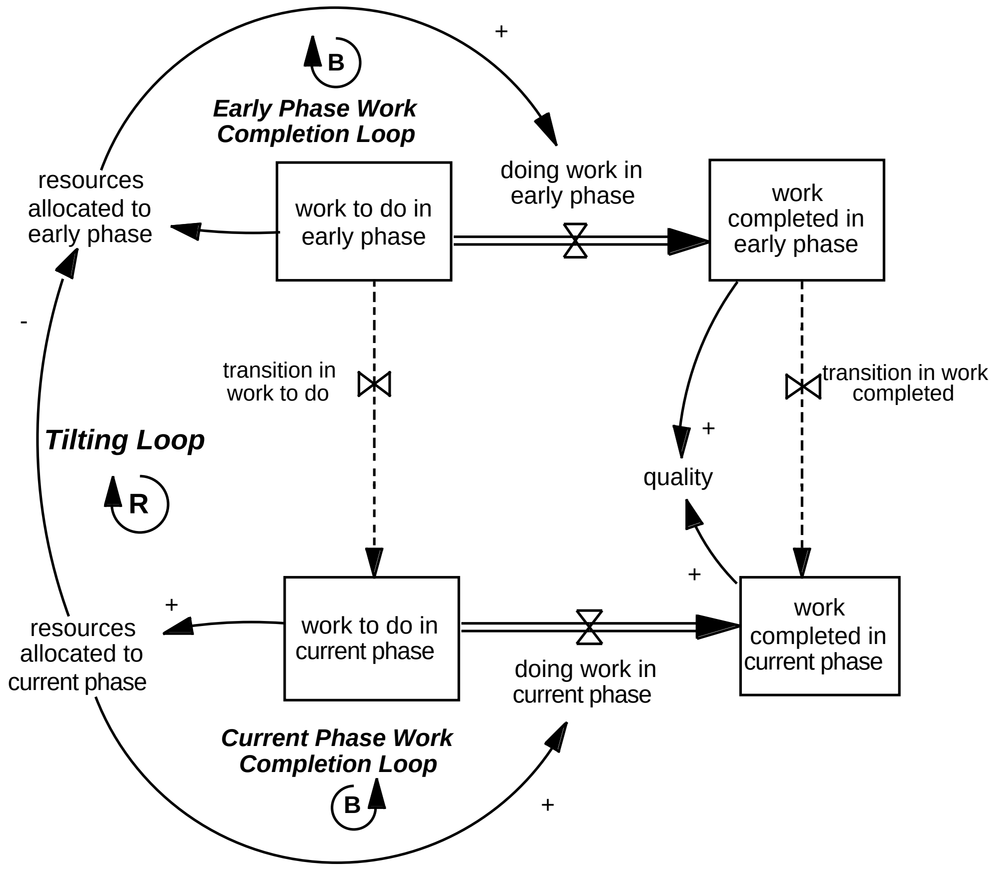
B
Early Phase Work Completion Loop
+
resources allocated to early phase
work to do in early phase
doing work in early phase
work completed in early phase
-
transition in work to do
transition in work completed
Tilting Loop
R
+
quality
+
+
resources allocated to current phase
work to do in current phase
doing work in current phase
work completed in current phase
Current Phase Work Completion Loop
B
+

Figure 12: Dynamics Arising from Resource Allocation

As resource allocation to current phase activity increases, however, resources allocated to early phase work necessarily decrease, since resources are finite in this model. Then less work is accomplished during the early phase. At the next model-year transition, incomplete work in the early phase suddenly becomes incomplete work in the current phase. Since current phase work receives priority, resource allocation to current phase increases again, perpetuating the lower early phase work rate. This phenomenon we call "tilting"--when the reinforcing dynamic that shifts resources *between* phases dominates the *within* -phase completion dynamics and drives more and more work to be completed only in the last months before launch.

Based on the parameters here, if the simulated company tries to complete all a model year's work within the twelve months before launch, it does not have sufficient resources to design (with a higher probability of mistake) all the parts and then fix all the resulting problems. A one-time increase in parts per model year can "tilt" a healthy product development organization into doing a greater percentage of a model year's work during the months immediately preceding launch each year. Once the company is caught in such a "firefighting" mode, since it favors completing all current phase activities before undertaking any early phase work, it is stuck, continuing to operate in a lower-quality regime long after the workload returns to its original level.

5. Policy Analysis

Adjusting the Timing of Prototype Builds

Having outlined the base case, we now turn to policy analysis. Following the suggestions of participants at our field site, we began our search for policies for improving performance by looking at the timing of the prototype builds. The rationale underlying this suggestion was that the builds, by surfacing problems, might force problem resolution earlier in each model year. Extensive analysis of the model, however, suggests that, within the framework we propose, the location of the prototype builds holds relatively low leverage for developing a more robust system. Figure 13 shows one simulation experiment.

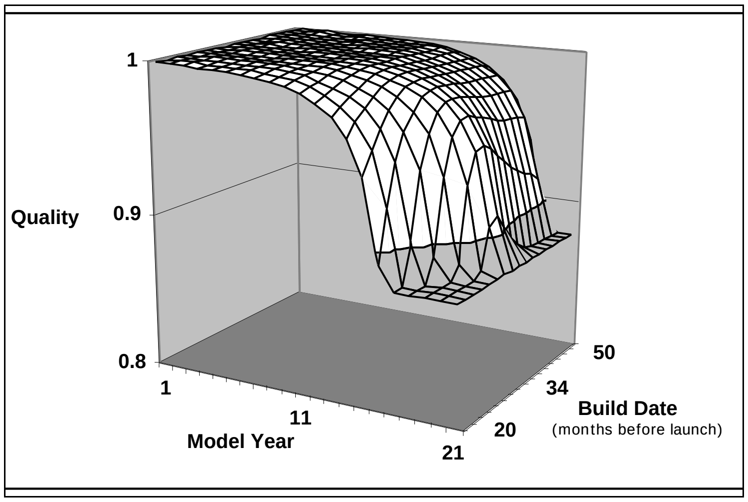

Figure 13: 3-Dimensional Surface Showing Quality

Under a Variety of Dates for Prototype Build

When There Is a One-Time 50 Percent Increase in Workload

To generate the figure, we ran a number of simulations in which we varied the date of the first prototype build between 52 weeks before launch and 20 weeks before launch (the date of the second prototype build at the field site studied). We did not find a single situation in which moving the build prevented the system from "tilting" due to an external shock (here, a one-time increase of about 50 percent of the base workload). Instead, as Figure 13 shows, the best the builds can accomplish is to slow the downward spiral. In fact, as the figure also suggests, the current location of the field site’s first build, at 34 weeks prior to launch, seems to provide the most damping to the decline in quality. In all cases we studied, however, the prototype build’s timing had little influence on the robustness of the system.

While it may not be obvious at first glance, this is a somewhat surprising result. Recall that as they are currently modeled, the prototype builds represent essentially free testing. Given that the pathological dynamics we identify result from too many projects and too few resources, it stands to reason that a prototype build, if it represents an incremental increase in resources, could provide an important leverage point. In fact, however, this is not the case. The error in the above logic stems from the fact that the builds actually have two impacts on the system's dynamics. As just highlighted, they increase effective capacity by introducing free testing. They have a second role, however, that, rather than making the system more stable, actually makes it more sensitive. This second role is highlighted in the following set of tests.

Altering Resource Priorities

In the second set of experiments, we alter the priority ordering of various tasks. Recall that in the base case, resources are allocated following the textbook recommendation of build, test, and problem-solve/revise. While many of the other possible resource allocation schemes yield results similar to the base case, one scheme, which we dub "urgent first," produces substantially different results. In this scheme, we assume that rather than following the textbook allocation, engineers give first priority to revising current parts that are found to be defective. Second priority then goes to designing new parts, and testing occurs only where there are additional resources not dedicated to designing or revising. In other words, testing happens only when there is no "real" engineering work to do. A similar scheme is followed in the early phase, and, of course, the current phase still gets priority over the early phase. We make an additional change in this scenario: No prototype builds take place. This further exaggerates the product development system’s characteristic that testing takes place only if engineers don’t have anything else to do.

With this ordering scheme in place, we again simulate the model for a range of shocks and summarize the results in Figure 14.

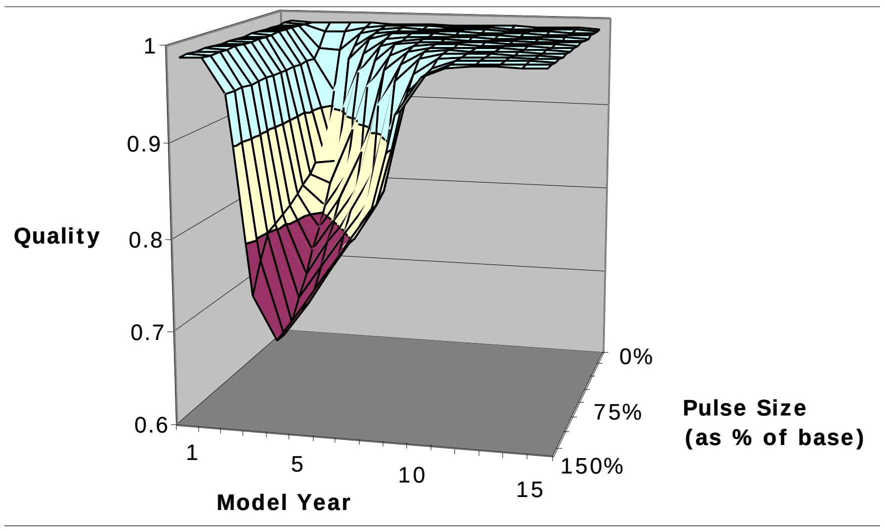

Figure 14: 3-Dimensional Surface for Quality

With a Temporary Increase in Workload in Model Year 1

Under the"Urgent First" Allocation with Resource-Constrained Testing

As the response surface shows, the temporary increase in workload causes a dramatic initial decline in quality, much larger than under the textbook allocation scheme. Although the effects of the one-time increase in workload manifest in lower quality for a number of model years, in contrast to the textbook scheme, the system does eventually recover to its initial performance level, regardless of the size of the one-time shock. Thus, with the alternative resource allocation scheme and no prototype builds, the system no longer has a second, lower performance equilibrium.

The long-term stability does, of course, come at a significant short-run cost. When the workload increases, quality falls far more dramatically than under the prior scheme. Few real organizations would be willing to suffer this dramatic drop in performance in the interest of long-term quality. Such a decline might drive away customers long before the system recovered its ability to produce high-quality output. While we are not recommending this policy, it does yield some insight into the dynamics of the system. Most important, it highlights the paradoxical role that testing (whether resource-based or accomplished by prototype builds) plays in determining performance.

On the one hand, testing is absolutely necessary to ensure integrity of the final product. No manager in good conscience can ignore that. Yet, on the other hand, once problems in parts are identified, they drive allocation of resources to fix problematic parts nearest launch. Although under this scenario problem-solving/revising is the highest priority activity, in these simulations few engineering hours are devoted to revising parts—simply because no testing or prototype build has identified any problems. Therefore engineers move to the next-highest priority job, designing parts. After design of all current phase parts for one model year is completed, engineers work on completing design of the following model year’s parts during its early phase.

Because early phase work generates fewer defects, as engineers complete a higher percentage of design in the early phase of model years following the one-time increase, quality improves, and eventually engineers can perform the testing and rework, as well as design, to ensure 100 percent quality in output.

In contrast, a scenario with “free” and/or higher priority testing actually strengthens the dominant positive loop and makes the system less resilient to shocks that cause permanent declines in performance. Such undesirable behavior in this system arises from the fact that problems in the downstream phase pull resources into less effective, lower-productivity activities as the organization strives to complete the project nearest launch. Of course, this steals resources from more effective upstream activities, creating a vicious cycle of declining competence. But allocating resources to a downstream problem requires having identified that such a problem exists. Thus, as testing takes on a higher priority or becomes more available, downstream problems appear more quickly and so steal more resources from more effective upstream activities.

As mentioned in the introduction, at many organizations, including the one we studied, project managers often complain of discovering major design problem late in the development cycle, often just a month or two before the product's scheduled launch. Paradoxically, this behavior is at least in part created by the search for problems. By testing for defects late in the development cycle, engineers create the possibility of finding defects that then require resources to correct them. In the organization we studied, the late discovery of defects, which often jeopardized the annual launch, was often followed by a call for increased investments in testing capability. The logic is that improved testing capability can allow defects to be discovered sooner, when there is time to do something about them. Yet this analysis suggests that increasing the efficacy and priority of testing in the upstream phase can improve the situation—provided that design is sufficiently complete in the early phase for testing to identify problems—while increasing the efficacy and priority of testing in the downstream phase can make the problem worse. Successfully identifying a defect, then, requires at least three conditions be satisfied. First the organization needs a level of resources sufficient to accomplish much of the design early enough in the development cycle to make increased testing capability worthwhile. Second, the organization needs testing capabilities adequate for the size model year it seeks to launch. Third, resources must be available to perform the actual testing procedures. Improving "testing capability" without commensurately increasing available resources suggests that more resources will still gravitate to downstream problems, reducing attention given to upstream projects, causing more incomplete or defective parts to be present when the project reaches the current phase. Completing more of a model year's design in the current phase creates more defects, and these additional defects are inevitably discovered late in the development cycle. Compensating for late defect discovery by improving testing capability provides a tenuous solution at best.

Thus the relationship between testing and performance is more complicated than perhaps initially appreciated. Testing is undoubtedly necessary, but maximizing its use requires a set of policies different from those considered so far. In what follows we experiment with several options.

Identify and Fix Only "Important" Defects

With some insight into the role of testing from the previous set of experiments, we now turn attention to other policies that might produce more robust system behavior. It is clear from the experiments above that a policy focused on fixing all downstream defects, while critical to maintaining product integrity in the short run, can create some undesirable long-run consequences. This suggests that if it were possible to ignore some of the least significant downstream defects, system performance might improve. In fact, this policy is actually used in many organizations. Problems discovered late in the development cycle are often categorized and addressed in a pre-established order (e.g., consumer safety issues receiving first priority). Any user of popular software packages for personal computers is probably well aware that products are often shipped before every defect is corrected. This scheme amounts to fixing a fraction of the total number of defects identified.

While we do not differentiate between types of defects in this model, we can test the dynamic consequences of this policy by simply assuming that the organization can instantly and costlessly identify the "important" errors. In the experiments shown below, we assume that 80 percent of the downstream defects can be ignored without endangering customers' safety. If the policy of allocating resources to fix only 20 percent of identified problems does not yield improved system robustness under these favorable assumptions, then its utility in any real situation is suspect, since it is not always easy to distinguish more significant from less significant defects. Figure 15 shows the result of this experiment.

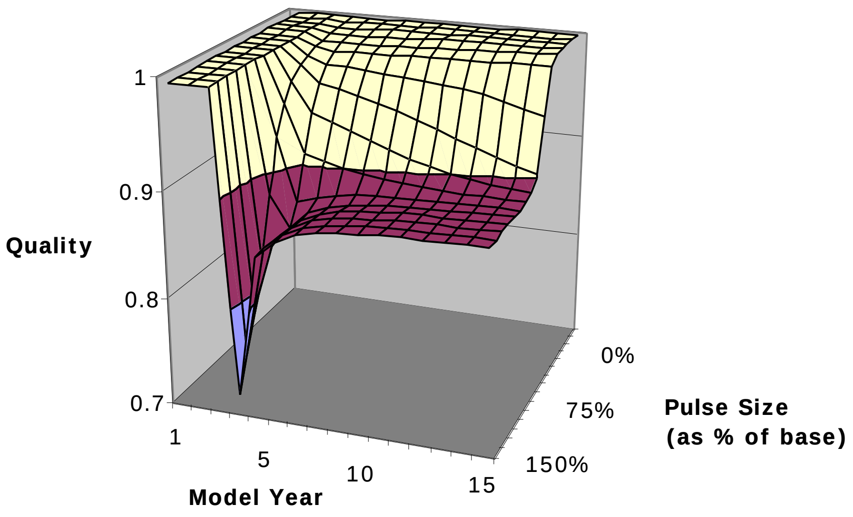

Figure 15: 3-Dimensional Surface Showing Quality

Under a Variety of Temporary Increases in Workload

When Resources Are Allocated to Solve 20 Percent of the Identified Problems

The surface plot suggests that, while the steady-state performance is marginally improved over the base case, this policy does not eliminate the possibility of becoming trapped in a low performance equilibrium. Furthermore, the improved steady-state performance comes at the expense of a slower recovery from the initial drop in quality to the second equilibrium. The impact of this policy can be best understood by first recognizing that it represents something of an intermediate option between the "textbook" and the "urgent first" resource allocation schemes discussed earlier. By ignoring defects in parts close to launch, some resources are shifted towards the next model year’s early phase, but meanwhile those unaddressed defects reduce final product quality. Thus, the "ignoring problems" policy presents managers with a trade-off: The more defects are ignored, the more long-run performance improves, but the more slowly that improvement manifests itself. If managers are in fact able to differentiate between important and unimportant defects, and the unimportant ones do not seriously compromise user safety or the firm's reputation, then this may be a viable policy for improving system performance.

Postponing Incomplete Work

While fixing a fraction of the possible defects was not a realistic alternative at the company we studied, one policy that was occasionally suggested and sometimes used was postponing projects. Often this decision emerged from a project review meeting that took place between four and eight months before the originally scheduled launch date. There was, however, some controversy concerning this policy’s efficacy. Several project teams complained that these postponement decisions were made too close to the scheduled launch, and others pointed out that postponing efforts did not always solve the project's problems and sometimes even exacerbated them. "Delaying projects doesn't solve the resource shortage," asserted one group. In one case, the decision to postpone a project was made after engineering resources for the *next* model year had already been allocated among projects, and the team whose problematic project was delayed again faced continuing problems with no resources available to solve them.

To understand the dynamic consequences of this proposal, we introduce a checkpoint during the current phase (the timing of which can be varied). At the checkpoint, all parts not yet designed are removed from the model year in the current phase and added to the model year in the early phase. For simplicity's sake, only the parts not yet designed are removed; parts that have been prototyped but that contain errors are not postponed. Because we assume that postponing parts incurs no cost to productivity and that there is no interdependence among parts, the simulated results of postponing should appear much rosier than what a company might actually experience. Again, however, if the policy fails to produce improved results under the conditions most favorable to its success, then it is probably worth little additional study.

Figure 16 shows quality of the products that reach the market under a policy of postponing all parts not yet designed six months prior to their launch date.

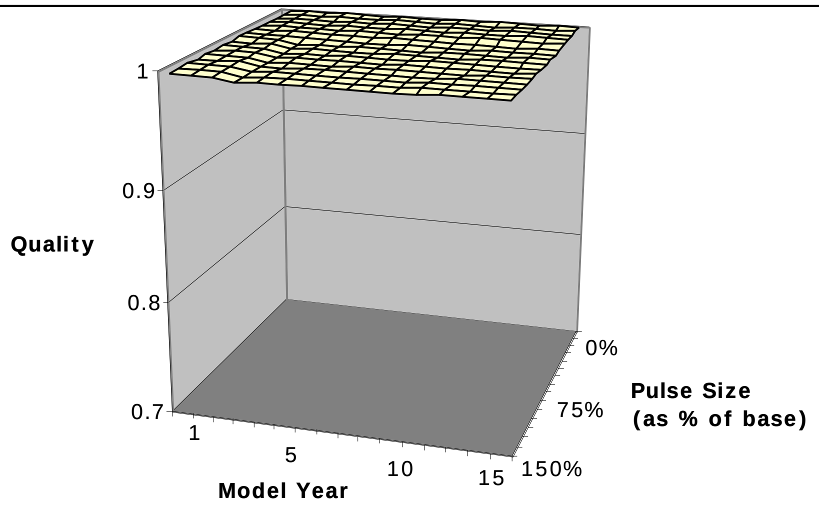

Figure 16: 3-Dimensional Surface Showing Quality

Under a Variety of Temporary Increases to Workload

When Incomplete Parts Are Postponed Six Months Before Launch

As the response surface shows, the policy is highly effective in maintaining the quality of the product in the face of a changing workload. With this policy in place, however, quality is no longer the sole measure of performance. Whereas in prior simulations it was assumed that all parts intended for a particular model year actually launched (albeit with poor quality), with the postponement policy in place, the percentage of the originally planned model year that actually enters the market can change from year to year. To capture this change, in Figure 17 we plot the behavior of a new variable, *performance*, which is calculated by multiplying the fraction of the model year launched by the parts' quality at launch.

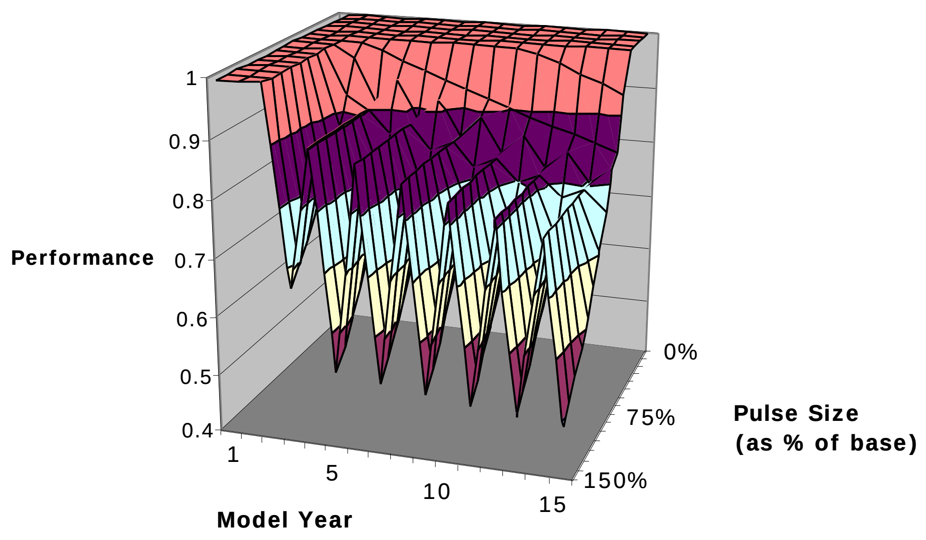

Figure 17: 3-Dimensional Surface Showing Performance

Under a Variety of Temporary Increases to Workload

When Incomplete Parts are Postponed Six Months Before Launch

Once we account for the change in model year content, the postponement policy appears less effective. Following the shock, performance falls in an oscillating fashion over an extended period of time. The sustained decline results from the continuing build-up of work-to-do in the early phase. The greater the shock, the greater the number of parts postponed. These postponed parts are added to the next model year, thus spreading the few resources devoted to early phase work across even more parts. As a consequence, at the following year's checkpoint, even fewer parts are completed, more work is postponed, and the fraction of the intended model year that is actually completed (the base workload of 750 parts plus the parts postponed from the previous year) declines. Performance oscillates from year to year because postponing many incomplete parts in one year lowers performance for that year but then frees more engineers to work on the early phase of the subsequent model year, which allows the next year's launch to include a larger number of parts. Even so, the trend in performance declines throughout the 15 simulated years. As more parts are delayed, performance (as measured here) degrades further, and the workload grows increasingly unwieldy. Clearly, our formulation exaggerates the impact of postponement. Few organizations would allow the stock of work-to-do to increase indefinitely. The dynamic that these simulations highlight, however, is very real. In the organization we studied, the number of projects in progress was so large that no single person within the organization could provide a complete list. More than once engineers delivered products to prototype builds to the surprise of senior managers and others, who hadn't known that these parts were included among the model year's changes. Thus a postponement policy, while effective in maintaining quality, has a serious and undesirable side effect.

Although the results are not graphically presented here, we also experimented with moving the postponement checkpoint to three months and one week before launch. Because only the parts not-yet-designed are postponed (rather than parts that are designed but have errors), the later checkpoints reveal a situation in which more parts are prototyped and fewer are postponed. When the postponement decision is made three months before launch, performance (as measured here, quality times the fraction of the model year launched) does not erode as sharply as when the checkpoint is six months before launch. Quality, however, suffers more, though it doesn't decline as much as when no parts are postponed. When the checkpoint is one week before launch, the simulation is identical to the one in which no postponement policy exists.

Canceling Incomplete Work

Finally, we turn to a policy of canceling incomplete work. Similar to the postponement policy, the canceling policy requires introducing a checkpoint at which the number of parts in current phase not yet prototyped are assessed. Under a cancellation policy, however, rather than moving the incomplete parts to the early phase of the following model year, they are eliminated. Figure 18 shows the series of simulations resulting from a checkpoint each year 6 months before launch. Again, performance, rather than quality, is used as a proxy for the health of the product development system.

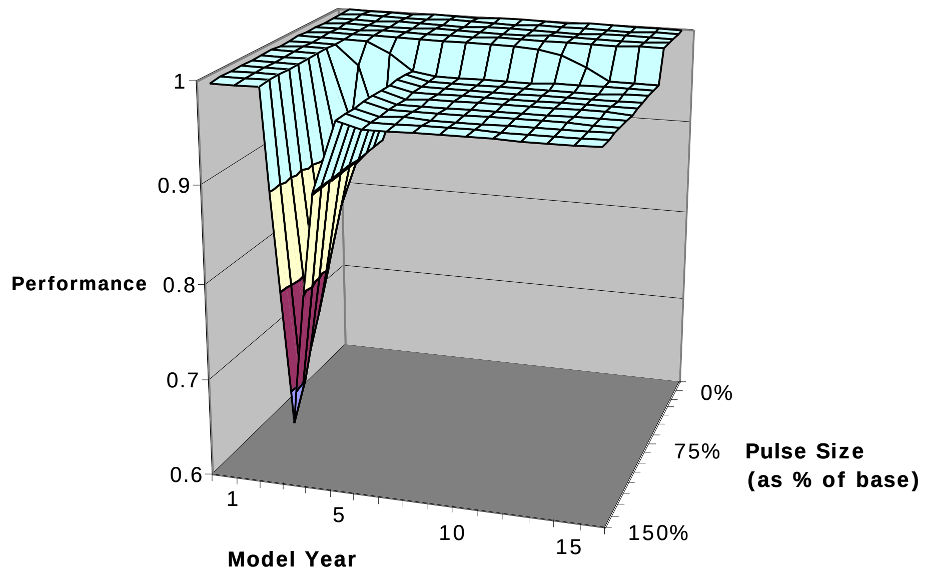

Figure 18: 3-Dimensional Surface Showing Performance

Under a Variety of Temporary Increases to Workload

When Incomplete Parts are Canceled Six Months Before Launch

For the largest shock in workload, the year that the high-content model year launches, more than 600 parts are canceled, and performance drops accordingly, to below 70 percent. Although the system begins to recover immediately, for increases in workload exceeding about 70 percent of the usual number of parts, engineers are never again able to perform as much work during the early phase as they were before the temporary increase in work. Even though the temporary imbalance between workload and resources tilts the product development system into a lower-performance regime, the lower performance equilibrium is considerably higher than in other scenarios (96 percent, rather than below 90 percent).

Again, when we look only at the quality of the work launched (ignoring the percentage of the intended model year completed), we see that quality remains high (Figure 19).

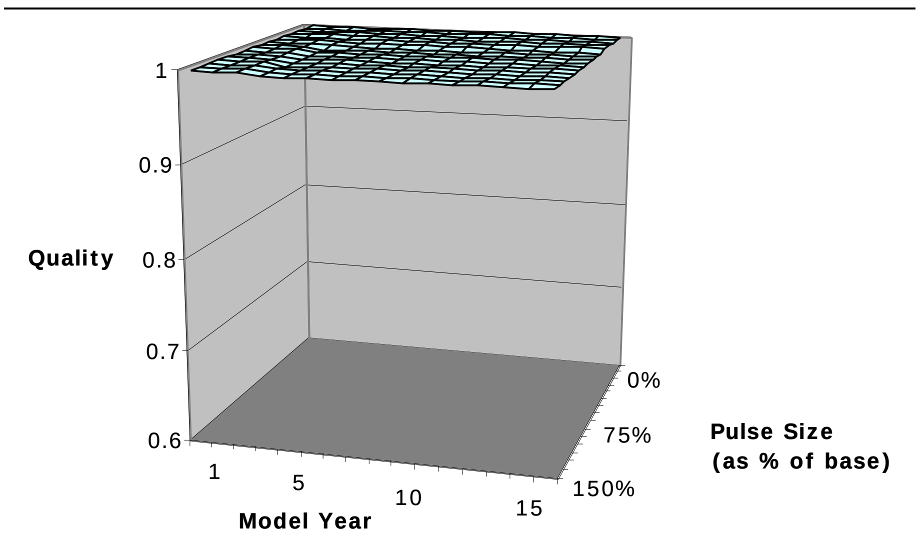

Figure 19: 3-Dimensional Surface Showing Quality

Under a Variety of Temporary Increases to Workload

When Incomplete Parts are Canceled Six Months Before Launch

Based on the benefits that a six-month cancellation policy provides, we now test the system's performance when the checkpoint occurs twelve months before launch. Under this scenario, when work from the early phase is moved to the current phase, parts that were not designed during the early phase are removed from the model year scheduled to launch in twelve months and eliminated from the new product plan. Figures 20 and 21 show the results for quality and performance.

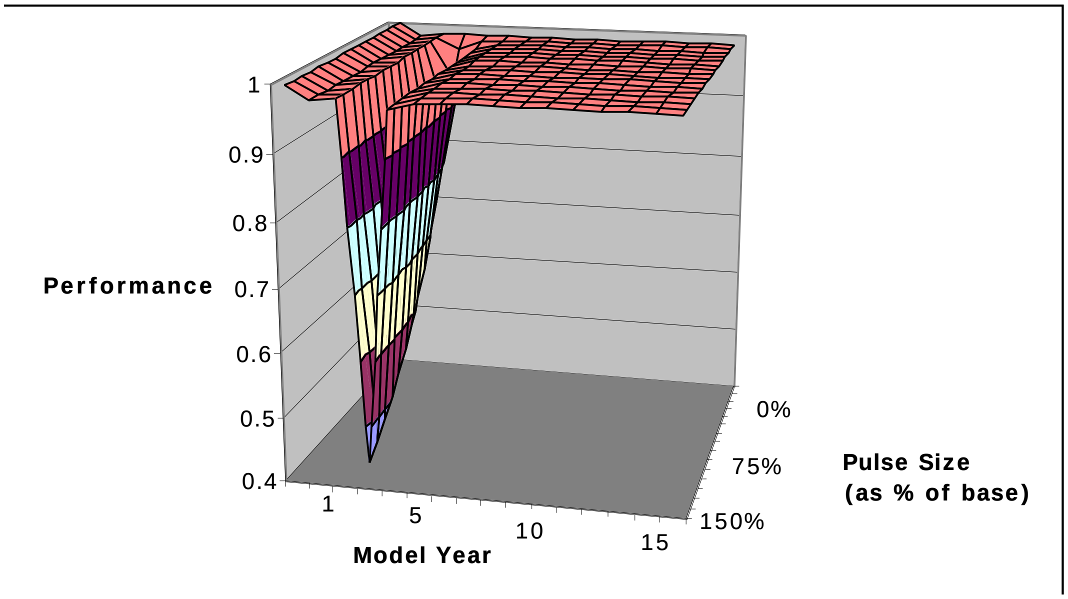

Figure 20: 3-Dimensional Surface Showing Performance

Under a Variety of Temporary Increases to Workload

When Incomplete Parts are Canceled Twelve Months Before Launch

With a twelve-month cancellation policy in place, temporary increases in workload cause a significant dip in performance (see Figure 20). The reduction comes solely from reduced content, and quality remains high (see Figure 21). Further, and more significantly, after the imbalance, the product development system’s recovery is immediate and enduring.

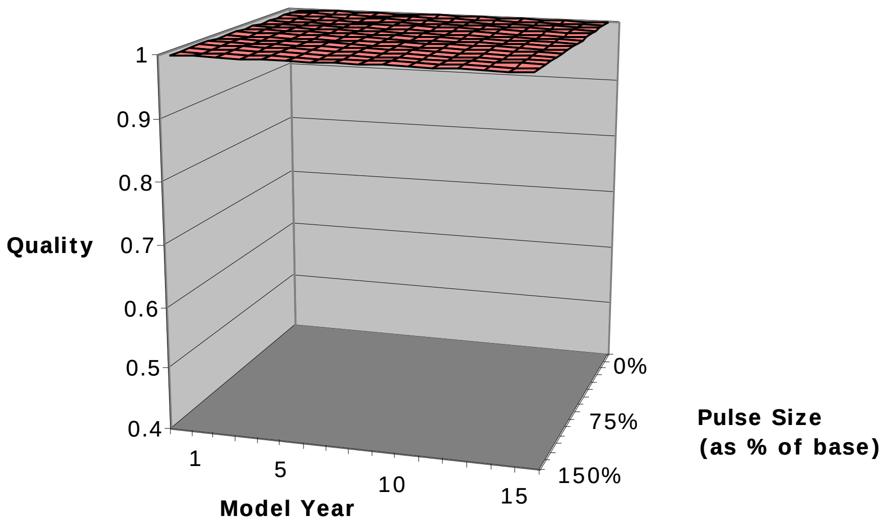

Figure 21: 3-Dimensional Surface Showing Quality

Under a Variety of Temporary Increases to Workload

When Incomplete Parts are Canceled Twelve Months Before Launch

To understand this policy, we present another response surface in which we plot the steady-state performance of the system (i.e., each point represents the final performance value of a particular simulation), while varying both the size of the temporary increase and the date of checkpoint at which incomplete parts are canceled.

Any checkpoint ensures an equilibrium value of performance as good as or better than that of the base scenario. Clearly, however, the timing of the checkpoint and decision-making makes a significant difference in the long-term outcome: The earlier checkpoints assure long-term performance better than later checkpoints—but a checkpoint can be too early. In this model, if the decision-making checkpoint occurs more than about 45 weeks before launch, performance begins to decline below the observed maximum of 99.9 percent.

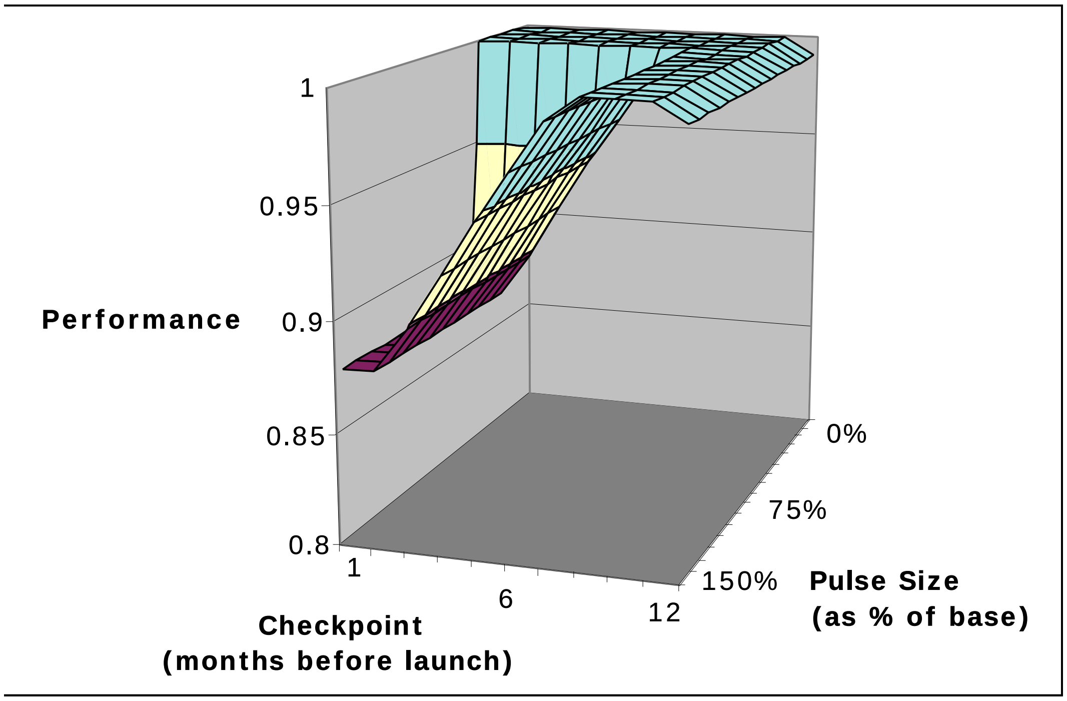

Figure 22: Steady-State Equilibria for Performance,

When the Checkpoint Timing is Varied

The checkpoint for making the decision to cancel part of a new product launch serves as a thorough reassessment of work-in-process and resources available to complete it. Although in the model we cancel work to restore balance, it would be just as possible to add resources. In practice, however, obtaining additional resources can pose more problems than simply canceling projects or scaling down scope of work underway. Hiring, either temporarily or permanently, skilled resources who can contribute significantly in a short time to projects already underway proves a challenging task even in abundant labor markets. Furthermore, although reneging on certain features or models that marketing and others depend on may have unpleasant consequences, those consequences are likely not as long-lasting as those resulting from decisions to deal with surplus employees. The checkpoint invites dynamic allocation of resources and work to maintain or restore balance to the product development system. The policy of canceling work that falls behind schedule well in advance of its launch date seems the only robust mechanism for ensuring consistently high performance and recovering productive capability after a temporary imbalance between work and resources.

Thus, we summarize our policy conclusion by saying that to maintain a stable, high-performing development system, effective testing must be complemented with effective cancellation of problematic projects.

6. Discussion

In this paper we tackle the problem of explaining why, in product development projects, major problems are so often discovered late in the development cycle. To paraphrase a quotation from the introduction, we were interested in explaining why, at both the firm under study and in many similar organizations, managers of development projects so often don't know what they don't know until it’s really late. The principal contribution of this analysis is to demonstrate that testing plays a paradoxical role in multi-project development environments. On the one hand, it is absolutely necessary to preserve the integrity of the final product. On the other hand, in an environment where resources are scarce, allocating additional resources to testing or to addressing the problems that testing identifies leaves fewer resources available to do the up-front work that prevents problems in the first place in subsequent projects. Thus, while a decision to increase the amount of testing can yield higher quality in the short run, it can also ignite a cycle of greater-than-expected resource requirements for projects in downstream phases, fewer resources allocated to early upstream phases, and increasingly delayed discovery of major problems.

The model from which this insight arises has limitations, of course. Most significant, the framework we propose represents an abstraction from the details of a real product development process. We aggregate all the components of a specific product into a single category, "parts"; the myriad activities required to create a product are considered simply "engineering"; and, while most of the development processes discussed in the literature contain either four or five phases, our model contains only two. Nevertheless, while the model is exceedingly simple, relaxing some of its most extreme assumptions would likely strengthen rather than weaken our main conclusions.

Representing the quality of a product as simply the sum of its respective parts neglects the considerable problems that can arise when two or more parts, which themselves are not defective, are inconsistent and thus create a defect between them. Such "system" or "sub-system" defects are quite common and often the source of major product problems. Extending our model to account for this category of problems would likely strengthen the conclusions drawn here, since many activities that should be done in the early phases of the development cycle are targeted at preventing just these types of defects. System-level errors identified and corrected *prior* to the detailed design of specific parts require fewer resources to correct than those discovered after much of the part-level design has been completed.

Similarly, while we have considered only one type of resource (engineers), explicitly accounting for a range of capabilities is also likely to worsen the situation. The argument is similar to that made above. Many problems do not fall into the domain of a single expertise. Thus, properly executing the early phases of a process necessitates the participation of numerous engineers with different skill-sets. If even one of these engineers is not available (perhaps because she is engaged in fixing problems on a downstream project), the effectiveness of the upstream phase is compromised. Thus, the system becomes more prone to the downward spiral we identify since, if even one resource is over-taxed and unable to complete early phase work, the entire system suffers the rework that becomes necessary in later phases.

Finally, if a development process has more than two phases, this, too, is likely to exacerbate the risk of "tilting," since there are then multiple opportunities for the dynamics we observe here. Specifically, while we focus in this paper on the relationship between upstream and downstream activities in product design and engineering, this relationship exists in multiple places in a development process. Establishing appropriate functions and features of a product is often more effective when performed upstream, by market research, than downstream, by expensive engineering prototypes. System-level design problems are more easily and cheaply resolved prior to the component design phase, and manufacturing and assembly problems are more easily and cheaply corrected when identified during the component design phase rather than after design completion or during production tooling. Thus, within any development process, there are multiple opportunities for resources and workload to become unbalanced due to unforeseen requirements in downstream phases. Styling experts spend their time on correcting poorly conceived products instead of on creating new and better ones; engineers become mired in fixing system-level problems in existing designs rather than developing better methods for preventing them; and manufacturing engineers end up focusing all their energy on correcting existing problems and tools rather than participating in the design process in order to forestall such problems in the future.

While the model captures only a small portion of the complexity of any real multi-product development environment, the features represented capture an important set of dynamics that play a critical role in determining overall system performance. Thus, given that these dynamics seem pervasive and are apparently experienced in a range of settings, we turn to the question, What should managers do about them? There are three complementary strategies identified by this and previous analyses. First, as highlighted by Wheelwright and Clark (1992), most development environments are well served by carefully matching projects to available resources. It is now well documented that most organizations have too many projects in progress; in such situations, the dynamics we identify are virtually assured to occur. Second, Repenning (2000) continues this line of reasoning by arguing that, while necessary, appropriately matching resources to known requirements is not sufficient. Developing new products is a fundamentally uncertain task and, while a firm may develop a product plan that matches resources to *known* requirements, any unanticipated problems that require additional resources can cause a permanent decline in system performance. Third, as highlighted by this paper, organizations can benefit from both frequent reassessment the workload resource balance and aggressive cancellation of problematic projects early in the development process.

There are, however, major challenges inherent in improving the resource allocation process. First, it is clear both from the field study on which this paper is based and from other studies reported in the product development literature that admonitions to "do fewer projects" and "cancel more," though often heard in R&D organizations, rarely if ever have appreciable impact. Although people may understand and subscribe to these admonitions at an abstract level, when decisions must be made on a day-to-day basis concerning specific projects, these general guidelines are simply inadequate to combat the deeply seated cognitive biases created by experience uninformed by a structural understanding of the system in which they work. Recall the first set of simulations. In the first model year, the system ably accommodates an increased workload, delivering the additional parts at a high quality. A manager making such a decision probably receives powerful and salient feedback that the increase was a good thing to do. Unfortunately, the short-run gain comes at the expense of long-run performance. In subsequent model years, performance progressively declines—even though workload returns to normal. Because the decline in performance occurs only with a significant delay, it is much less likely to be associated with the earlier increase in workload in any manager’s mind. This creates a strong bias for managers to focus on "doing just a little more."

Second, even if managers seek to rebalance workload to resources by canceling projects, another problem arises: Nobody likes having her project canceled. Engineers and project managers are rewarded for delivering projects, not for having them canceled. During project reviews there is a strong tendency to overstate the level of project completion and understate the resources required to complete the project adequately. Managers don’t always receive accurate information concerning the state of projects in the pipeline, thus creating additional uncertainty and further reducing the likelihood that they will take the difficult step of canceling projects.

Given the strength of managers' biases and the incentives facing project managers, we believe that a particular strict and inflexible version of the cancellation policy offers the highest potential to produce significant improvement in product development. Specifically, we propose that managers establish a policy of allowing only a fixed number of projects into the detailed design phase and, more important, uphold it by canceling those projects that are not allowed to proceed past this checkpoint.

Why might such draconian measures be needed to produce the desired outcome? First, as previously highlighted, the psychological biases and institutional structures that create the pathologies we studied here are deeply seated and unlikely to change soon. A more flexible version of the cancellation policy could be subject to modifications by managers and have its efficacy eroded over time. In other words, if managers are given discretion to increase the number of projects, they are likely to use it, thus putting the system in the undesirable situation from which it started in the “pulse” tests of the model. As many studies of performance in simulations games (e.g. The Beer Game) demonstrate, simple decision rules often outperform the complicated, contingent decision rules we humans often construct, particularly when the structure of an environment creates counterintuitive dynamics.

Furthermore, such a policy might prove an effective method of forcing an organization to develop its competence in evaluating projects mid-cycle as well as in performing thorough up-front work. There is an additional reason, so far not discussed, why organizations so often fail to do up-front work: They don't know how. Organizations that do not invest in up-front activities are unlikely ever to develop significant competence in executing them. Thus, the decision to change the balance of resources not only necessitates reducing the number of projects; it may also require a significant investment in learning. Individual managers and project leaders are unlikely to make such investments for fear of incurring the inevitable but temporary decline in performance. More senior managers have few methods to force people to seriously undertake such learning activities.

We believe the policy we propose has the potential for reversing many of the feedbacks that otherwise work against higher performance. Managers who are encouraged to cancel projects on an annual basis will need to make these decisions based on some criteria. Project managers will soon learn that to survive the cut, they must demonstrate the likelihood that their project will succeed. In contrast to the situation we observed, in which there are few incentives to do up-front work, in a world where some projects will *always* be canceled, project managers may find strong incentives to invest in early phase activities and develop clever ways to test the efficacy of a proposed product far in advance of detailed design. Similarly, while project managers may always portray their specific projects in the kindest light, more senior managers may have strong incentives to develop more objective methods of determining which projects are allowed to proceed into the detailed design phase.

As mentioned in the introduction, while single-project models comprise a well-developed strand in the system dynamics literature, models of multi-project development environments are in their infancy. Following the example of single-project models, an appropriate next step might be to develop increasingly detailed, situation-specific analyses of multi-project environments. Eventually such an approach may develop, again following the example of single-project models, into full-scale decision support systems that managers can use to evaluate the health of their product portfolios in real time.

Bibliography

Abdel-Hamid, T.K. 1988. "The Economics of Software Quality Assurance: A Simulation-Based Case Study," *MIS Quarterly* 12(3): 395-411.

Adams, S. 1996. *The Dilbert Principle.* New York: HarperBusiness.

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

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.

Ford, D. N. and J. D. Sterman. 1998. "Dynamic Modeling of Product Development Processes," *System Dynamics Review* 14 (1): 31-68.

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

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.

Moxnes, E. 1999. "Misperceptions of Bioeconomics," *Management Science*, 44 (9):1234-1248.

Repenning, N. 2000. "A Dynamic Model of Resource Allocation in Multi-Project Research and Development Systems," *System Dynamics Review* 16 (3).

Repenning, N. P. and J. D. Sterman. 2000. " Getting Quality the Old-fashioned Way: Self-confirming Attributions in the Dynamics of Process Improvement," in R. Scott and R. Cole, Editors, *Improving Research in Total Quality Management*. Newbury Park, Ca: Sage.

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

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

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

Walton, M. 1997. *Car: A Drama of the American Workplace.* New York: W.W. Norton.

Wheelwright, S.C. and K.B. Clark. 1992. *Revolutionizing Product Development: Quantum Leaps in Speed, Efficiency, and Quality*. New York: The Free Press.