Understanding Fire Fighting

in

New Product Development

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: <nelson@mit.edu>

March 2001

Forthcoming in

Journal of Product Innovation Management

Work reported here was supported by the MIT Center for Innovation in Product Development under NSF Cooperative Agreement Number EEC-9529140, the Harley-Davidson Motor Company and the Ford Motor Company. Extremely valuable comments have been provided by Laura Black, Andrew Jones, John Sterman, Scott Rockart, Rogelio Oliva, Steven Eppinger, Steve Graves, John Hauser, Rebecca Henderson, two anonymous referees and the editor. Thanks also to seminar participants at MIT, Wharton, Harvard, Stanford, and INFORMS Seattle. Minsoo Cho developed the web site that accompanies this article. Special thanks to Don Kieffer of Harley-Davidson for providing the catalyst for this study.

Complete documentation of the model as well as more information on the research program that generated this article can be found at <http://web.mit.edu/nelsonr/www/>.

Biographical Sketch

Nelson P. Repenning is an assistant professor at the MIT Sloan School of Management. His work focuses on understanding the factors that contribute to the successful implementation, execution, and design of business processes. Current research interests include organizational change, process improvement in new product development, and the creation of cross-disciplinary management theory. His work draws on a number of modeling methods including simulation, non-linear dynamics, and game and contract theory. He is currently working with MIT's Center for Innovation in Product Development. Address: E53-339, 30 Wadsworth Street, Cambridge MA 02142, <nelson@mit.edu>, <http://web.mit.edu/nelsonr/www>.

Understanding Fire Fighting in New Product Development

Abstract

Despite documented benefits, the processes described in the new product development literature often prove difficult to follow in practice. A principal source of such difficulties is the phenomenon of fire fighting–the unplanned allocation of resources to fix problems discovered late in a product's development cycle. While it has been widely criticized, fire fighting is a common occurrence in many product development organizations. To understand both its existence and persistence, in this article I develop a formal model of fire fighting in a multi-project development environment. The major contributions of this analysis are to suggest that: (1) fire fighting can be a *self-reinforcing* phenomenon; and (2) multi-project development systems are far more susceptible to this dynamic than is currently appreciated. These insights suggest that many of the current methods for aggregate resource and product portfolio planning, while necessary, are not sufficient to prevent fire fighting and the consequent low performance.

Introduction

The design of effective product development processes has received considerable attention from scholars and practitioners [9,37,38,39]. Unfortunately, however, practice does not necessarily follow theory. Many organizations experience considerable difficulty in following the development processes prescribed in the literature, and mounting evidence suggests that in many organizations the desired development process and the sequence of tasks actually used to create products are two very different things. For example, Griffin [16] reports that, despite widespread acceptance in the literature, almost 40% of firms surveyed still use no formalized development process. Smaller sample studies also report similar results. O'Connor [24] studies the efforts of six organizations in depth and finds that no organization in his sample was able to gain the full benefit of a new development process, even after, in some cases, three years of continuous effort. Similarly, Repenning and Sterman [28], Oliva, Rockart, and Sterman [25], Jones and Repenning [18], and Krahmer and Oliva [21], all document cases in which, despite general agreement concerning their benefits, organizations struggled to follow the development processes prescribed in the literature. One engineer in a study by Repenning and Sterman [28] aptly summarized the dilemma facing many organizations when, in describing his experience with a newly instituted development process, he said, “The [new process] is a good one. Some day I’d like to work on a project that actually uses it.”

Unfortunately, while the academic literature has made numerous contributions to understanding how product development should work, less attention has been paid to the question of why organizations so often fail to execute their development processes as desired. Similarly, while there are numerous frameworks to assist managers in designing new development processes, the prescriptive literature also offers relatively little in terms of tactics and strategies to insure that engineers actually follow that process when developing new products [24 is a notable exception]. The history of management contains numerous examples of innovations like Total Quality Management that, despite documented benefits, failed to significantly influence practice in the

majority of organizations that tried to implement them (see [13] for a discussion of TQM and [20] for an overview). To avoid a similar fate, the considerable advances in process *design* must be complemented with an improved understanding of the factors that contribute to effective process *execution* in New Product Development.

In this article, I study one of the most common and well-documented impediments to successful process execution in new product development, the phenomenon of *fire fighting.* The metaphor of fighting fires is widely used in the management literature, typically referring to the allocation of scarce resources to solve unanticipated problems or "fires." In the product development context, fire fighting describes the unplanned allocation of engineers and other resources to fix problems discovered late in a product's development cycle. Fire fighting imposes numerous costs on the project that requires it: Introduction dates are often slipped, reducing the chance of market success; engineers and managers sometimes work extraordinary hours, leading to fatigue, burnout, turnover, and increasing the chance of further errors; and additional people are often added to the project, thus requiring additional expense. Despite these costs, however, it is not surprising that fire fighting occasionally occurs; developing new products is a fundamentally uncertain task, often involving numerous unproven technologies and processes.

There is, however, a growing sense that, in many organizations, fire fighting is more than an infrequently occurring phenomenon confined to individual projects. Detailed field studies of various product development organizations suggest that fire fighting, rather than being isolated to specific projects, often becomes the de facto process for developing new products. For example, one manager in a study by Repenning [29] described the state of affairs in his organization by saying:

> ...if you look at our resource allocation on traditional projects, we always start late and don't put people on the projects soon enough...then we load as many people on as it takes...the resource allocation peaks when we launch the project.

Similarly, in an ethnographic study of a copier manufacturer, Perlow [26] identifies fire fighting or "crisis management" as both a steady state phenomenon and a principle source of low productivity.

The existence of fire fighting as a steady state rather than a temporary phenomenon limits the ability of organizations to properly execute their NPD processes in a number of ways. Most importantly, while studies of successful projects almost universally confirm the value of investing in the early phases of the development process [10,11,17], organizations engaged in fire fighting find it extremely difficult to make such commitments, thus making it impossible to follow the development processes described in the literature. As one manager in the Jones and Repenning [18] study said:

> …the completion date [the date at which the project is ready to launch] is getting later and later each year. We are starving the ensuing model years to make the one we are on. We never have time to do a model year right, so we have lots of rework and so on.

Similarly, organizations that dedicate large portions of their available development resources to fixing unanticipated problems also find it difficult to successfully implement NPD tools and processes. As one engineer in Repenning's [29] study of a failed process change effort said:

> To be perfectly honest, I really don't think [the new process] changed the way engineers did their jobs. In many ways we worked around the system. Good, bad, or indifferent that's what happened. We had a due date and we did whatever it took to hit it.

That fire fighting constitutes a serious impediment to performance in many product development environments is not a new insight. It has been vilified in venues ranging from scholarly studies [26] to practitioner-oriented publications [30] and self-help books [12]. Such widespread consensus on its evils, however, makes its ubiquity all the more puzzling: Everybody agrees that fire fighting is detrimental to performance, yet, paradoxically, it persists.

To study the causes and consequences of fire fighting, in this article I propose and study a dynamic model of a multi-project development environment. By capturing the dynamics of resource allocation among competing projects in different phases of the development process, the model

yields a number of insights into the existence and persistence of fire fighting. Most importantly, it suggests that fire fighting can be a self-reinforcing phenomenon; once it starts in one project, it is likely to spread to others, permanently degrading the capability of the development system. The analysis also suggests that, due to a combination of structural and psychological factors, multi-stage, multi-project NPD processes are far more susceptible to this phenomenon than is currently appreciated.

To develop these insights along with their implications for both future practice and research, the rest of the article is organized as follows. The next section contains a brief overview of the model and a detailed justification of its core assumptions. The following section contains the analysis. I then take up the question of why fire fighting is so persistent and conclude with implications for future research and practice.

The Model

Overview

Figure 1 shows the basic structure of the model (an equation by equation description is provided in appendix A). The simulated organization introduces one new product to the market each *model year*, and two model years are required to develop a product. A new project is introduced into the system at the product introduction date, so at any moment in time two projects are under development. Since the hallmark of fire fighting is the unplanned allocation of resources to fix problems in the later phases of the development cycle, I decompose the development process into two phases. While most authors suggest development processes that contain four or five stages [e.g. 9,37,38], in the interest of simplicity, I assume there are only two, the *concept development* phase and the *product design and testing* phase. I divide the process at this juncture because, as many authors point out, the concept development and product design phases involve fundamentally different activities: concept development work is not focused on the design of the actual product, but, instead, on making the subsequent design work more productive.

****

Insert Figure 1 about here

****

Within each phase there is a set of tasks to be completed. The set *D* of design tasks represents those activities required to physically create the product. The set *C* of concept development tasks represents those activities that, by reducing the probability of introducing a defect, make subsequent design work more effective. Only one type of resource, engineering hours, is needed to accomplish both types of tasks. It is important to note that this structure does not require that each phase last exactly one year, only that each phase be no longer than one year. In fact, in the base run of the simulation, activity only begins with three months remaining in the concept development phase, roughly corresponding to estimates of the balance between up and downstream work in current NPD practice [15].

Figure 2 shows a more detailed view of the model in the form of a stock and flow diagram. In such diagrams stocks (or levels) are denoted by rectangles and represent accumulations of "stuff" (designs, parts, defects, etc.). Flows are shown by arrows with "valve" symbols and represent action or activity within a system. Solid arrows depict how other variables in the system influence the flows. For example, the upper half of the figure shows the assumed stock and flow structure of the concept development phase. At the outset of each model year, a new set of tasks is introduced into the concept development phase. Initially, these tasks reside in the stock of *Concept Development Tasks Remaining*. As resources are dedicated to concept development, the stock of tasks remaining is drained by the flow of *Concept Development Task Completion*. When those tasks are completed, they accumulate in the stock of *Concept Development Tasks Completed***.**

****

Figure 2 about here

******

The design and testing phase has a slightly more complicated structure. Once a project reaches the design and testing phase, there is a P_D probability that each task is done incorrectly and thus

requires rework and additional resources. So, at the outset of the model year, all design tasks reside in the stock of *Design Tasks Remaining*. As resources are applied to design work, the tasks are executed and then accumulate in the stock of *Design Tasks Awaiting Testing*. Tasks are then tested. If they pass, they flow to the stock of *Design Tasks Completed*. If they fail, they accumulate in the stock of *Design Tasks Awaiting Rework.* For simplicity, testing, rather than being a separate phase, is assumed to take place concurrently with design. In the model, testing is represented as an uncapacitated delay, meaning it takes time, but consumes no resources. Additional analysis, not reported here, demonstrates that adding a resource constraint on testing, while considerably complicating the analysis, only strengthens the conclusions outlined below [see 3]. When resources are applied to rework, tasks leave the rework stock and flow back into the stock of *Design Tasks Awaiting Testing*.

The solid black arrows at the right of the diagram represent the connection between the concept development and design phases. The central assumption of the model is that the probability of introducing an error in the design phase is a function of how many tasks were completed when that project was in its concept development phase. Thus, given the assumed timing, the probability of finding an error in the design phase in *this model year* is a function of the number of concept development tasks completed in the *previous model year*. As the diagram highlights, this structure introduces a critical time delay in the development system. The execution of concept development work does nothing to improve the quality of the product currently in the design phase. Instead, the impact of additional effort dedicated to concept development this year only manifests in the subsequent model year, when the product in question reaches the design and testing phase. The equation used to capture this dependence is as follows:

$$P_D(s)=P_\alpha + P_\beta(1-f(s-1)) \qquad (1)$$

In this equation, the variable s indexes the model year and the variable $P_D(s)$ represents the probability of making an error in a design task in model year s. P_α represents the portion of the

defect fraction that cannot be eliminated by doing concept development work. P_{β} is the portion of the defect fraction that can be eliminated by doing up front work and *f(s-1)* is the fraction of concept development work completed in model year *s-1*. The equation indicates that the probability of introducing an error in model year *s* is a function of the amount of concept development work completed in the previous year, *s-1*.

Figure 3 adds to the previous diagram by showing the feedback structure determining the allocation of resources between the two phases. As the figure highlights, the system contains three important feedback loops. The first two, B1 and B2, are negative or balancing loops controlling the rates of design task and design rework completion. In both cases, as the stock of outstanding work increases, more resources are allocated to increase the respective completion rates, thus reducing the stock of outstanding work. These loops regulate the level of remaining work by adjusting the allocation of resources.

The third is a positive or reinforcing feedback loop (labeled with an R). If the resources required for design work decrease, then more concept development tasks are completed, and the completion fraction rises. In the next model year, fewer tasks are done incorrectly, and the resource requirement in the design phase decreases further. Here, the loop works as a virtuous cycle. In contrast, however, if the resource requirement in the design phase increases, then fewer resources are dedicated to concept development work, thereby reducing the task completion rate and, ultimately, the completion fraction. If the completion fraction declines, in the next model year the defect rate grows, more rework is generated, more resources are required for downstream work, and even fewer concept development tasks are completed. When operating in this direction, the loop manifests as a vicious cycle of declining attention to the upstream portions of the development cycle and increasing error rates in downstream work.

Central Assumptions

Three core assumptions are thus embodied in the model's structure. First, executing additional concept development tasks is assumed to improve the effectiveness of downstream work. Formally, this is captured in equation (1), in which an increase in $f(s\text{-}1)$ reduces the defect fraction $P_D(s)$. The feature of new product development that I am trying to capture is the existence of activities in the early phases of a project which, when resources are scarce, can be skipped, but whose primary effect is to increase the effectiveness of downstream tasks. An example of such a task is the documentation of customer requirements. Additional effort dedicated to understanding customer needs often pays substantial benefit when a product reaches the detailed design phase, improving the chance that initial design efforts will be well received by the market. Such up-front tasks are, however, often skipped [10]. A weak or non-existent understanding of what the customer actually wants greatly increases the chance that early designs are not well received, thus generating additional rework. The validity of this assumption is supported by both the prescriptive writings of those who suggest concept development as an important first step in a product development process [9, 38] and by extensive empirical study [5,11,17].

The second key assumption is that, when resources are scarce, priority is given to the project in the design and testing phase. This allocation rule represents an incentive scheme that was described by one project manager (in the Repenning and Sterman [28] study) as “Around here the only thing they shoot you for is missing product launch, everything else is negotiable.” There are at least three reasons why organizations, sometimes despite their members' best intentions, might use such a policy. First, there is the firm’s reputation and continued viability. Doing a concept development task, which prevents rework in future projects, does nothing to fix the problems in the project currently in the design phase, which may compromise the safety of its users and/or irreparably damage the company’s reputation. Second, in organizations focused on short-term profit and cash flow, projects nearing completion represent a more immediate return on investment. Even in cases where it might be in the organization's best interest to abandon a project, the well-known and amply documented sunk cost fallacy [2,31,32] suggests that managers will continue investing in

projects well beyond the point of economic return. Third, even absent reputation and financial constraints, managers may still be biased towards design tasks due to more basic features of human cognition such as ambiguity aversion [14] and biases towards salient information [27,28].

The third assumption, that products are introduced and launched at fixed intervals, is made for computational and expository convenience. In many industries, this assumption closely mirrors actual practice. US auto manufacturers typically introduce new vehicles on an annual cycle, as do manufacturers of recreational products such as motorcycles and snowmobiles. Further, even in industries where the launch date can, in principle, be moved, competitive pressures often dictate a fixed introduction date. For example, firms that manufacture semi-conductors often face fixed market windows dictated by the introduction of the next generation of a consumer product (e.g. cell phones, personal computers, etc.). In these cases, the costs of late delivery are so high (ceding essentially all market share to competitors) that as a practical matter, firms must deliver to a specific launch date. The value of this assumption is that it allows me to analyze the performance of the system via one output variable, the quality of the finished product.

Analysis

Base Case

To highlight the dynamics of the assumed system, I begin with the model's *base case*. The base case represents the behavior that the system produces without the introduction of shocks or other interventions. The chosen parameters are shown in Table 1. Figures 4 and 5 show the behavior of selected variables.[i]

***
Table 1 about here
****

In the early portion of the model year all development resources are dedicated to the design phase and the stock of design tasks remaining declines rapidly (Figure 4b). As design tasks are completed, the stock of rework begins to grow (Figure 4b), and resources are shifted towards completing rework. Multiple iterations through the rework cycle reduce the fraction of defective tasks in the downstream product (Figure 5a). As the stock of outstanding rework declines,

resources are moved to the project in the concept development phase (Figure 5b), and, consequently, the stock of concept development tasks begins to decline (Figure 4a). As the end of the model year approaches, some rework remains uncompleted so resources are shifted back towards design work (Figure 5b).

****

Figures 4a and 4b about here

****

****
Figures 5a and 5b about here
*****

In the base case the simulated organization completes the vast majority of its planned concept development work and only engages in a modest amount of fire fighting (as shown by the increased allocation of resources to the design phase at the end of the model year).

Response to Shocks

Given this desirable mode of operation, the question naturally arises, what might cause the system to descend into fire fighting? Two simulation experiments provide an instructive entry into this question. In the simulations shown below, in model year one, the number of tasks required per project is increased by 20 percent and 25 percent respectively and then, in subsequent model years, returned to the base level. Such a transient increase might arise, for example, from an attempt at a particularly ambitious or complex product.

*****
insert Figure 6 a and b about here
*****

In both cases, the fraction of concept development tasks completed falls following the increased workload (Figure 6a). With less attention dedicated to concept development, projects experience more problems in the design phase and the quality of the final product begins to suffer (Figure 6b). In the case of the 20% increase, once the workload is returned to normal in the subsequent model

year, the concept development completion fraction begins to rise and the quality of the finished project also improves. Here, while the shock creates some fire fighting, it does not spread, and the system eventually recovers to its initial performance level and execution mode.

The larger shock, however, produces a different outcome. In the model years following the increase (in which the workload is returned to normal), the system does not recover. Instead, the completion fraction continues to fall and the defect rate continues to rise. Following the shock, the system, rather than returning to its initial execution pattern, settles into a new mode in which little concept development work is done and the performance is substantially degraded. In this case, the increased workload traps the system in a low performance regime.

The Source of Persistent Fire Fighting

Although the system is relatively simple, its non-linear structure makes it difficult to analyze using the traditional tools of linear systems analysis. So, to understand *why* the system produces such divergent behavior, I use a two-step strategy to analyze the model. The first step builds on the fact that, while *within* any given model year the system's dynamics can be complicated due to its higher order structure, only one variable, *f(s)*, carries over *between* model years (this dependence is captured in (1)). Thus, by making suitable approximations to the within-year dynamics, the model can be reduced to a one-dimensional map in *f(s)*. Some straightforward analysis of this equation then yields a number of insights into the dynamics of multi-project development systems. In the second step, extensive simulation runs of the full model (most of which, in the interest of space, are not presented here) are used to confirm the validity of the approximations.

The details of simplifying the model are discussed in appendix B. The critical steps are to eliminate the delay in testing by assuming all defects are discovered immediately and to make corresponding changes in the desired completion rates. The resulting equation is:

$$f(s) = Min\left(1, \frac{1}{C}\ Max\left(K\ T - \frac{D}{1-\left(P_{\alpha} + P_{\beta}\ \left(1 - f(s-1)\right)\right)}, 0\right)\right) \quad (2)$$

This equation captures the dynamics of process execution in a multi-project development system by relating the fraction of concept development tasks completed in a given model year (represented by *f(s)*) to the amount of work in the system (represented by *C* and *D*), the annual capacity to do that work (*K*•*T*), and how the process was executed in the previous model year (represented by *f(s-1)*). Viewing this equation in graphical form, as a *phase plot*, provides a useful perspective on the structural causes of fire fighting (see figure 7).

****
Insert figure 7 here
*****

To read the phase plot, start at any point on the horizontal axis, read up to the solid black line and then over to the vertical axis. So, for example, suppose that, in a given model year, the organization manages to accomplish about 60 percent of its planned concept development work, what happens next year? Reading up and over suggests that, if it accomplishes 60 percent of the up-front work this year, the dynamics of the system are such that about 70 percent of the up-front work will get done next year. Determining what happens in a subsequent model year requires simply returning to the horizontal axis and repeating; accomplishing 70 percent this year leads to almost 95 percent being accomplished in the year that follows. Continuing this mode of analysis shows that, if the system starts at any point to the right of the solid black circle in the center of the diagram, over time the concept development completion fraction will continue to increase until it reaches 100%. Here, the positive loop (R) works as a *virtuous* cycle: Each year a little more up front work is done, decreasing errors and, thereby, reducing the need for resources in the downstream phase. With a fewer resources required in the design phase, even more effort can be dedicated to concept development work. As this cycle continues, the system converges to the point where, each year, the organization accomplishes all its desired up-front work and thus is able to deliver consistently high-quality products.

In contrast, however, consider another example. Imagine this time that the organization starts to the left of the solid black dot and accomplishes only 40 percent of its planned concept development activities. Now, reading up and over, shows that instead of completing more early phase work in the next year, the organization completes less—in this case only about 25 percent. In subsequent years, the completion fraction declines further, creating a *vicious* cycle of declining attention to up-front activities and increasing error rates in design work. In this case, the system converges to a mode in which concept development work is ignored in favor of fixing problems in the downstream project.

The phase plot thus reveals two important features of the system. First, note from the discussion above that anytime the plot crosses the forty-five degree line (meaning that $f(s)=f(s-1)$) the execution mode in question will repeat itself. Formally, at these points the system is said to be in *equilibrium*. Practically, equilibria represent the possible "steady states" in the system, the execution modes that, once reached, are self-sustaining. As the plot highlights, this system has three equilibria (highlighted by the solid black circles), two at the corners and one in the center of the diagram.

Second, also note that the equilibria do not have identical characteristics. The equilibria at the two corners are *stable*, meaning that small excursions will be counteracted. If, for example, the system starts in the desired execution mode ($f(s)=1$) and is slightly perturbed, perhaps pushing the completion fraction down to 60%, then, as the example above highlights, over time the system will return to the point from which it started, mainly $f(s)=1$. Similarly, if the system starts at $f(s)=0$ and receives an external shock, perhaps moving it to a completion fraction of 40%, then it will also eventually return to its starting point. The arrows on the plot line highlight the "direction" or trajectory of the system in disequilibrium situations. In contrast to those at the corners, the equilibrium at the center of the diagram is *unstable* (the arrows head "away" from it), meaning small excursions are not counteracted. Instead, once the system leaves this equilibrium, it does not

return and instead heads toward one of the two corners. Given its instability, it is unlikely that the system will ever settle at the interior equilibrium. Despite this, however, it plays a critical role in determining the dynamics of this system and is central to understanding the source of persistent fire fighting in NPD.

Formally, the unstable equilibrium represents the boundary between two basins of attraction. If the system starts at any point *below* the unstable equilibrium, the positive loop works in a vicious direction and moves the system towards *f(s)=0*; if the system starts from any point *above* the unstable equilibrium, the positive loop works in a virtuous direction and moves the system towards *f(s)=1*. This boundary, or *tipping point*, plays a critical role in determining the system's behavior because it is the point at which the positive loop (R) *changes* direction. If the system starts in the desirable execution mode and then is perturbed, if the shock is large enough to push the system over the tipping point, it does not return to its initial equilibrium and desired execution mode. Instead, the system follows a new downward trajectory and eventually becomes trapped in the fire fighting equilibrium.

The role of the tipping point in determining the system's dynamics is highlighted by the simulation experiments shown above (see figure 6). The twenty-percent increase in workload is *insufficient* to push the system over the tipping point, so the initial shock is counteracted and the system returns to its starting point. In contrast, however, the twenty-five percent increase is *sufficient* to push the system over the boundary, reversing the direction of the positive loop, and causing the system to descend into a vicious cycle of fire fighting. And, as the experiment highlights, once the system reaches the lower equilibrium, absent an additional intervention, it never recovers.
In fact, the difference between recovery and permanent fire fighting is even smaller than these experiments suggest. Figure 8 shows the results of a more comprehensive set of simulations in which the model has been run for a large number of shock sizes.

****

figure 8 about here

*****

As the figure highlights, up to a shock of approximately 20%, the system recovers to its initial performance level. Beyond that, however, the system rapidly descends into fire fighting. For these cases, the fire created by the increased workload rapidly spreads to subsequent products, permanently reducing the performance of the development system.

This characterization of the system's structure–two stable equilibria separated by a tipping point–yields at least two insights into the existence and persistence of fire fighting in multi-project NPD system. First, the existence of a stable equilibrium at $f(s)=0$ suggests that fire fighting can be a steady-state phenomenon. Practically, this means that a fire fighting execution mode characterized by little attention to the early phases of the development cycle and a consequent focus on rework can become the *de facto* development process. Indeed, as the analysis highlights, inadequate attention to the early portions of the development process creates a self-reinforcing cycle that traps the system in a permanent state of unbalanced and undesirable resource allocation.

Second, the existence of a tipping point suggests that even when the system starts in the desirable execution mode, there is no guarantee that it will persist. Shocks to the system such as overly ambitious projects, additional projects, or the discovery of major errors can push the system over the tipping point into a downward spiral of fire fighting and undesirable process execution. Importantly, these shocks do not have to be longstanding changes. While a number of scholars have highlighted the non-linear costs associated with permanently overloading a development system [1,37], this analysis suggests that development systems are even more fragile. A *temporary* shock, if it is large enough to push the system over the tipping point, can cause a *permanent* decline in performance.

When is a Development System Prone to Fire Fighting?

The model thus provides one explanation for the existence and persistence of fire fighting: multi-project development environments can have tipping points, which once crossed, cause fire fighting to become a self-reinforcing, self-sustaining phenomenon. Assessing the relevance of this characterization to actual practice requires, however, understanding the conditions required for its existence. Thus, before considering the model's implications, it is useful to establish the conditions under which its core results are likely to occur. Some additional analysis of the reduced form model answers this question and provides additional insight into the dynamics of process execution.

Consider first the conditions required for the existence of the stable equilibria. At the upper equilibrium $f(s)=f(s-1)=1$ in every model year, so, substituting $f(s-1)=1$ into equation (2) indicates that for $f(s)=1$ to be an equilibrium, the second term inside the minimum function of (2) must be greater than or equal to 1. Thus, the existence of an equilibrium in which the process is executed as designed requires:

$$K \cdot T \geq C + D/(1 - P_{\alpha}) \qquad (3)$$

This relation has an intuitive interpretation. The term on the left-hand side, $K \cdot T$, represents the annual capacity to do development work. The term on the right-hand side represents the number of tasks required to develop a defect-free product when executing the process as desired. To see this, recall that executing the process as desired requires that all concept development tasks be completed (thus, the C term) and that, as a consequence, the defect rate is at its minimum, P_{α}. Also note that every time a design task, whether it is new work or rework, is executed, there is a P_{α} probability that it is done incorrectly. If the organization continues doing design tasks and correcting defects until none remain, the total number of tasks required to produce a defect free product asymptotically approaches $D/(1-P_{\alpha})$. Equation (3) indicates that for the system to have the potential to operate in the desired mode (meaning $f(s)=1$), annual capacity, $K \cdot T$, must be greater than or

equal to that number of tasks. Thus, (3) suggests that operating the process as designed requires enough resources to complete all of the activities that the given process requires.

Similarly, consider the conditions required for the "maximum fire fighting" mode, *f(s)=0*, to be an equilibrium. Substituting *f(s-1)*=0 into (2) shows that the existence of this equilibrium requires that the first term inside the maximum function is less than or equal to zero. So, for the maximum fire fighting mode to persist the following must hold:

$$K \cdot T \leq D \Big/ \left(1 - \left(P_{\alpha} + P_{\beta}\right)\right) \tag{4}$$

This equation has an interpretation similar to the previous one. The left-hand side again represents annual capacity. The right-hand side represents the number of tasks required to develop a defect-free product when no work is done in the concept development phase and, as a consequence, the defect rate is at its maximum. Thus, this relation suggests that, for the fire-fighting mode to persist, capacity must be insufficient when operating in that mode.

Taken together, these two conditions lead to four distinct parameter combinations (both are satisfied, both are not, (3) is satisfied while (4) is not, and (4) is satisfied while (3) is not). The phase maps that result from these four combinations, which are shown in figure 9, fully characterize the dynamics of the reduced system.

****
insert figure 9 about here
*****

Consider first the phase plot in the upper left quadrant (figure 9). In this case neither of the two conditions are satisfied. Capacity is *adequate* when fire fighting but *inadequate* when executing the process as desired. Under these conditions, the system has a unique, stable equilibrium; when shocked it will constantly return to its starting point. Formally, under these conditions the positive feedback loop shown in Figure 3 does not *dominate* the system's behavior. Intuitively, this means

that, under these conditions, fire fighting is not a self-reinforcing phenomenon. It may be experienced on individual projects for various reasons, but it will not spread. Additional insight into the conditions required for this case can be gained by combining equations (3) and (4) to yield the following inequality:

$$D/\left(1-\left(P_{\alpha}+P_{\beta}\right)\right) \leq C + D/\left(1-P_{\alpha}\right)$$

As the relation shows, for this case to hold, the total number of tasks needed to develop a defect-free product when all the concept development tasks are completed must be greater than the total number of tasks needed to develop a defect-free product when concept development activities are ignored. Thus, this case only holds when concept development tasks require more resources to complete than they save via the reduced defect fraction. Given the substantial empirical support for the value of the early phases of the development cycle [10,11], this is the least interesting of the four cases.

The second case is shown in the upper right quadrant. Here (3) is satisfied and (4) is not, implying that resources are *sufficient* regardless of the execution mode. Under these conditions, the system has a unique equilibrium at *f(s)=1*. In this situation, the positive loop (R) dominates the behavior of the system, but always works as a virtuous cycle, constantly driving the system towards the desired execution mode. In contrast to qualitative discussions [26,30,37], this case suggests that, even if the up-front tools are "worth it", fire fighting is not necessarily a self-reinforcing phenomenon. If resources are sufficient, regardless of the state of the development process, then any departure from the desired execution mode will be offset by a reinforcing cycle of decreasing error rates in the design phase and sustained investment in concept development activities. While this appears to be a highly desirable situation, note that it requires that total capacity, K•T, be *greater* than the workload regardless of whether or not concept development tasks are completed. This leads to a somewhat ironic result; insuring that the desirable operating mode prevails requires a level of resources that completely negates the efficiency gains that the desirable operating mode provides.

In contrast, when (4) is satisfied and (3) is not, implying that resource are *insufficient* regardless of the execution mode, the system again has one equilibrium, but it is located at *f(s)=0* (see the lower left quadrant in figure 9). Here the positive loop works only as a vicious cycle. Importantly, in this case external shocks are not necessary to ignite the fire-fighting dynamic. Instead, due to the lack of resources, the system is in a constant downward spiral of declining attention to upfront activities and increasing rework in the design phase.

This case both confirms and extends the existing intuition concerning the role of resources. Numerous scholars have argued that overloading a development process degrades its performance. For example, Wheelwright and Clark [37] suggest that overloading a development process creates a system that is susceptible to undesirable self-reinforcing dynamics because, they write, "…if any one project runs into unexpected trouble, there is no slack available, and it will be necessary to take resources from other projects. This causes subsequent trouble on other projects and the effects cascade." While this is true in the fourth case, case number three suggests that if resources are sufficiently scarce, then undesirable self-reinforcing dynamics *will* occur. Here the system has only one possible trajectory: a downward spiral of increasing error rates in downstream work and decreasing investment in upstream activities.

Finally, as shown in the lower right quadrant, in order for the system to have the structure discussed above– two stable equilibria separated by a tipping point– resources must be *sufficient* when executing the processes as desired, but *insufficient* when fire fighting. Combining this analysis with previous empirical studies suggests that the first two cases are unlikely to occur, since up front activities are typically "worth it" [10,11] and development systems rarely have substantial excess resources [37]. In contrast, the dynamics highlighted in cases three and four, which occur whenever resources are, to varying degrees, scarce, capture important features of actual development practice.

Beyond establishing the conditions under which each case will arise, this analysis also highlights an additional and important feature of the system: the location of the tipping point, when it exists, is determined by resource utilization. As resource utilization increases, the tipping point moves up the forty-five degree line and approaches the upper equilibrium at $f(s)=1$. When equation (3) is satisfied with equality, the tipping point and the upper equilibrium converge. If it is increased any further (so (3) is no longer satisfied), both the upper equilibrium and the tipping point disappear and case three obtains. Similarly, as resource utilization falls, the tipping point moves down the forty-five degree line until it reaches the lower equilibrium (at this point equation (4) is satisfied with equality). If the workload is further reduced, both the tipping point and the lower equilibrium disappear and only the upper equilibrium remains. The distance between the tipping point and the upper equilibrium is, thus, a function of the balance between work and resources, implying that there is an important trade-off between steady state performance and the ability to absorb unplanned increases in workload. As resources utilization is increased, progressively smaller shocks are necessary to push the system over the tipping point and into a vicious cycle of fire fighting.

Summary

The results of this analysis can be summarized in two insights. First, multi-project development systems can have multiple equilibria, implying that identical processes can produce dramatically different results depending on the mode of process execution. Further, these equilibria are stable, implying that incremental interventions are unlikely to improve performance. Thus, it is quite possible for NPD systems to get permanently "stuck" in the fire fighting execution mode. Second, the susceptibility of a system to fire fighting is determined by resource utilization. If resources are totally adequate regardless of the execution mode, then fire fighting is always a self-correcting phenomenon. If resources are totally inadequate, then fire fighting is always a self-reinforcing phenomenon. In the intermediate cases, the size of the shock required to push the system in to the

self-reinforcing regime is a function of the balance between resources and work: as utilization increases, progressively smaller shocks are required to push the system over the tipping point.

Why Does the Phenomenon Persist?

The behavior studied in this article is created by the interaction between the physical structure of the product development process and the decision rules used by participants within that process. At the outset the decision rules were justified by empirical observations and well-documented decision making biases. Each of these justifications is, however, essentially static. Thus, while it might be the case that managers would initially allocate resources in such a fashion, wouldn't they eventually learn to overcome these dynamics? In other words, why is fire fighting so persistent? The answer rests on two more basic questions: does the structure of this system support rapid and accurate learning, and, once caught in such a downward spiral, to what do managers attribute the cause of low performance?

To answer to the first question, consider the outcome feedback a manager receives from the decision to allocate additional resources to a project experiencing problems late in its development cycle. As the model shows, in a world of scarcity, this decision has two consequences. First, additional resources lead to improvements in the *project* to which they are allocated. This outcome and the decision that led to it are contiguous in both time and space and the impact is relatively easy to characterize. Second, the decision to allocate resources to a downstream project, if it initiates the fire-fighting dynamic, degrades the performance of the product development *system*. In contrast to the improved performance of the downstream project, this outcome occurs with a significant delay, the decision and the outcome are not closely linked in space, and the characterization of its impact is very ambiguous. Further, the product development system is less salient, less tangible, and more ephemeral than the products it produces. Thus, managers making resource allocation decisions in multi-project development systems are faced with a "better-before-worse" trade-off in which the positive, but transient, consequence of the decision happens quickly and is easy to

assess, while the negative, but permanent, consequence occurs only with a delay and is difficult to characterize.

In such situations, the literature on human decision making has reached a clear conclusion: people do not learn to manage such systems well [34]. In experiments ranging from managing a simulated production and distribution system [35] to fighting a simulated forest fire [4] to managing a simulated fishery [23], subjects have repeatedly been shown to grossly overweight the short run positive benefits of their decisions while ignoring the long run, negative consequences. Participants in such experiments produce wildly oscillating production rates and inventory levels, they allow the fire fighting headquarters to burn down, and they kill the fishery through over-fishing. Applying these results to the product development context suggests both that managers will be biased towards downstream projects and that they will not learn to overcome the undesirable dynamics that such a bias creates.

The problem is, however, worse than the experimental results might suggest. Once caught in a downward spiral, managers must make some attribution of cause. The psychology literature also contains ample evidence suggesting that managers are more likely to attribute the cause of low performance to the attitudes and dispositions of people working within the process rather than to the structure of the process itself (an example of the widely documented Fundamental Attribution Error, see [27]). Thus, as performance begins to decline due to the downward spiral of fire fighting, managers are not only unlikely to learn to manage the system better, they are also likely to blame participants in the process. To make matters even worse, the system provides little evidence to discredit this hypothesis. Once fire fighting starts, system performance continues to decline even if the workload returns to its initial level. Further, managers will observe engineers spending a decreasing fraction of their time on up-front activities like concept development, providing powerful evidence confirming the managers' mistaken belief that engineers are to blame for the declining performance.

Finally, having blamed the cause of low performance on those who work within the process, what actions do managers then take? Two are likely. First, managers may be tempted to increase their control over the process via additional surveillance, more detailed reporting requirements, and increasingly bureaucratic procedures. Second, managers may increase the demands on the development process in the hope of forcing the staff to be more efficient. The insidious feature of these actions is that each amounts to increasing resource utilization and makes the system more prone to the downward spiral. Thus, if managers incorrectly attribute the cause of low performance, the actions they take both confirm their faulty attribution and make the situation worse rather than better. The end result of this dynamic is a management team that becomes increasingly frustrated with an engineering staff that they perceive as lazy, undisciplined, and unwilling to follow a pre-specified development process, and an engineering staff that becomes increasingly frustrated with managers that they feel do not understand the realities of the system and, consequently, set unachievable objectives.

Implications for Research and Practice

The most important result of this study for researchers is, then, to suggest that managing multi-project NPD systems effectively does not come naturally. The dynamics of multi-project systems coupled with basic features of human psychology make fire fighting a likely occurrence in many NPD environments. For scholars, this suggests that developing new decision support technologies that help managers fully account for both the *benefits* and *costs* of their actions represents a significant opportunity to influence and improve practice. NPD systems are prone to fire fighting in large measure because the benefits of focusing on downstream projects are realized long before the costs. Research in other domains suggests that decision support systems and "management flight simulators", which compress time through the use of simulation, can improve decision making in such dynamically complex environments (see [22] for an overview and [6] for an extended example).

Creating "fire-resistant" NPD systems requires the development of more *dynamic* methods of resource planning. To be effective, such systems must be capable of accurately assessing both the current state of the development process and, given that state, estimating future resource requirements. This suggestion echoes Cooper's [8] evaluation of Third Generation development processes. He writes, "Developing an information system to forecast resources committed to known projects is no easy task, but it is essential: only in this way are decision makers able to visualize what resources will be available to "new" projects, and what the impact of approving more projects will be on current and future resource availabilities." Cooper [8] also suggests that, currently, even best practice falls far short of this goal. Existing methods for portfolio management and making go/kill decisions often fail to accurately account for the total number of projects in progress and their associated resource requirements [7]. Only by allowing managers to "visualize" the state of their development processes, and then rapidly simulate the outcomes of various interventions, will the natural tendency towards fire fighting be offset. Thus, an important next step for NPD researchers is to develop more detailed, accurate models of resource consumption and allocation in multi-project development environments that might eventually provide the basis for the system that Cooper envisions.

For practitioners, this study suggests that, absent such a resource planning system, there are a number of intermediate steps to help minimize fire fighting and improve performance. First, as numerous authors have highlighted, there is no substitute for good aggregate resource planning. Managers who persistently overload their development systems are virtually guaranteed to experience persistent fire fighting. Also note that, while the existing literature suggests that overloading causes performance to decline due to the costs associated with switching between and spreading resources over multiple projects [37], this analysis suggests an additional cost of overloading: it changes the *mode* of process execution. Importantly, this implies that there is *not* a proportional trade-off between the quantity of projects produced and the long run capability of the development system. When doing resource planning, there is often a strong temptation to do "just

one more" project, figuring that an extra product this year will cost at most the loss of one product in the next model year. But such a strategy is premised on a faulty assumption. The relationship between the number of projects and overall system performance is highly non-linear. Increasing resource utilization increases efficiency until the system reaches its tipping point. Once over that point, however, capability rapidly declines due to changes in the mode of process execution. Adding an extra product in this model year (if the workload increase pushes the system over the tipping point) reduces the number of products released for many years to come.

Second, when doing resource planning, it is critical to assess resource requirements given the *current* mode of process execution, not that which would be required were the process operating as desired. Case studies suggest that all too often, senior managers have far too rosy a view of the development processes they oversee, frequently grossly overestimating their capabilities [28]. Unfortunately, assuming the process is operating in the high-performance mode, when it is in fact not, leads to the persistent under allocation of resources, thereby trapping the system in the fire-fighting mode.

Third, high quality aggregate resource planning, while necessary, is not sufficient to prevent fire fighting. At the outset of a given project, the organization may develop a plan that well matches resources to the known requirements. Projects, however, often require more resources than anticipated due to changes in scope or other problems. Such contingencies constitute transient increases in resource requirements capable of pushing the system over the tipping point and into the downward spiral of fire fighting. Practically, this means that absent the ability to perfectly forecast resource requirements, slack development resources provide a valuable buffer against fire fighting. By increasing the distance between the upper equilibrium and the tipping point, slack resources create a more robust, fire resistant development system. As the discussion above highlights, due to a variety of psychological and social factors the temptation to fully utilize

development resources can be quite strong, but doing so creates a very fragile development system.

Fourth, the existence of a tipping point implies that the cost of allowing troubled projects to enter down stream phases of the development process are often far greater than managers perceive. When an important project reaches an early phase gate with incomplete specifications, it is often tempting to let it pass in the belief that the team will "catch up" in the next phase. But, such decisions are usually made without an estimate of the true costs. Problems discovered late in the development cycle impose both the "direct" costs of additional resources and the "indirect" costs associated with pushing the development system into the fire-fighting mode. While the de facto decision heuristic often appears to be "when in doubt, let a project proceed", performance would often improve if the opposite rule were used, "when in doubt, do not let a project continue."

Fifth, the analysis strongly suggests that managers should not use resource scarcity as a change strategy. For example, a failure mode observed in a large scale product development improvement effort studied in [29] was the allocation of resources under the assumption that the development process was being properly executed and operating at full capability. This rule was not inadvertent, but an explicit part of the implementation strategy under the rationale that, if the resources were removed, to get their work done participants would have no choice but to follow the new, more efficient process. The flaw in this logic is that there were already projects in progress executed in the fire fighting execution mode, thus requiring more downstream resources than the new process dictated. The ensuing scarcity perpetuated the downward spiral and caused the failure of the initiative.

Finally, managers also should realize that managing such systems well doesn't come naturally. The potential downfall of any policy directed at eliminating fire fighting is that, to be effective, it must be followed. The ultimate source of fire fighting is the faulty mental models of those who set

resource levels and allocate those resources among competing projects. A manager who does not understand the dynamics discussed above and attributes the cause of low system performance to those working in the process is likely to find her organization trapped in the fire-fighting mode. Thus, above all, preventing fire fighting requires the discipline to resist the natural tendency to focus on specific projects and instead target interventions at maintaining the integrity of the development process. Only by focusing explicitly on the health and performance of its NPD *system* will an organization overcome the fire fighting dynamics discussed here.

References

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

2. Arkes, H. R., and Blumer, C. The psychology of sunk cost. *Organizational Behavior and Human Decision Processes 35*: 124-40 (1985).

3. Black, L. and Repenning, N. Why Fire Fighting is Never Enough: Preserving High-quality Product Development. *System Dynamics Review 17(1)*: (2001).

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

5. Burchill, G. and Fine, C. Time Versus Market Orientation in Product Concept Development: Empirically-Based Theory Generation. *Management Science 43(4)*: 465-478 (1997).

6. Carroll, J., Sterman, J. and Markus, A. Playing the Maintenance Game: How Mental Models Drive Organization Decisions. In: *Debating Rationality: Nonrational Elements of Organizational Decision Making*, R. Stern and J. Halpern (eds.). Ithaca, NY: ILR Press 1997. pp. 92-121.

7. Cooper, R. G., Edgett, S. and Kleinschmidt, E.J. New Product Portfolio Management: Practices and Performance. *Journal of Product Innovation Management 16*: 333-351 (1999).

8. Cooper, R.G. Perspective: Third-Generation New Product Processes. *Journal of Product Innovation Management 11*:3-14 (1994).

9. Cooper, R.G. *Winning at New Products.* Reading, MA: Addison Wesley. 1993.

10. Cooper, R.G. and Kleinschmidt, E.J. New Products: What separates winners from losers. *Journal of Product Innovation Management 4(3)*: 169-184 (1987).

11. Cooper, R.G. and Kleinschmidt, E.J. An Investigation into the New Product Process: Steps, Deficiencies and Impact. *Journal of Product Innovation Management 3*: 71-85 (1986).

12. Covey, S. *The Seven Habits of Highly Effective People.* New York: Simon and Schuster. 1989.

13. Easton, G. and Jarrell, S. The Effects of Total Quality Management on Corporate Performance: An Empirical Investigation. *Journal of Business 71(2)*: 253-307 (1998).

14. Einhorn, H. J. and Hogarth, R.M. Ambiguity and uncertainty in probabilistic inference. *Psychological Review 92*: 433-461 (1985).

15. Griffin, A. Product Development Cycle-Time for Business-to-Business Products. Working Paper: (2000).

16. Griffin, A. PDMA Research on New Product Development Practices: Updating Trends and Benchmarking Best Practices. *Journal of Product Innovation Management 14*: 429-458 (1997).

17. Gupta, A. and Wilemon, D. Accelerating the Development of Technology-Based Products, *California Management Review*: 24-44 (Winter 1990).

18.Jones, A.P. and Repenning, N. P. Sustaining Process Improvement at Harley-Davidson. Case Study available from second author, MIT Sloan School of Management, Cambridge, MA 02142 (1997).

19.Kahneman, D., Slovic, P., and Tversky, A. *Judgment under uncertainty: Heuristics and biases.* Cambridge: Cambridge Univ. Press 1982.

20.Klein, K. and Sorra, J. The Challenge of Innovation Implementation. *Academy of Management Review 21(4)*: 1055-1080 (1996).

21.Krahmer, E. and Oliva, R. A Dynamic Theory of Sustaining Process Improvement Teams in Product Development. In: *Advances in Interdisciplinary Studies of Teams* , Beyerlein, M. and D. Johnson (eds), Greenwhich, CT: JAI Press 1996.

22.Morecroft, J. and Sterman, J. (eds). *Modeling for Learning Organizations.* Portland, OR: Productivity Press 1994.

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

24.O'Connor, P. From Experience: Implementing a Stage-Gate Process: A Multi-Company Perspective. *Journal of Product Innovation Management 11*: 183-200 (1994).

25.Oliva, R., Rockart, S., and Sterman, J. Managing multiple improvement efforts: Lessons from a semiconductor manufacturing site. In: *Advances in the Management of Organizational Quality* . Fedor, D. and S. Ghosh (eds.). Greenwich, CT: JAI Press 1998. Pp. 1-55.

26.Perlow, L. The Time Famine. *Administrative Science Quarterly 44*: 57-81 1999.

27.Plous, S. *The Psychology of Judgment and Decision Making*, New York, McGraw-Hill (1993).

28.Repenning, N. P., and Sterman, J.D. Getting quality the old-fashioned way: Self-confirming attributions in the dynamics of process improvement. In: *Improving Research in Total Quality Management* . R. Scott & R. Cole (eds). Newbury Park, CA: Sage 2000. Pp. 201-235.

29.Repenning, N. P. Reducing Cycle Time at Ford Electronics Part II: Improving Product Development. Case Study Available from the Author (1996).

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

31.Staw, B. M. Knee-deep in the big muddy: A study of escalating commitment to a chosen course of action. *Organizational Behavior and Human Performance 16(1)*: 27-44 (1976).

32.Staw, B. M. The escalation of commitment to a course of action. *Academy of Management Review 6*: 577-587 (1981).

33.Sterman, J.D. *Business Dynamics*. Chicago, IL: Irwin-McGraw Hill. 2000.

34.Sterman, J.D. Learning in and about complex systems, *System Dynamics Review 10(3)*:291-332 (1994).

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

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

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

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

39. Zangwill, W. *Lightning Strategies for Innovation: how the world's best create new projects*, Lexington Books: New York, NY1993.

Appendix A

This appendix provides a more detailed description of the model's structure. The model year transitions are discrete and indexed by *s*. Within a given model year the model is formulated in continuous time indexed by *t.* Each model year is *T* time units in length. A product launched at the end of year *s* is composed of two sets of tasks, *C*, the concept development tasks, and *D*, the design tasks.

At any time *t* in model year *s*, the system is characterized by seven states (see figure 2). Within the concept development phase tasks have either been completed and reside in the stock *Concept Development Tasks Remaining, $C^r(t)$,* or remain to be completed and reside in the stock of *Concept Development Tasks Completed, $C^c(t)$.* In the design phase, tasks can either be in the stock of *Design Tasks Remaining, $D^r(t)$*, the stock of *Tasks Awaiting Testing, V(t)*, the stock of *Tasks Awaiting Rework, R(t)*, or have been completed and successfully passed testing, thus residing in the stock of *Design Tasks Completed, $D^c(t)$.* The final state, *f(s)*, represents the fraction of concept development tasks completed in model year *s*. At the model year transition, all states are reset to their initial values except for *f(s)*. Thus, $C^r(0)=C$, $D^r(0)=D$, and $C^c(0)$, $V(0)$, and $D^c(0)$ all equal zero.

Given these states, resources are allocated in the following order: first priority is given to design tasks that have yet to be completed; second priority is given to design tasks that have been completed but are found to be defective; and any remaining resources are allocated to concept development work.

The rate at which design tasks are completed, *d(t)*, is determined by the minimum of development capacity, K, and the desired design task completion rate, *d*(t).*

$$d(t) = Min(K, d^*(t)) \qquad \text{(A1)}$$

Tasks completed accumulate in the stock of tasks awaiting testing, *V(t)*. Testing is represented by an exponential delay so the stock of tasks awaiting testing drains at a rate of $V(t)/\tau_v$, where τ_v represents the average time to complete a test. Tasks either pass the test, in which case they accumulate in the stock of tasks completed, $D^c(t)$, or fail and flow to the stock of tasks awaiting rework, *R(t)*. The fraction of tasks that fail in model year *s* is $P_D(s)$.

The stock of outstanding rework, *R(t)*, is drained by the rework completion rate, *r(t)*. The rate of rework completion is equal to the minimum of available capacity to do rework, *K-d(t)*, and the desired rework completion rate, *r*(t)*:

$$r(t)=Min(K-d(t),r^*(t)) \quad \text{(A2)}$$

The rate of concept development task completion, *c(t)*, is equal to the minimum of the desired concept development task completion rate, *c*(t)*, and the remaining development capacity, *K-d(t)-r(t)*. Thus:

$$c(t)=Min(K-d(t)-r(t),c^*(t)) \quad \text{(A3)}$$

Concept development tasks are assumed to be done correctly.

The desired completion rates, *d*(t)*, *r*(t)*, and *c*(t)* are each computed in a similar fashion. In anticipation of future rework, engineers are assumed to try to complete each of these tasks as quickly as possible, so:

$$d^*=D^r/\tau_c \quad \text{(A4)}$$
$$r^*=R^r/\tau_c \quad \text{(A5)}$$
$$c^*=C^r/\tau_c \quad \text{(A6)}$$

The parameter τ_c represents the average time required to complete a task.

The only dependence between model years is captured in $P_D(s)$, the probability of doing a design task incorrectly. $P_D(s)$, is a function of *f(s-1)*, the fraction of the concept development tasks completed when the product was in its concept development phase:

$$P_D(s)=P_\alpha+P_\beta(1-f(s-1)) \quad \text{(A7)}$$

P_α represents the portion of the defect fraction that cannot be eliminated by doing concept development work, while P_β represents the portion of the defect fraction that can. The term *f(s-1)* is the fraction of the total number of concept development tasks completed in the previous model year:

$$f(s)=\frac{C^c(T)}{C} \quad \text{(A8)}$$

The measure of system performance used in this study is $\pi(s)$, the quality of the product at its introduction date measured as the fraction of its constituent tasks that are defective.

$$\pi(s)=\frac{R(T)+P_D(s)\ V(T)}{D}$$

Appendix B

To reduce the full dynamic system to a first order map, begin with the equation for *f(s),* and substitute in the definition for $C^c(T)$: This yields:

$$f(s) = \frac{\int_0^T \left(Min(K - d(t) - r(t), c^*(t)\right)dt}{C}$$

Reducing the model to a one-dimensional map requires two additional assumptions. First, some form must be specified for *c** and *d**, the desired completion rates. A mathematically convenient and plausible approach is to assume that these are calculated by allocating the remaining work stocks, C^r and D^r, evenly across the time remaining in the model year. Thus:

$c^*=C^r/(T-t)$

$d^*=D^r/(T-t)$

Second, for simplicity, the rework and testing delays are eliminated. This can be accomplished by making the behavioral assumption that participants in the process know the defect rate, $P_D(s)$, with certainty, and plan their work accordingly. Specifically the total work that will need to be accomplished on the current project over of the model year *t* is $D/(1-P_D(s))$, and the desired completion rate, absent a capacity constraint would be constant throughout the model year:

$$d^* = \frac{D}{1 - P_D(s)} \quad \frac{1}{T}$$

Substituting into the equation for *f(s)* yields:

$$f(s) = \left(\frac{1}{C}\right) \int_0^T \left[MIN\left(C, \frac{C}{T}\right), Max\left(K - \frac{D}{1 - P_D(s)} \quad \frac{1}{T}, 0\right)\right]dt$$

With this work allocation rule, each of the three elements is constant throughout the model year and, thus, the integration can be conducted separately and the expressions can be evaluated afterwards. Integrating the three separate elements of the Max and Min functions yields:

$$f(s) = \left(\frac{1}{C}\right) \quad MIN\left(C, Max\left(K \quad T - \frac{D}{1 - P_D(s)}, 0\right)\right)$$

Finally, substituting the definition for $P_D(s)$ again yields the following map for the fraction of concept development work completed each model year, *f(s)*:

$$f(s) = Min\left(1, \frac{1}{C} \quad Max\left(K \quad T - \frac{D}{1 - \left(P_\alpha + P_\beta \quad \left(1 - f(s-1)\right)\right)}, 0\right)\right)$$

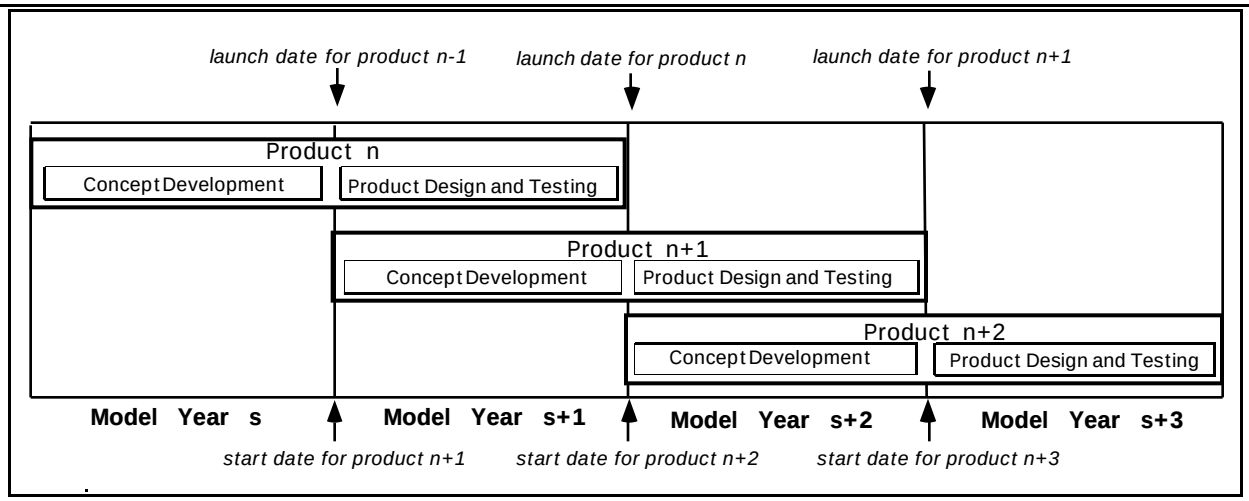


Figure 1: Overview of Development Process. The variable *s* indexes model years and the variable *n* indexes products. As the figure highlights, developing a product requires two years and, at any point in time, two development projects are in progress; one in the concept development phase and one in the design and testing phase.

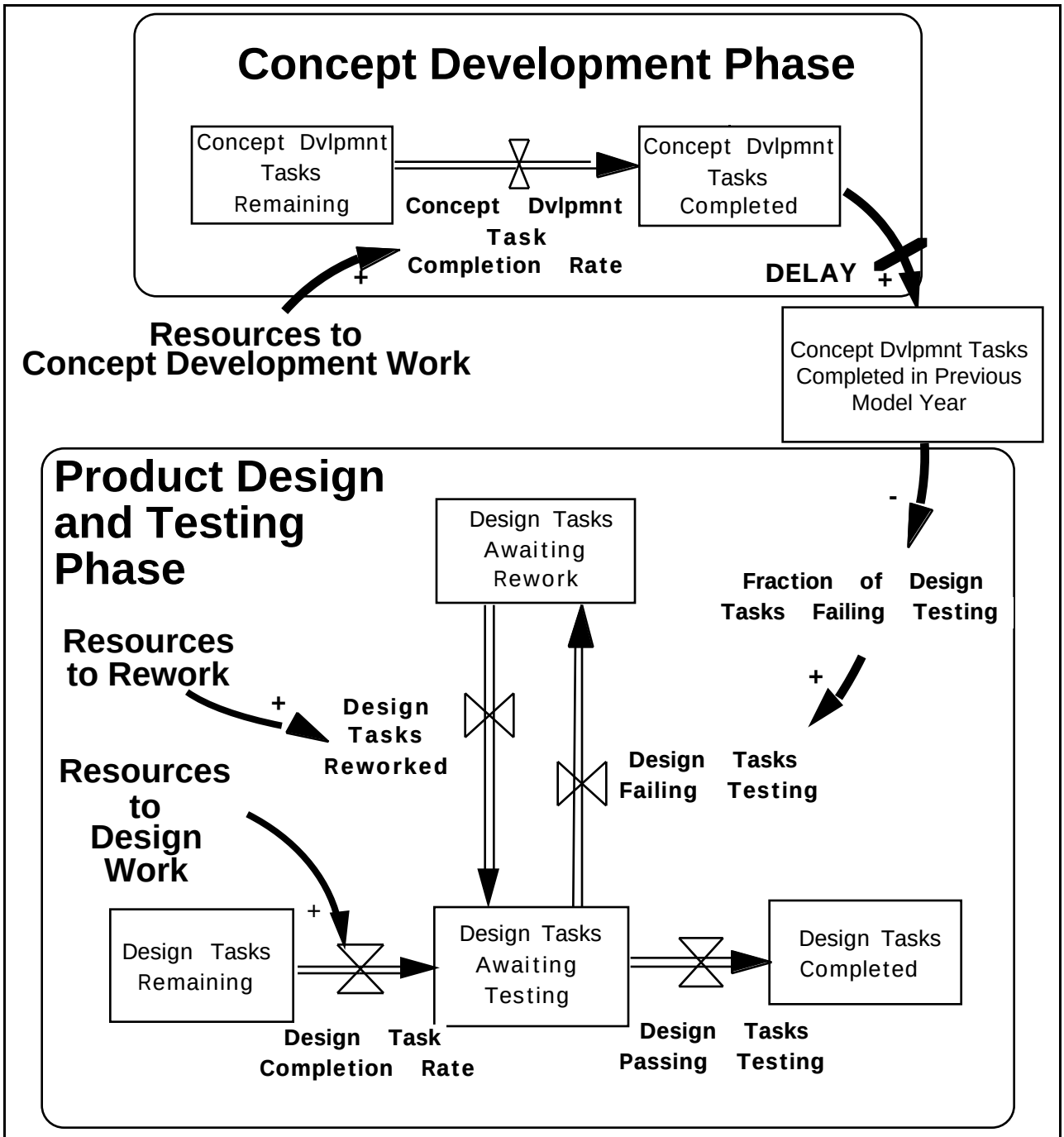


Figure 2: Model Structure. Variables surrounded by rectangles represent stocks or system states. Variables attached to valve symbols represent the activities that move tasks from one state to another. Solid arrows represent the information inputs that are used to determine resource allocation and indicate causal relationships. Signs ('+' or '-') at arrow heads indicate the polarity of relationships: a '+' denotes that an increase in the independent variable causes the dependent variable to increase, *ceteris paribus* (and a decrease causes a decrease); a '-' denotes that an increase in the independent variable causes the dependent variable to decrease, *ceteris paribus* (and a decrease causes a increase). See [33].

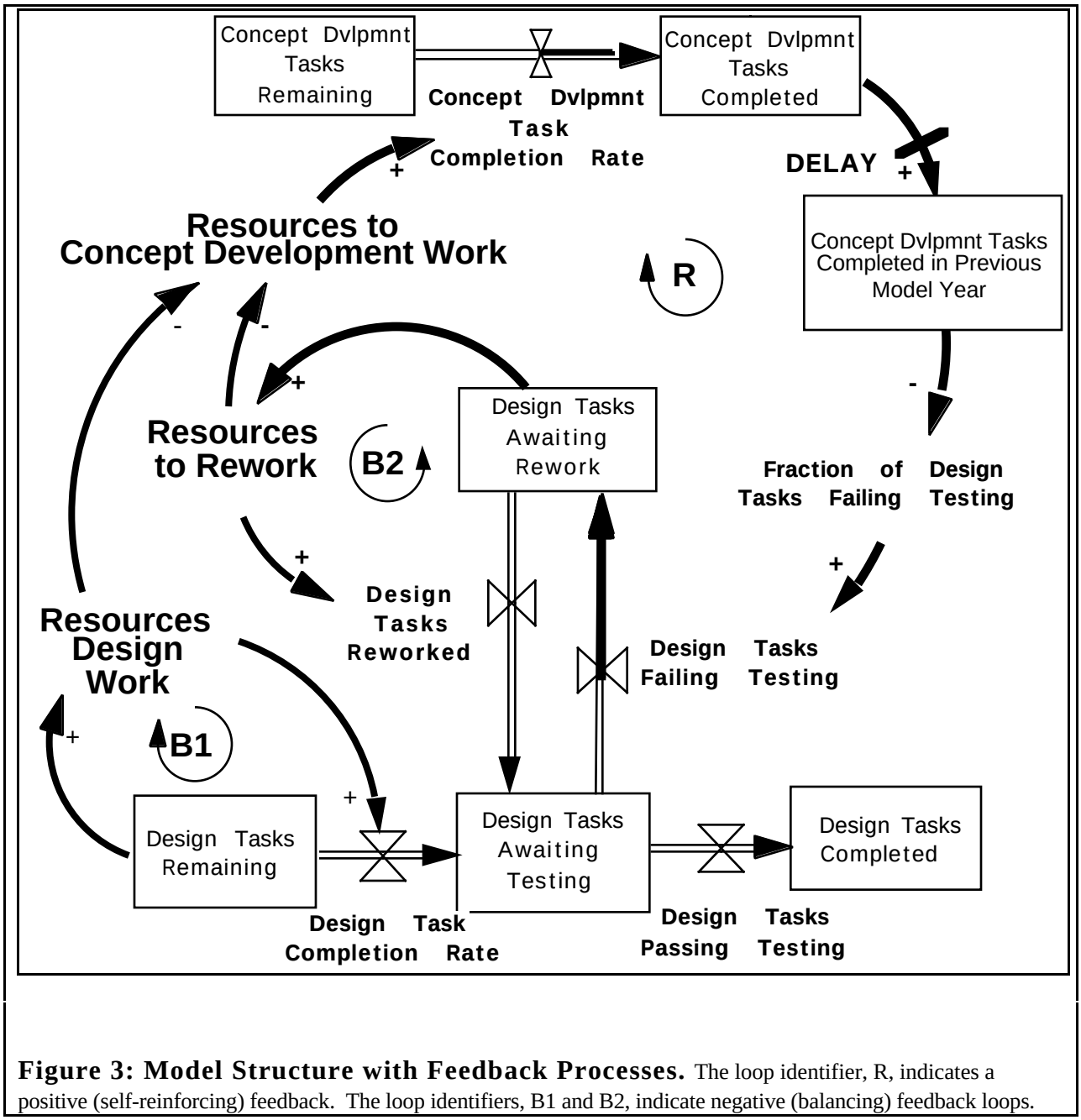
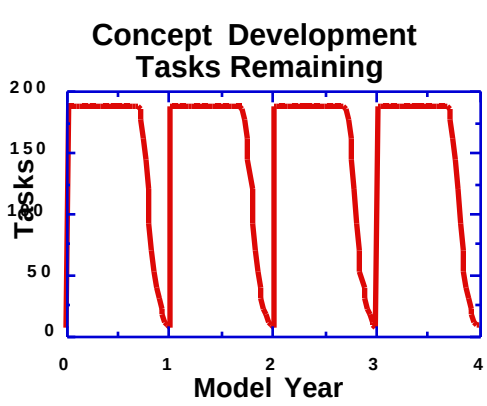
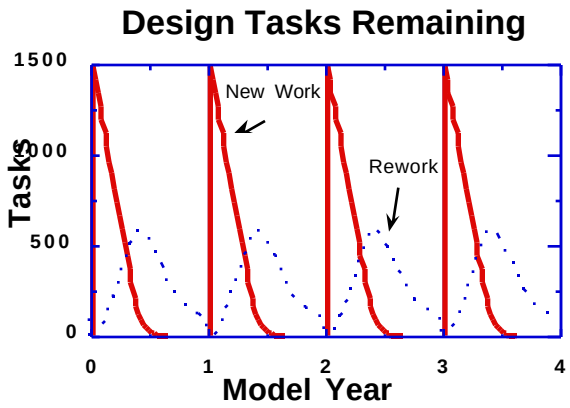


Figure 3: Model Structure with Feedback Processes. The loop identifier, R, indicates a positive (self-reinforcing) feedback. The loop identifiers, B1 and B2, indicate negative (balancing) feedback loops.



a



b

Figure 4: Selected Variables from the Base Case

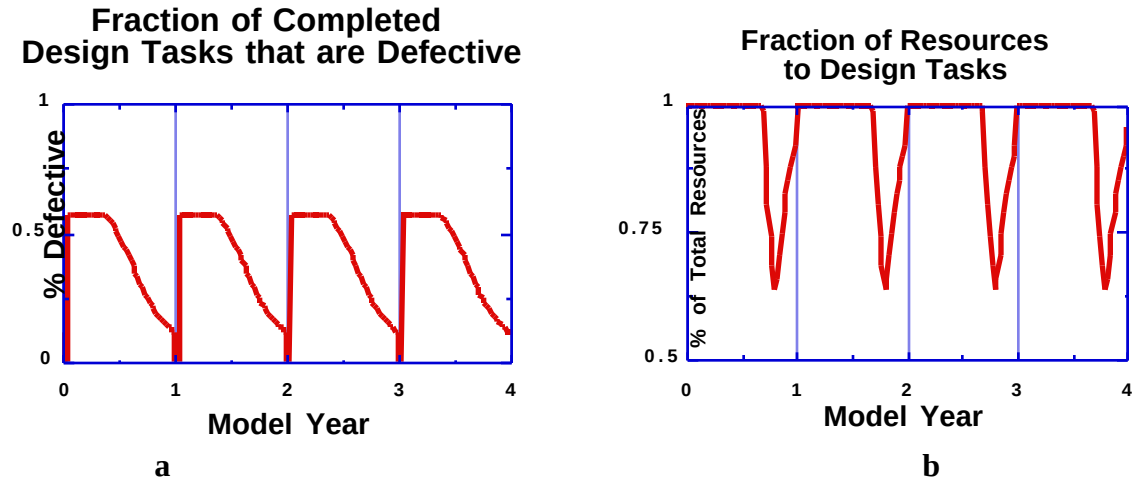
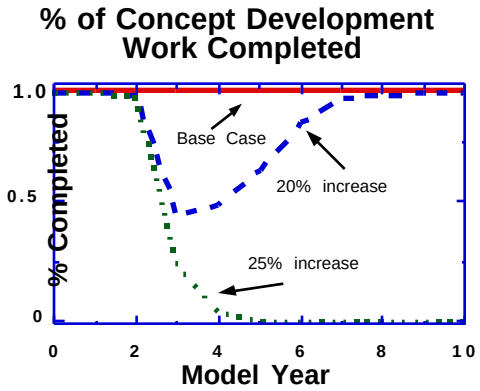
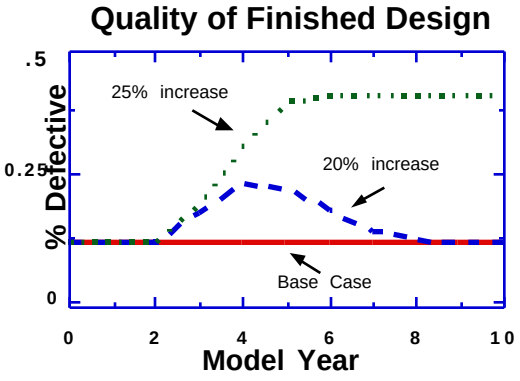


Figure 5: Selected Variables from the Base Case



a



b

Figure 6: Response of System to Temporary Increases in Workload

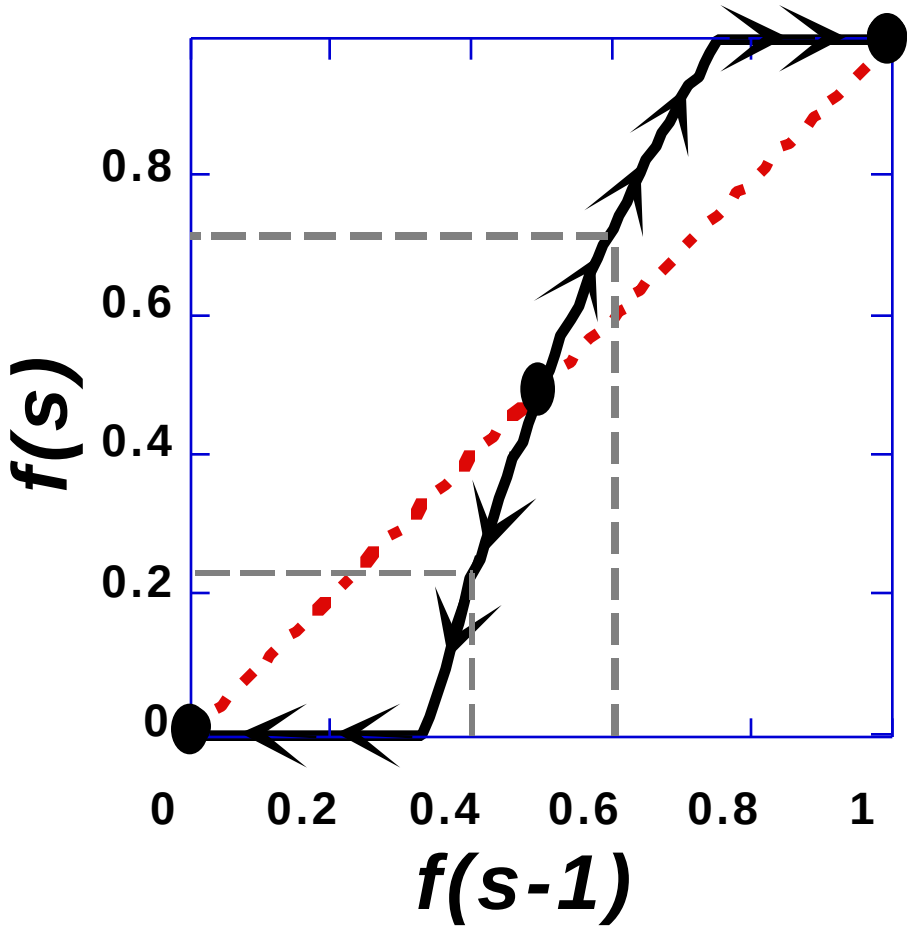


Figure 7: Phase Plot for Simplified System. Solid circles denote equilibria, execution modes that repeat themselves. Arrows on the plot show the "direction" or trajectory of system in disequilibrium situations. Dashed lines indicate how to read the plot: if the organization accomplishes 60% of its planned concept development work this year, then, as the figure indicates, given the dynamics of the system, it will accomplish approximately 70% next year. Similarly, accomplishing 40% of the planned concept development work this year implies that only about 25% will be accomplished next year.

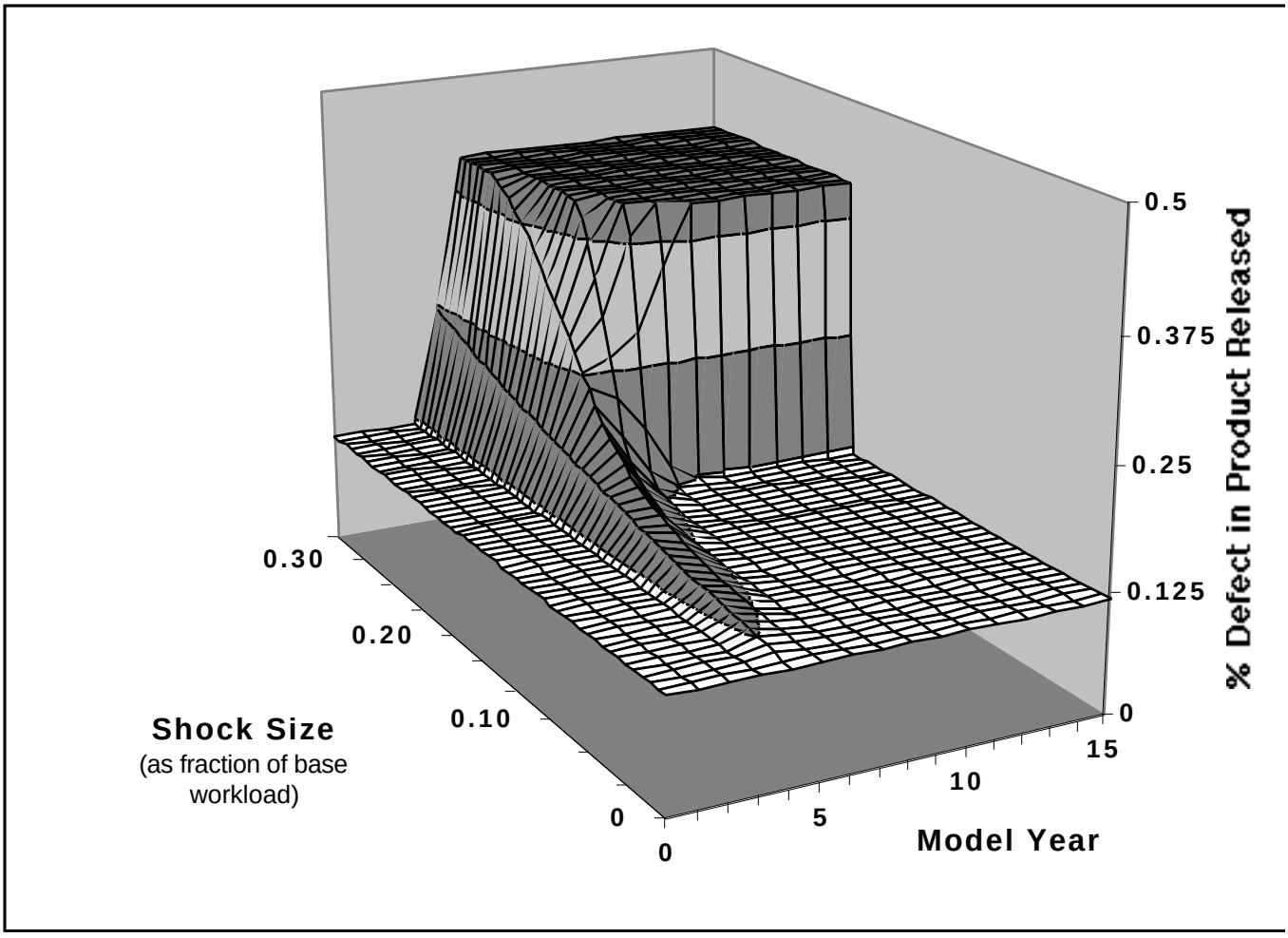


Figure 8. Response of Full System to Temporary Increases in Workload.

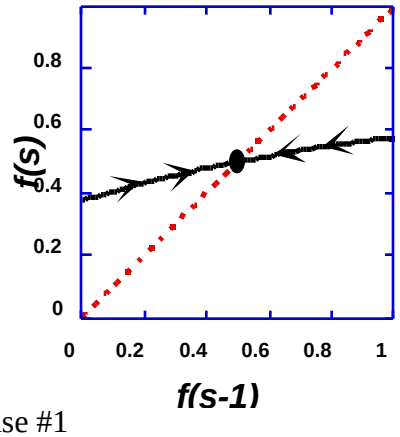
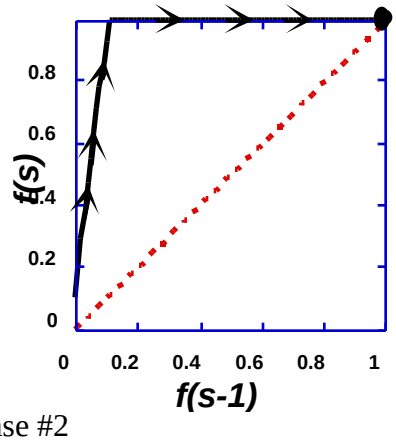
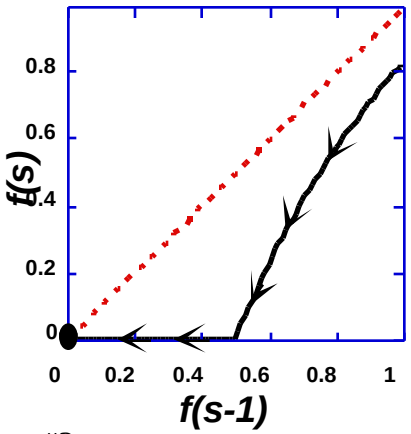
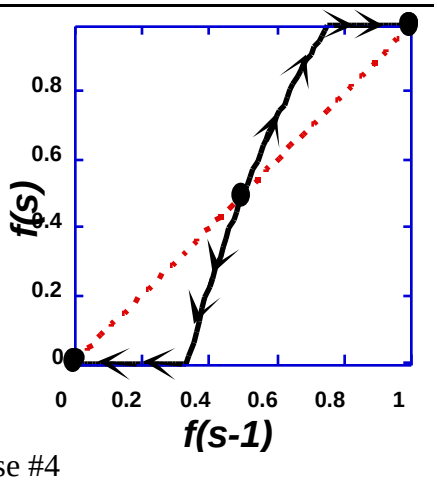
en erating he Fire hting ecution de…		When Operating in the Desired Execution Mode…	
		Capacity is Inadequate $K\ T \le C + \frac{D}{1-P_\alpha}$	**Capacity is Adequate** $K\ T \ge C + \frac{D}{1-P_\alpha}$
	Capacity is Adequate $K\ T \ge \frac{D}{1-(P_\alpha + P_\beta)}$	 Case #1	 Case #2
	Capacity is Inadequate $K\ T \le \frac{D}{1-(P_\alpha + P_\beta)}$	 Case #3	 Case #4

Figure 9: Possible Phase Plots for Simplified System

Tables

Parameter	Definition	Value
C	Number of Concept Development Tasks	180 tasks per year
D	Number of Design Tasks	1500 tasks per year
$P_{\alpha} + P_{\beta}$	Probability of Introducing an Error when Ignoring Concept Development	.75
P_{α}	Probability of Introducing a Defect When Doing Concept Development	.1875
K	Development Capacity	300 tasks per month
T	Length of the Model Year	12 months

Table 1: Base Case Parameters

[i] The model is simulated using the Euler integration method and runs in months with a time step of .25. In each case, it is run for one hundred and eighty months to eliminate transients. The model is written using the VENSIM software produced by Ventanna Systems Inc. A run-only version of the model can be downloaded from <http://web.mit.edu/nelsonr/www/>. Complete documentation is also available at this site.