Why good processes sometimes produce bad results: A formal model of self-reinforcing dynamics in 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: <nelsonr@mit.edu>

July 1999

Version 2.0

[*]*(formerly titled 'The Transition Problem in Product Development')*

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 Andrew Jones, John Sterman, Scott Rockart, Rogelio Oliva, Steven Eppinger, Steve Graves, Rebecca Henderson and Karl Ulrich. Thanks to seminar participants at MIT, Wharton, and INFORMS Seattle. Minsoo Cho developed the web site that accompanies this paper. 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 paper can be found at <http://web.mit.edu/nelsonr/www/>.

Why good processes sometimes produce bad results: A formal model of self-reinforcing dynamics in product development

Abstract

The design of product development processes has received considerable attention from both scholars and practitioners. Unfortunately, however, practice does not always follow theory: many organizations continue to experience difficulty in following the development processes prescribed in the literature. One source of such difficulties is the existence of self-reinforcing processes that drive the organization to allocate an increasing fraction of its scarce resources towards fixing problems in existing projects and away from preventing problems in future ones. While the existence of such dynamics has been widely acknowledged (popular labels include vicious circles, death spirals, and doom loops), they remain poorly understood. In this paper I develop a simple model of a self-reinforcing process in a multi-project product development environment. The analysis of this model leads to a number of new insights. First, it provides boundary conditions for the phenomenon. Second, it shows that such dynamics can work as either virtuous or vicious cycles and provides the conditions under which each of these will occur. Third, the analysis highlights the role that resource utilization plays in determining the robustness of the system to undesirable behavior modes. I also consider one extension to the basic model, the introduction of new tools and processes, and discuss policies for system improvement.

1. Introduction

The design of effective product development processes has received considerable attention from management scientists (recent examples include Ulrich and Eppinger 1995, Cooper 1993, Zangwill 1993, Wheelwright and Clark 1992). Despite the considerable advances in process*design*, however, many organizations continue to struggle with the *execution* of their desired development process. Wheelwright and Clark (1995), for example, lament that, despite the fact that their earlier work (Wheelwright and Clark 1992) has produced significant results in isolated projects, many organizations fail to achieve such success on an ongoing basis. Similarly, Repenning and Sterman (1997), Jones and Repenning (1997), Krahmer and Oliva (1998), Oliva, Rockart, and Sterman (1998) and Ford and Sterman (1999) 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 the Repenning and Sterman (1997) study summarized this dilemma 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."

In this paper I investigate one source of such difficulties. Specifically, I study the possible existence of self-reinforcing processes that drive an organization to allocate an increasing fraction of its scarce resources towards fixing problems in existing projects and away from preventing problems in future ones. This general class of self-reinforcing dynamics has been widely discussed in the management literature (Weick 1979, Senge 1990), and goes under a wide range of names including vicious circles, death spirals and doom loops. Recent applications include Repenning and Sterman (1997) and Carroll, Marcus and Sterman (1997) who use the construct to understand how specific organizations allocate scarce resources between prevention and correction activities, and Perlow (1999) who studies a product engineering organization in which scarce

resources were progressively pulled into 'crisis management', thus leaving little time for the activities that would have prevented such crises.

The self-reinforcing process I study is created by the interaction between the need to allocate scarce resources between competing projects in *different* phases of the developments process and the policy, found in many organizations, of favoring projects in the later stages of that process. Resource scarcity creates interdependence in a multi-project development environment since a decision to allocate more resources to a given project influences those projects that, consequently, receive fewer resources. If such dependence extends to projects in different phases of the development process, then a policy of favoring downstream projects can create a vicious cycle in which the organization allocates an increasing fraction of its resources to fixing problems in projects in the later phases of their development cycles, thus starving projects in the earlier phases. The behavior created by this structure was described by an engineer in the company that provided the basis for this study, when, in discussing the difficulty he experienced in following a prescribed development process, he 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."

Although only one manifestation of a self-reinforcing process, this structure is quite general and its existence has been documented in a variety of product development settings. For example, although they do not conceptualize it explicitly in terms of feedback loops, Wheelwright and Clark (1992) highlight this structure when, in discussing the potential problems associated with overloading a development system, 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." Burchill and Fine (1997),

Perlow (1999), and Repenning and Sterman (1997) also document cases in which such self-reinforcing processes are at work in product engineering systems.

Despite the fact that the existence of such self-reinforcing dynamics has been widely acknowledged in product development, however, the phenomenon remains poorly understood. For example, the existing literature contains little discussion of the conditions required for such a positive feedback loop to both exist *and* dominate the behavior of the system in which it is embedded. Even though it has been almost totally neglected in existing discussions, the second requirement is important because it is quite possible for a feedback loop in a dynamic system to exist but play little or no role in determining the total system's behavior (see Sterman *forthcoming* for a discussion of loop dominance). Thus, to be useful, any theory that posits a positive feedback loop as the core structure generating an observed behavior must not only identify the structures and policies that create that loop, but also suggest the conditions under which such a process creates the behavior mode of interest. Similarly, few attempts have been made to describe the final outcome created by such self-reinforcing structures. Are 'death spirals' transient phenomena? Do they always cause the complete breakdown of a product development process (as is the case in many systems dominated by positive feedback), or is there some intermediate final outcome? Current theory has little to offer in answering these questions.

To fill this gap, in this paper I provide a formal model of self-reinforcing dynamics in a multi-project development setting. The analysis of this model provides a number of new insights. First, my analysis provides the conditions required for a positive loop to dominate the behavior of the system, thus establishing important boundary conditions on the phenomenon. Second, my analysis suggests that self-reinforcing dynamics do not necessarily work in an undesirable direction. While those developing formal feedback models have long recognized that a dominant positive feedback loop can often drive the state of a system in either of two directions, this

possibility is typically ignored in more qualitative treatments. The fact that it can work in either a desirable or an undesirable direction has important consequences for the steady state behavior of such systems and leads to the third major insight of this study: if the positive loop dominates, the system has two stable equilibria. The existence of multiple equilibria suggests that product development processes can get stuck in low performance modes from which, absent outside intervention, they will not recover.

Finally, I discuss one extension to the base model. Specifically, a common response to low performance is investment in a more effective set of development tools. Often such improvements are focused on the early phases of the development process, for example by improving the documentation of customer requirements or introducing 'soft' prototyping methods (Ulrich and Eppinger 1995, Zangwill 1993, Wheelwright and Clark 1992). My analysis suggests that, even if one assumes that such tools, if appropriately adopted, will unequivocally help the organization in the long run, their naive introduction can cause a permanent decline in system performance.

The rest of the paper is organized as follows. In the next section I present the main model and analysis. In section three I extend the model to account for the introduction of new development tools. Section four contains discussion, and section five presents concluding thoughts.

2. The Model

2.1 Model Structure

Overview

The model represents a development system that introduces a new product into the market at a fixed time interval called the *model year.* Two model years are required to develop a product, and at any moment in time two projects are under development. New projects are introduced into the system at fixed intervals coinciding with the product introduction date.

To study the dynamics of resource allocation, I decompose the development process into two phases. A common feature of many product development processes suggested in the literature is an emphasis on *concept development* in the early phases of a project (Wheelwright and Clark 1992, Cooper 1993). For example, Ulrich and Eppinger (1995) suggest a development process that contains five phases–concept development, system-level design, detail design, testing and refinement, and production ramp-up. Since I am interested in the *simplest* possible model that can capture the dynamics discussed above, I assume there are only two phases in the development cycle: the *concept development* phase, which is identical to Ulrich and Eppinger 's first step, and the *product design and testing* phase, which aggregates all the subsequent activities. 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. Figure 1 shows a schematic of this basic structure and timing.

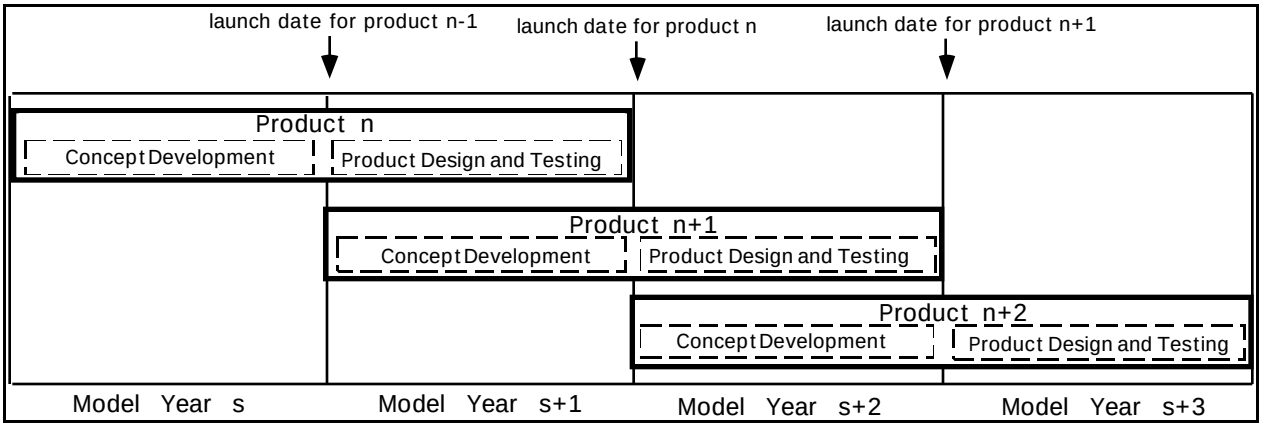


Figure 1

Within each phase there is a set of tasks to be completed. Tasks in the design and testing phase represent those activities required to physically create the product. Tasks in the concept development phase are those activities which make the subsequent design work more effective by reducing the probability that a given design task is done incorrectly. Only one type of resource,

engineering hours, is needed to accomplish both types of tasks. Once a project reaches the product design and testing phase, there is some probability, P_D, that each task is done incorrectly, must be reworked and, thus, requires additional resources. The dependence between the two phases is captured by assuming that P_D is a function of the number of tasks completed when the project was in its concept development phase.

Central Assumptions

The model is built on three central assumptions. The first two are structural and drive all of the results of the paper, while the third is made for computational and expository convenience. The first assumption concerns the connection between the concept development phase and the design and testing phase. The feature of product development that I am trying to capture is the existence of activities in the early phases of a project which can be skipped, but whose primary effect is to increase the effectiveness of down stream tasks. With this in mind, I assume that accomplishing additional tasks in the concept development phase reduces the probability that subsequent design tasks are done incorrectly. The validity of this assumption is supported by those who suggest concept development as an important first step in a product development process. For example, Ulrich and Eppinger's (1995:15) concept development phase includes investigating the feasibility of the product concept, building and testing experimental prototypes, and assessing production feasibility. Consistent with my conceptualization, none of these activities is directly related to designing and testing the actual product but, instead, is focused on increasing the effectiveness of subsequent design work. Similarly, Cooper (1993) suggests that the most common source of major problems encountered in the design phase of a development project is the lack of an appropriate product definition (a key component of the concept development phase). The linkage between concept development and design is also supported by the empirical work of Gupta and Wilemon (1990) who found that a lack of attention to documenting customer requirements

(another key component of the concept development phase) was the most often cited reason for late design changes and attendant rework delays. Burchill and Fine (1997) also report similar findings.

The second key assumption is that when resources are scarce, priority is given to the project in the design and testing phase. The resource allocation rule represents an incentive scheme that was described by one project manager (in the Repenning and Sterman 1997 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. In many cases problems simply must be corrected before a project is launched: the existing design may comprise 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 closer to their introduction date represent a more immediate return on investment, and as projects reach their introduction dates other investments, such as tooling and dedicated production lines, are often predicated on those dates. Further, 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 (Arkes and Blumer 1985; Staw 1976, 1981) 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. One or more objects (e.g. physical prototypes) typically characterize a project in the design phase, while projects in the concept

development phase often have no such physical manifestation. People have been shown to over-weight available and salient features of the environment (Kahneman *et al.* 1982), suggesting that projects in the later phases will receive more attention. Similarly, efforts to design specific elements of a product have a more unambiguous characterization than efforts to determine the larger system in which those elements will fit. People have also been shown to be ambiguity averse (Einhorn and Hogarth 1985), again suggesting a bias towards projects in their later stages.

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. In other industries, however, the introduction of new products can be postponed until the offering reaches a certain quality standard. 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. If the launch date is flexible, product quality must be traded off against time to market, thus requiring the specification of a more complex objective function to evaluate the system's performance. Even with this restrictive assumption, however, the model produces a number of interesting dynamics that are directly applicable to those industries that face an annual product launch date, and the analysis suggests potential failure modes in industries with more latitude to slip their product delivery dates. A formal model with a flexible development cycle remains, however, for future work.

Detailed 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, $A(s\text{-}1)$, the concept development

tasks, and *C(s)*, the design tasks. The model year index *s* will be suppressed unless needed to avoid confusion.

At any time *t* in model year *s*, the system is characterized by seven states. Within the advanced phase tasks have either been completed and reside in the stock $A^c(t)$, or remain to be completed and reside in $A^r(t)$. In the design phase tasks can either remain to be completed, $C^r(t)$, have been completed but require testing ,*V(t)*, have been found to be defective and require rework, *R(t)*, or have been completed and successfully passed testing, $C^c(t)$. The final state, *f(s)*, represents the fraction of concept development tasks that were completed in model year *s*. At the model year transition, all states are reset to their initial values except for *f(s)*. Figure 2 shows the states and how they are interrelated in the form of a stock, flow, and information feedback diagram.

Given these states and the discussion above, 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 resources that remain are allocated to concept development work.

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

$$c(t) = Min(K, c^*(t)) \quad (1)$$

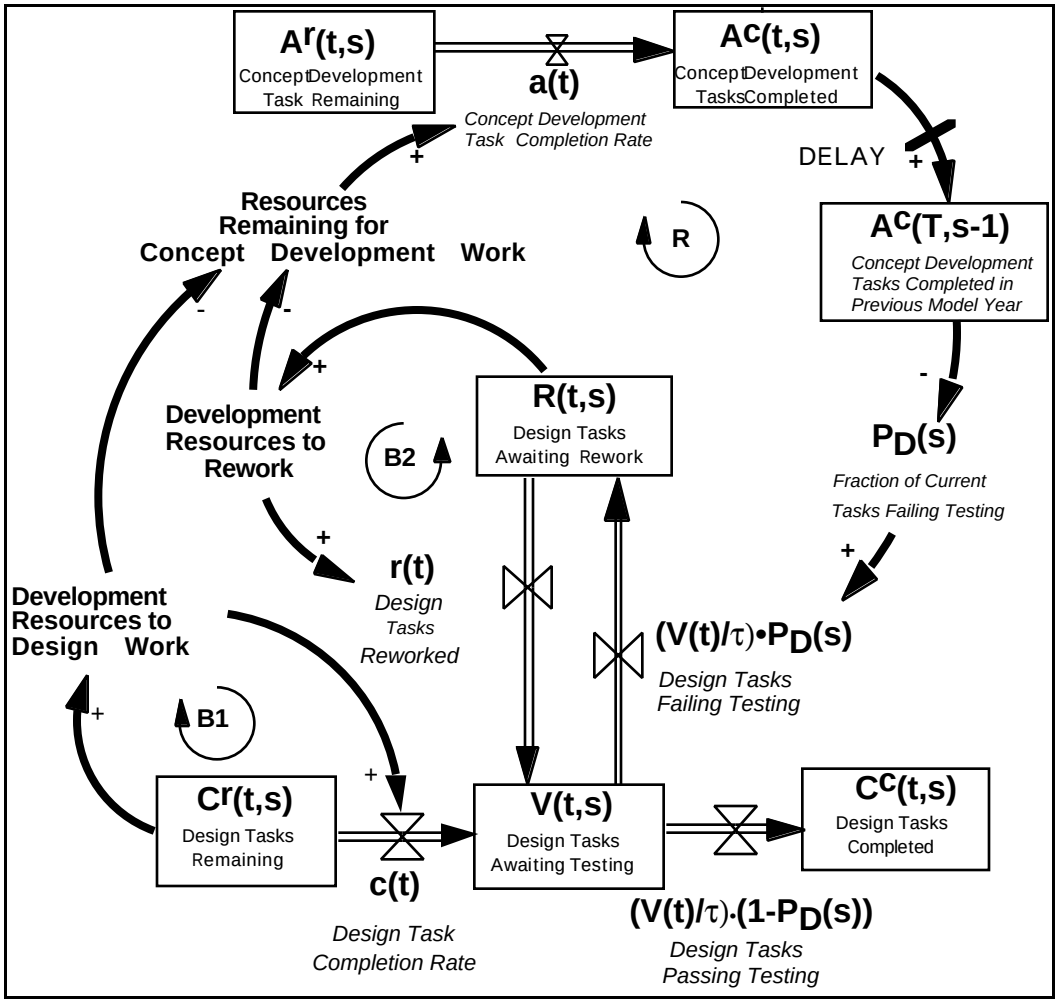


Figure 2

Tasks completed accumulate in the stock of tasks awaiting testing, *V(t).* Testing is represented by a simple 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, $C^c(t)$, 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-c(t),* and the desired rework completion rate, *r*(t)*:

$$r(t)=Min(K-c(t),r^*(t)) \quad (2)$$

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

$$a(t)=Min(K-c(t)-r(t),a^*(t)) \quad (3)$$

Concept development tasks are assumed to be done correctly.

The desired completion rates, *c*(t)*, *r*(t)*, and *a*(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:

$$c^*=C^r/\tau_c \quad (4)$$
$$r^*=R^r/\tau_c \quad (5)$$
$$a^*=A^r/\tau_c \quad (6)$$

where 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:

$$P_D(s)=P_\alpha+P_\beta(1-f(s-1)) \quad (7)$$

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 phase:

$$f(s)=\frac{A^c(s,T)}{A(s)} \quad (8)$$

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.

$$\pi(s)=\frac{R(s,T)+P_D(s)*V(s,T)}{C(s)} \quad (9)$$

With this structure and resource allocation policy, the system contains three important feedback loops (also shown in Figure 2). The first two, B1 and B2, are negative feedback loops that control 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. The third is the positive feedback loop discussed in the introduction. If the resources dedicated to design work increase, then fewer resources are dedicated to concept development work, reducing the concept development task completion rate and, ultimately, the completion fraction. If the completion fraction declines, in the *next* model year the defect rate is higher, more rework is generated, more resources are required for design work, and even fewer concept development tasks are completed.

2.2 Model Analysis

Base Case

Figures 3 and 4 show the behavior of selected variables in the base simulation run of the model.[1] All parametric assumptions are summarized in Table 1.

Parameter	**Value**
$A(s)$	180 tasks per year
$C(s)$	1500 tasks per year
τ_v, τ_c	1 month
$P_\alpha + P_\beta$	.75
$P_\beta / P_\alpha + P_\beta$	.25
K	300 tasks per month
T	12 months

Table 1

In the base case, 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 3b). As design tasks are completed, the stock of rework begins to grow (Figure 3b), and resources are shifted towards completing rework. Multiple iterations through the rework cycle reduce the fraction of defective tasks in the downstream product (Figure 4a). As the stock of outstanding rework declines, resources are moved to the project in the concept development phase (Figure 4b), and the stock of

[1] 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

concept development tasks begins to decline (Figure 3a). As the end of the model year approaches, some rework remains uncompleted so resources are shifted back towards design work (Figure 4b). In the last month of the model year almost all the resources are focused on improving the quality of the product in the design phase.

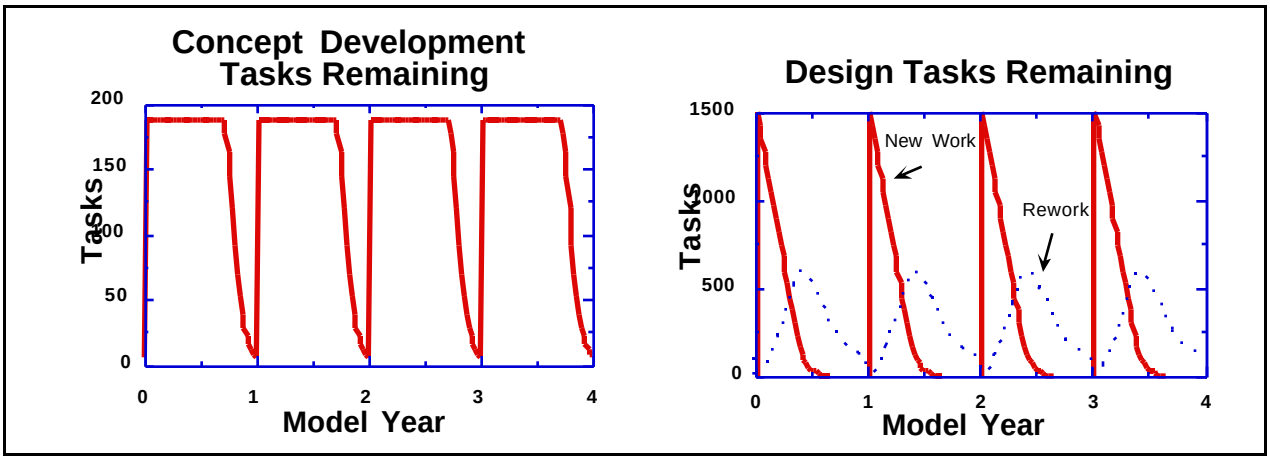


Figure 3

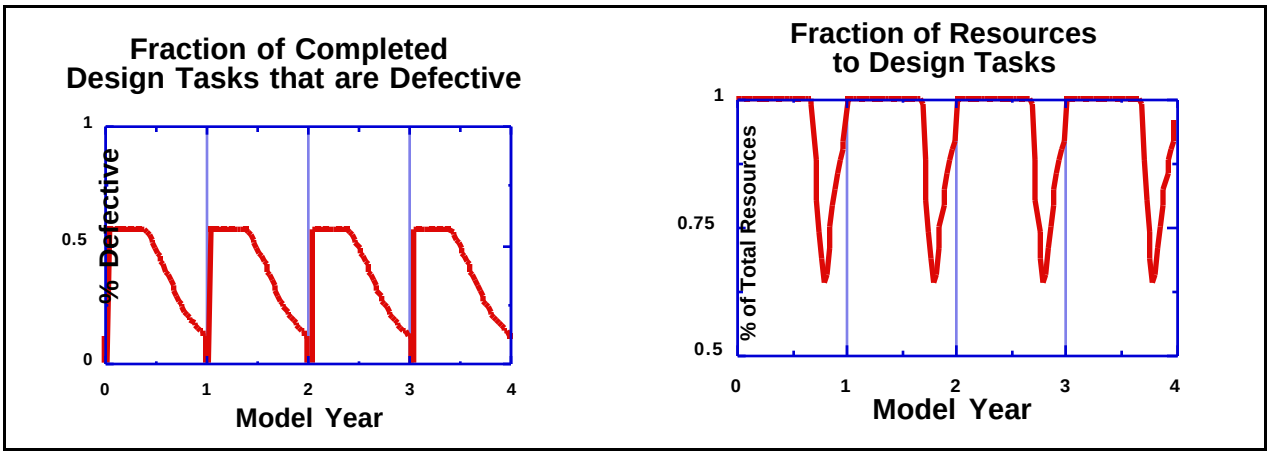


Figure 4

The base case depicts an organization that operates in a desirable mode. It completes the vast majority of the concept development work it undertakes and does only a modest amount of 'fire fighting' (as shown by the increased allocation of resources to the design phase at the end of the

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.

model year). I now turn to the question of what might cause the system to leave such a desirable mode of operation.

Causes and Consequences of Self-Reinforcing 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 carries over *between* model years. Thus, with suitable approximations to the within-year dynamics, the model can be reduced to a one-dimensional map in *f(s)*. Having made this reduction, it is then possible to describe the global behavior of the system in a parsimonious manner. In the second step, extensive simulation runs of the full model are used to confirm the validity of the approximations as well as explicate some of the interesting dynamics.

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

$$f(s) = Min\left(1, \frac{1}{A} \cdot Max\left(\mathrm{K} \cdot \mathrm{T} - \frac{C}{1-\left(P_\alpha + P_\beta \cdot (1 - f(s-1))\right)}, 0\right)\right) \tag{10}$$

The behavior of this reduced system is described by two key conditions. First, an equilibrium will exist at $f^*(s)=1$ when the second term inside the minimum function of (10) is greater than or equal to 1. Thus, to get such an equilibrium:

$$\mathrm{K} \cdot \mathrm{T} \geq A + C/(1 - P_\alpha) \tag{11}$$

The quantity on the right-hand side of the inequality represents the number of tasks required to develop a defect-free product when all concept development tasks are completed and, as a result,

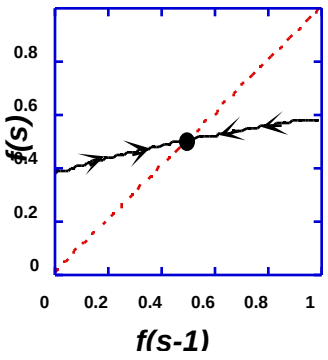
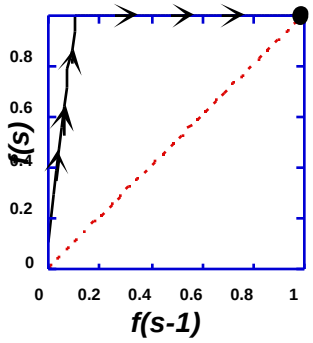
the defect rate is at its minimum. Equation (11) indicates that for the system to have the potential to operate at $f^*(s)=1$, annual capacity, $K \bullet T$, must be greater than or equal to that number of tasks.

Second, an equilibrium will exist at $f^*(s)=0$ when the first term inside the maximum function of (10) is less than or equal to zero. This equilibrium requires:

$$\mathrm{K} \cdot \mathrm{T} \le C \Big/ \left(1 - \left(P_\alpha + P_\beta\right)\right) \tag{12}$$

The right-hand side of this equation 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. Equation (12) indicates that for an equilibrium to exist at $f^*(s)=0$, the capacity level has to be less than this quantity.

Using these two conditions, the space of maps relating $f(s)$ and $f(s-1)$ can be partitioned as shown in Figure 5. Four cases describe the range of potential behavior modes.

	$\mathrm{K} \cdot \mathrm{T} \le A + \frac{C}{1 - P_\alpha}$	$\mathrm{K} \cdot \mathrm{T} \ge A + \frac{C}{1 - P_\alpha}$
$\mathrm{K} \cdot \mathrm{T} \ge \frac{C}{1 - \left(P_\alpha + P_\beta\right)}$		

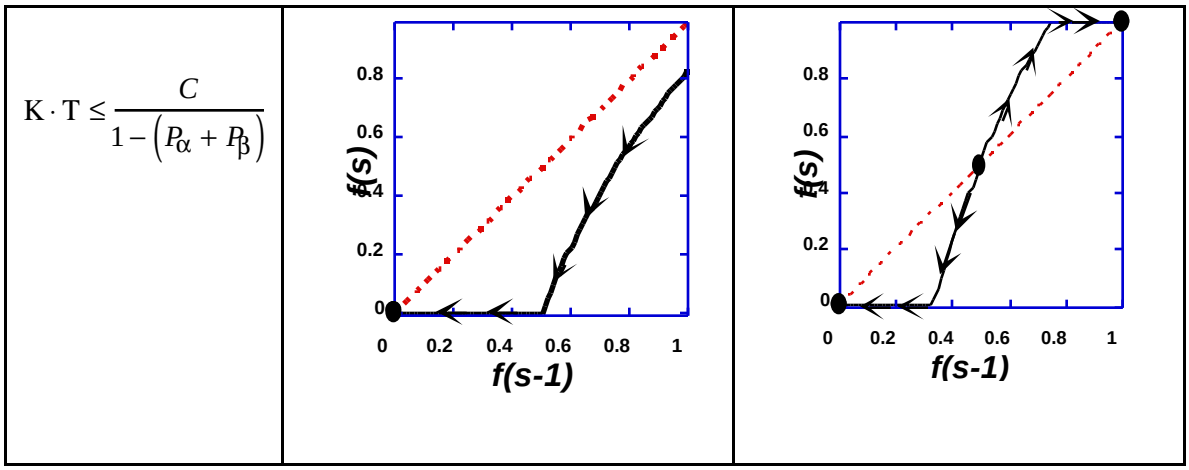


Figure 5

First, as shown in the upper left quadrant, if neither (11) nor (12) is satisfied then the system has a unique, stable interior equilibrium. In this case, combining equations (11) and (12) yields the following inequality:

$$C/\left(1-\left(P_\alpha + P_\beta\right)\right) \le A + C/\left(1-P_\alpha\right)$$

For this case to obtain, 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 product when concept development activities are ignored. The inequality is only satisfied when concept development tasks require more resources to complete than they save via the reduced defect fraction. Here the organization would be better off ignoring concept development altogether and doing more design tasks.

Second, as shown in the upper right quadrant, if (11) is satisfied and (12) is not, then the system has one stable equilibrium at $f^*(s)=1$. The conditions for this case indicate that the existence of a unique equilibrium in which all concept development tasks are completed requires that total capacity, KT, must be *greater* than the workload regardless of whether or not concept development tasks are completed. Third, as shown in the lower left corner, if (12) is satisfied and (11) is not, then the system has one equilibrium at $f^*(s)=0$. In contrast to the previous case, for

this outcome to obtain, resources must be insufficient to support development regardless of the mode of execution.

Finally, as shown in the lower right quadrant, if both (11) and (12) are satisfied then the system has three equilibria, two are stable and one is unstable. Here, resources are sufficient when concept development work is completed, but insufficient when it is ignored. Combining the two conditions yields the following:

$$A + \frac{C}{1 - P_{\alpha}} \leq \frac{C}{1 - \left(P_{\alpha} + P_{\beta} \right)} \tag{13}$$

In contrast to the first case, for (13) to be satisfied, concept development tasks must be 'worth it' and save more resources through the reduced defect fraction than are required to complete them.

The analysis of this simple model and the four resulting cases leads to a number of insights not present in existing treatments of this topic. First, case number one (the unique, stable equilibrium) highlights the fact that the positive loop does not necessarily dominate the behavior of the system, and, if it does not, then the system is not prone to self-reinforcing dynamics. The insight here is that the mere existence of up-front activities, which may be required for reasons other than improving the productivity of downstream work, does not create self-reinforcing behavior. The positive loop only dominates the behavior of the system if concept development tasks have a sufficiently large impact on the productivity of downstream design activities. Thus, the analysis provides an important boundary condition on the phenomenon.

Second, in contrast to qualitative discussion of this issue (e.g. Senge 1990, Wheelwright and Clark 1992, Perlow 1999) which are often based on the implicit assumption that such processes only work in one direction, the second case demonstrates that, even if the positive loop dominates, it does not necessarily work in the undesirable direction. If resources are sufficient, regardless of the state of the development process, then the positive loop works only in a virtuous direction, creating

a reinforcing cycle of decreasing error rates in the design phase and sustained investment in concept development activities. The paradoxical feature of this result is that creating a system that guarantees that all of the concept development tasks are completed requires enough resources to support development even when those tasks are ignored. In other words, insuring that the desirable operating mode prevails requires a level of resources that completely negates the efficiency gains that the desirable operating mode might otherwise provide.

Third, the analysis both confirms and extends the existing intuition concerning the role of resources. Wheelwright and Clark (1992) suggest that overloading a development process creates a system in which self-reinforcing dynamics *may* occur (if, for example, a project runs into "…unexpected trouble."). While this is true in the fourth case to be discussed below, case number three suggests that if the resources are sufficiently scarce, then undesirable self-reinforcing dynamics *will* occur. If resources are insufficient to support development in any mode of operation (as in case three), the system has only one possible trajectory: a downward spiral of increasing error rates in design work and decreasing investment in upstream activities.

Fourth, in contrast to the existing discussions of self-reinforcing processes (in which they are often treated as purely transient phenomenon), the analysis shows that the dominant positive feedback loop can create a system with two stable equilibria. Equation (13) implies that this case obtains when resources are *insufficient* to support development if concept development tasks are ignored but *sufficient* if all the concept development tasks are completed (this case also requires the assumption that concept development tasks are 'worth it'). This is perhaps the most interesting case since it holds when managers try to reap the benefit of concept development activities by either increasing the workload or reducing the allocation of resources. Under these conditions, once the system reaches one of the two stable equilibria small excursions away from that equilibrium are counteracted by the positive loop, which acts as a powerful force driving the

system towards one of two steady state behaviors. The existence of a stable equilibrium at *f(s)=0* suggests that a product development system can get 'stuck' in an undesirable mode of operation from which, absent an additional intervention, it will never recover. In this system, getting caught in a 'vicious' cycle does not mean that the system collapses, but instead that it is trapped in a constant state of low performance. The paradoxical feature of this condition is that, by reducing the allocation of resources to take advantage of the efficiency gains inherent in concept development tasks, managers *create* a system in which the use of those tasks is not guaranteed.

Variability and Performance

Finally, the model also suggests what types of external shocks are required to push the system from one equilibrium to the other. Specifically, the unstable equilibrium represents the boundary between two basins of attraction: if the system starts at any point *below* the unstable equilibrium, then it will move towards *f*(s)=0*; if the system starts from any point *above* the unstable equilibrium, then it will move towards *f*(s)=1*. If a shock is large enough to move the system beyond the boundary, then the system does not return to its initial equilibrium, but instead follows a new trajectory to the other equilibrium. The dynamic behavior created by the boundary point is highlighted by the following simulation experiments. 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.

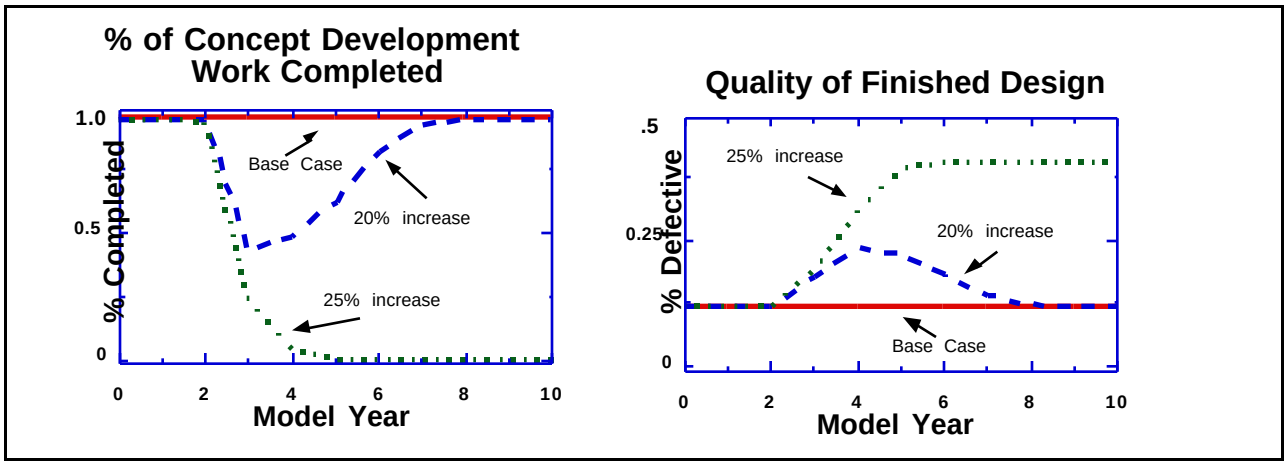


Figure 6

In the first model year after the increase the additional work reduces the fraction of concept development work completed (Figure 6a). This leads to an increase in the defect rate for design work (Figure 7a) which, in turn, creates additional rework (Figure 7b). The 20 percent change prevents the concept development task completion fraction from returning to its pre-pulse level, but it does *increase* over the previous year (Figure 6a).

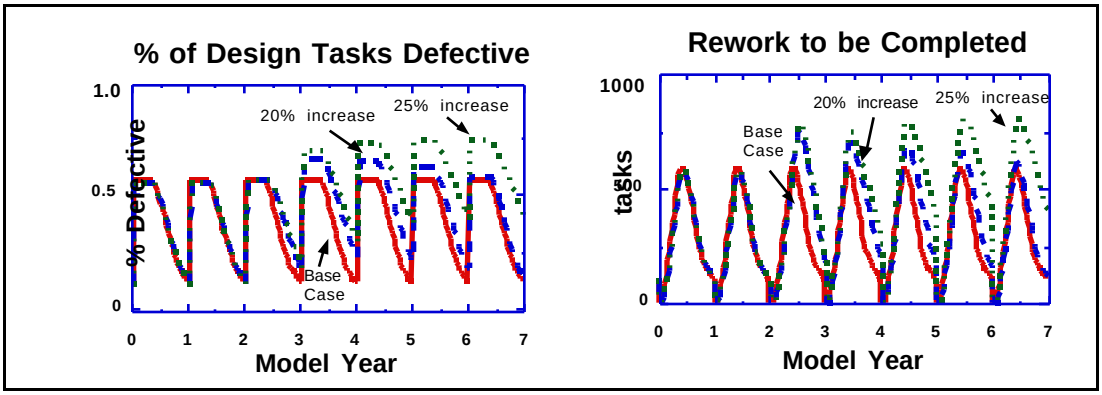


Figure 7

The increased completion fraction causes the defect rate to decline, the resources required for rework in subsequent years to decline, and the completion fraction to increase further. In this case the positive loop works in a virtuous direction and drives the system back to the initial equilibrium and performance level (Figure 6b).

In contrast, the 25 percent increase shows different behavior. The initial decline in the concept development completion fraction is greater (Figure 6a), and the larger drop leads to a higher defect rate for subsequent design work (Figure 7a) which, in turn, creates additional rework (Figure 7b). In contrast to the previous case, the growth in resources required for rework is large enough to cause the completion fraction to decline further in subsequent model years, thus initiating the downward spiral (Figure 6a).

The difference in the final outcomes of the two experiments is a consequence of the fact that the first shock is not sufficient to move the system beyond the boundary while the second shock is. Figure 8, which shows the system's performance for a range of test inputs, highlights the role that the boundary point plays in determining the system's reaction to external shocks.

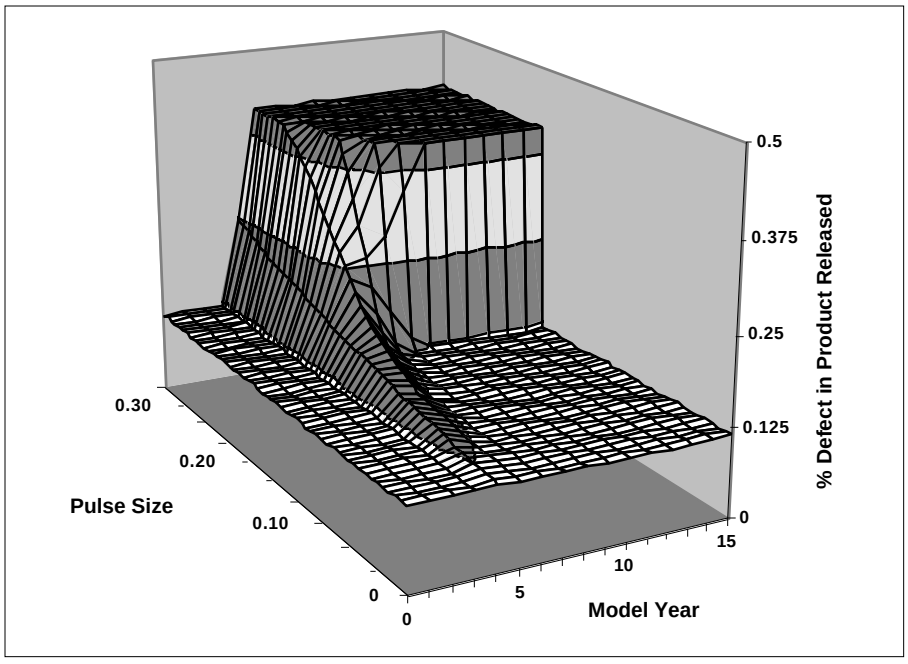


Figure 8

The boundary point, and the role it plays in determining the dynamics of the system, lead to three additional insights. First, as the simulation tests highlight, in this system a *temporary* change large enough to move the system from one basin to another causes a *permanent* decline in performance. Second, the size of the shock required to shift the system from one basin to the other is determined

by the location of the boundary point: if the boundary point is quite close to *f(s)=0*, then a substantial increase in workload is required to shift the trajectory towards the *f(s)=0*, and, conversely, if the boundary point is quite close to *f(s)=1*, then only a small increase is need to push the system into the new regime. Third, and most importantly, inspection of the four phase plots reveals that the location of the boundary point is determined by the utilization of resources in steady state. As the steady-state level of utilization increases, the unstable equilibrium moves closer to *f(s)=1* until resources become totally insufficient and both the boundary point and the stable equilibrium at *f(s)=1* disappear (case number two then obtains). Similarly, as steady-state utilization decreases, the unstable equilibrium moves towards *f(s)=0* until resources become totally ample and only the equilibrium at *f(s*)=1* remains (case #3). Thus, there is a trade-off between the system's robustness to the downward spiral and its steady state performance.

2.3 Implications for Practice

All of the undesirable dynamics identified so far ultimately stem from the combination of too many projects and too few resources. That too few resources hurts performance does not constitute a particularly novel insight. However, while the connection between overloading and poor development performance has been well recognized, the analysis demonstrates that the linkage between the two is more subtle than the existing literature suggests, and that these subtleties are more than theoretical novelties, they also have at least three important implications for practice.

First, the existing literature suggests that overloading causes performance to decline due to the costs associated with switching between projects and spreading resources over those projects (Wheelwright and Clark 1992:88-90). My analysis highlights an additional, and potentially significant, cost of overloading; it changes the *mode* of process execution by shifting the balance of attention towards downstream activities. Managers allocating development resources should

realize that there is not a proportional trade-off between the quantity and quality of projects. Instead, the relationship is non-linear, and, as utilization increases, the marginal cost of introducing an additional project grows dramatically due to the changes induced in process execution.

Second, the analysis shows that an organization can have sufficient resources to execute the process correctly for all projects and *still* be stuck in an undesirable equilibrium. The important implication of this insight for practice is that, when doing resource planning, it is critical to assess the resource requirements for projects given the *current* mode of process execution, not that which would be required were the process operating as desired. Assuming that the process is operating in the high-performance mode, when it is in fact not, leads to resource allocation decisions that can trap the system in the undesirable operating mode. Moving the system from $f^*(s)=0$ to the desired mode of execution requires that the number of projects in progress be reduced until the balance between concept development and design work is improved.

Finally, the analysis also suggests that managers must pay close attention to increases in project resource requirements caused by unforeseen problems and changes in the desired product. At the outset of a given project the organization may develop a plan that well matches resources to the project's known requirements. Projects, however, often require more resources than anticipated due to changes in scope or major design problems. Such contingencies constitute a transient increase in workload, and, if resource plans are not rapidly updated to account for them, they may be sufficient to initiate the downward spiral and cause a permanent decline in system performance. Managers who are either unable or unwilling to modify resource plans in response to such unexpected changes face an important trade-off between the steady state performance of the systems they oversee and the robustness of those systems to pathological self-reinforcing dynamics. As resource utilization increases, the size of the unplanned increase in workload necessary to initiate the downward spiral declines.

3. Extensions: Introducing New Tools in the Concept Development Phase

The introduction of new tools and processes is often considered as an antidote to low performance in product development. In this section, I extend the model to study the consequences of introducing such innovations in the concept development phase. The main result of this extension is that, while the introduction of concept development tasks has the potential to improve overall performance, if one assumes that time is required before the organization becomes fully capable with these tasks, then the naïve introduction of such tasks can potentially initiate the downward spiral discussed above.

3.1 Model Structure

The model developed so far is based on the assumption that the defect fraction, P_D, is divided into two fixed components, P_α, the fraction of defects that cannot be eliminated by concept development tasks, and, P_β, the fraction that can. The introduction of additional development tools is modeled by making these two parameters variable and assuming that they are determined by the number of concept development tools currently in use. The introduction of a new set of tools means that A, the set of up-front activities, is increased. So in the year of introduction, A is increased from A^1 to A^2 ($A^2 \supset A^1$). The cost of introducing new tools is captured by the fact that A is now larger and thus requires more resources.

The benefit of the new tools is captured by changes in P_α and P_β. The sum of P_α and P_β is assumed to be constant, $P_\alpha + P_\beta = P^{max}$, and the split between the two probabilities is determined by the variable $w(s)$:

$$P_\alpha(s) = P^{max} \cdot (1 - w(s)) \quad (14)$$

$$P_\beta(s) = P^{max} \cdot w(s) \quad (15)$$

$w(s)$ captures the organization's current ability to use the tools it has at its disposal and is determined by the following equations:

$$w(s) = \gamma \cdot w(s-1) + (1-\gamma) z(s), \ \gamma \in [0,1] \tag{16}$$

$$z(s) = \lambda \cdot z(s-1) + (1-\lambda) \bullet \left\{ \frac{A^c(s,T)}{A^{\max}} \right\}, \ \lambda \in [0,1] \tag{17}$$

w(s), the organization's ability, is a moving average of *z(s)*, the organization's current knowledge of the tools. Equation (16) captures the fact that, once a tool is learned, additional time is required to develop experience before it can be used effectively. $1/\gamma$ represents the average time required to gain such experience and is assumed to be one and one half model years.

z(s) is determined by $A^c(s,T)/A^{max}$ where A^{max} is the maximum available set of tools and $A^c(s,T)/A^{max}$ represents that fraction of that set used in year *s*. To capture the delays inherent in learning how to use new tools, *z(s)* is assumed to be a weighted average of that fraction where $1/\lambda$ represents the average learning delay (assumed to be one and one half model years). Thus, if the feasible set of tools is increased, the organization does not immediately benefit from using them. Instead, time is required to learn the new tools and develop the appropriate skill. This structure also adds a 'use it or lose it' dynamic to the model: if the organization does not perform concept development tasks, then its ability to execute those tasks declines.

3.2 Analysis

Figures 9 and 10 show two simulations in which new tools are introduced. In the base case the firm is assumed to use only 25 percent of the available tools, but at 100 percent effectiveness ($A^1/A^{max} = w(s) = .25$). In the first simulation, in model year one, the set of tools is doubled so that $A^2/A^{max} = .5$. In the second, the set of tools is increased by a factor of four so that $A^2/A^{max} = 1$.

In both cases, due to the increase in the number of concept development tasks, the fraction of those tasks completed declines (Figure 9a). In the case of a 50 percent introduction, the initial decrease is insufficient to initiate the downward spiral. Further, since the fraction of concept development work stays high, experience with the tools, represented by *w(s)*, begins to increase (Figure 9b), and the fraction of defects that can be eliminated by concept development work also increases.

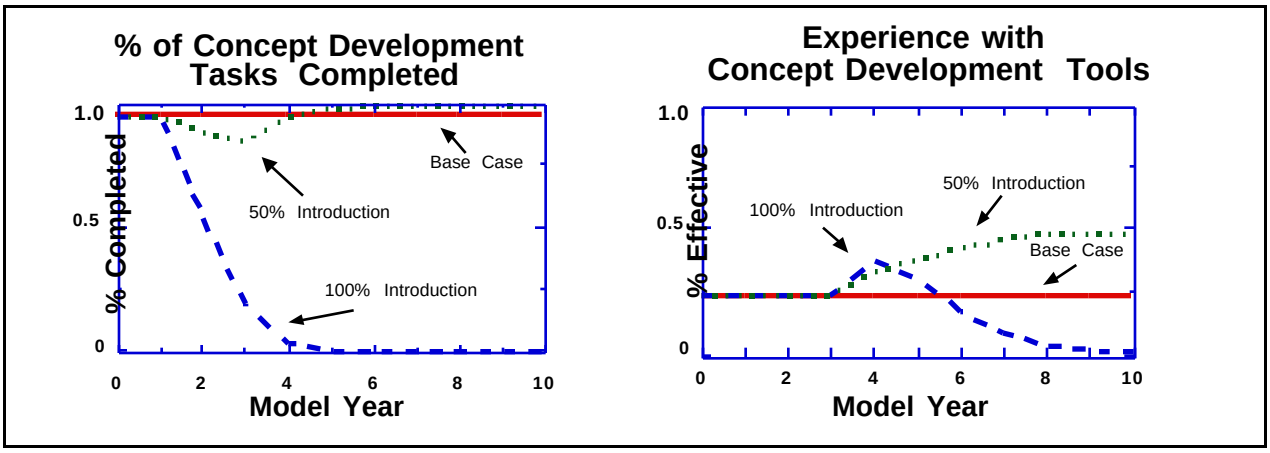


Figure 9

The increase in P_β makes the dominant positive loop stronger, further speeding the virtuous cycle that drives the system towards *f*(s)=1*. The steady state performance of the system is significantly improved (Figure 10).

In contrast, although the second simulation is identical to the first in every respect except that more tools are introduced, the results are quite different. The more ambitious change in the process causes a larger initial decline in the concept development task completion fraction (Figure 9a), a greater defect fraction in design tasks, and a further reduction in the completion fraction. Initially, experience with the new tools begins to rise making design work more productive (Figure 9b), but the growth is quickly offset by the decline in the concept development task completion rate. As the completion rate declines, experience with and knowledge about the tools also decline. The downward spiral continues until the model reaches an undesirable equilibrium in which little concept development work is accomplished (Figure 9a) and system performance is reduced (Figure 10). The cause of the decline is the introduction of a new tool set sufficiently large that the initial cost of learning moves the system past the boundary into the other basin of attraction. Once beyond that boundary, the positive loop progressively reduces the amount of concept development work completed, thereby increasing the defect fraction and reducing the levels of knowledge and experience.

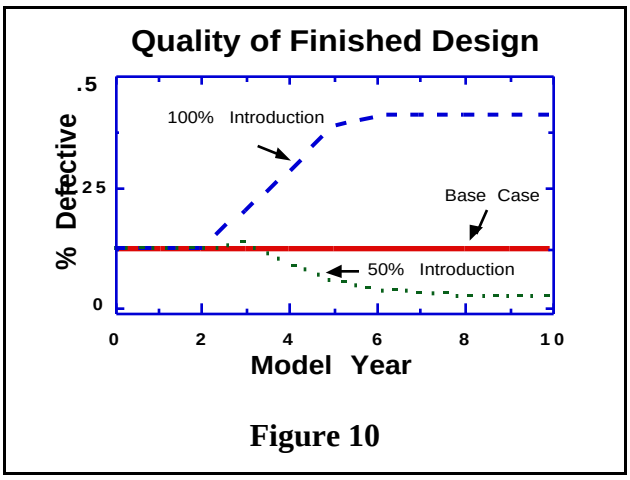


Figure 10

Figure 11 shows the response surface for a range of introduction strategies. As the size of the new set of tools increases, steady state performance also improves until the boundary point is reached. Beyond an introduction level of 40 percent, the loop operates in the opposite direction and pushes the system towards the undesirable equilibrium. Thus, although the tools would unequivocally help the organization *if* the appropriate knowledge and experience were developed, their introduction can make the system's performance worse rather than better.

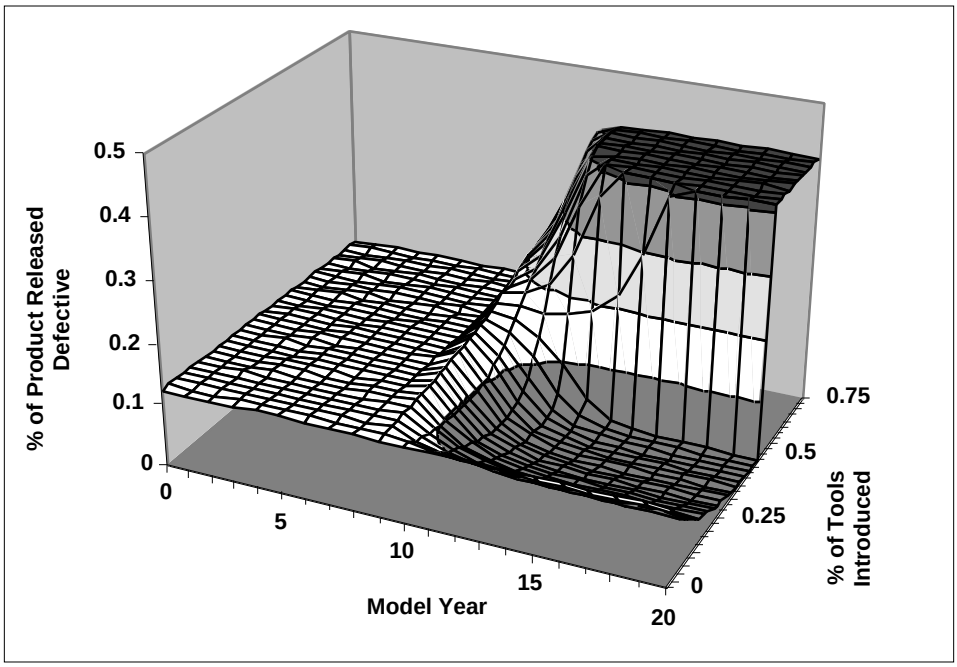


Figure 11

3.3 Implications for Practice

These experiments produce an important insight for practice: introducing new tools without either temporarily increasing the resource level or reducing the workload can cause both the failure of the effort *and* move the system into the undesirable equilibrium. While this may seem somewhat

obvious *ex post*, the phenomenon is subtler than it first appears for two reasons. First, it constitutes an important challenge to currently popular change strategies. For example, a failure mode observed in a large scale product development improvement effort studied in Repenning and Sterman (1997) was the allocation of resources under the assumption that the new process was already in place and operating at full capability. This rule was not inadvertent, but an explicit part of the improvement strategy under the rationale that if the resources were removed participants would have no choice but to follow the new, and more efficient, process to get their work done. The flaw in this logic is that there were already projects in progress that had not been done using the new process and thus required more downstream resources than the new process dictated. The ensuing scarcity initiated the downward spiral and caused a significant decline in capability and the failure of the initiative.

Second, managers that fail to account for the 'worse-before-better' behavior of new tools and processes run the risk of both failing to reap the benefits of those tools *and* of pushing the system in an undesirable trajectory and permanently degrading performance. Neither the practitioner nor the scholarly literature has identified or discussed this second cost of introducing new tools and processes. Managers now face a literal alphabet soup of improvement methods and techniques, and often show little patience in implementing them. Many organizations are in an almost constant state of change due to attempts to introduce the newest and latest tools and processes. While any one of these innovations, if given the necessary time and resources, might improve performance, the continued onslaught of new tools and methods may both significantly hurt performance and prevent any single effort from being successful.

4. Discussion

4.1 Why Does the Phenomenon Persist?

The behavior studied in this paper 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 rule was justified based on empirical observations and various cognitive biases that are well documented in the decision making literature. Each of the justifications is, however, essentially static. Thus, while it might be the case that managers would initially allocate resources in such a fashion, with time would not managers eventually learn to overcome these dynamics? In other words, why does this phenomenon persist? 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?

The answer to the first question is an emphatic no. Consider the outcome feedback a manager receives from the decision to allocate additional resources to a project that experiences problems late in its development cycle. 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 its impact is relatively easy to characterize. Second, the decision to allocate resources to a downstream project, if it initiates the downward spiral, degrades the performance of the product development *system.* In contrast to the improvement in the performance of the downstream project, this outcome occurs with a significant time 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 such systems are faced with a 'better-before-worse' trade-off in which the positive, but transient, consequence of the decision is easy to assess and happens quickly, while the negative, but permanent, consequence is difficult to assess and happens with a delay.

In such situations, the literature on human decision making has reached a clear conclusion: people do not learn to manage such systems well (Sterman 1994). In experiments ranging from managing a simulated production and distribution system (Sterman 1989) to fighting a simulated forest fire (Brehmer 1992) to managing a simulated fishery (Moxnes 1999), subjects have 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, however, is even 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 to suggest 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 the structure of the process itself (an example of the widely documented 'Fundamental Attribution Error', see Plous 1993). Thus, as performance begins to decline due to the downward spiral, 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 the downward spiral is initiated system performance will continue 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 workload in 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.

4.2 Policies for Improving Performance

What, then, can an organization that finds itself in this undesirable state of affairs do about it? The most obvious solution is to improve the aggregate resource planning process. As discussed above, there are at least four potential failure modes in the resource planning process. First, resources may simply be insufficient, thus dooming the system to low performance. Second, resources may be sufficient if the process was operating as designed, but insufficient given its actual operating mode, again creating the downward spiral. Third, resources may be sufficient in steady state, but the plan is not updated quickly enough to reflect changes in the product plan or other unforeseen contingencies. Fourth, resource plans may fail to account for the likely 'worse-before-better' behavior mode created by the introduction of new up-front tools and processes.

The potential downfall of any policy directed at eliminating these failure modes is that managers are the ones making the plans. The ultimate source of all of these dynamics is the faulty mental models of those who set resource levels and those who must allocate those resources among different 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 virtually guaranteed to find herself in a downward spiral. Thus, the high leverage point for improving system performance lies in improving the collective understanding of the system's dynamics.

The focus on improving mental models has proven successful in overcoming similar dynamics in domains outside of product development. For example, Carroll *et al.* (1997) discuss efforts to improve machine 'up-time' performance at the du Pont Corporation. Preventive and reactive maintenance are coupled in a dynamic structure similar to the one discussed in this paper: focusing on reactive maintenance reduces the resources available for preventive efforts, further increasing the need for reactive maintenance. du Pont found that, despite investing more in maintenance than many of its competitors, as a consequence of being trapped in a self-reinforcing spiral, it had below average machine up-time rates. Overcoming this spiral required a dramatic shift in the mental models of both managers and maintenance mechanics. To change these mental models, du Pont used a hands-on simulation board game to help participants develop a deeper understanding of the self-reinforcing dynamics inherent in the maintenance function. The results have been quite dramatic. Plants playing the game have seen dramatic improvements in up-time and significant cost savings. Such simulation-based learning environments focused on improving participants' understanding of a system's dynamics have proven successful in many applications (Morecroft and Sterman 1994). An important next step in this line of research is to use the model presented in this paper as the basis for a similar learning environment focused on improving

managers' understanding of the dynamics of surrounding resource allocation in product development.

5. Conclusion

In this paper I have developed a model of a self-reinforcing process in product development. While the structure analyzed here has been widely discussed in previous work, it has not been the subject of formal analysis. The model developed here leads to a number of new insights. First, the analysis provides boundary conditions on the phenomenon. Second, the results suggest that such processes do not necessarily work in an undesirable direction. Third, under plausible conditions, the system has two stable equilibria and the location of the third, unstable, equilibrium determines the direction of the model's out-of-equilibrium trajectory. The location of the unstable equilibrium is determined by the utilization of development resources in steady state.

While the analysis adds to the existing knowledge, the understanding of this phenomenon is far from complete. In particular, the model presented here could be profitably extended in at least two directions. First, the existing model is highly stylized, and, in the interest of simplicity, a number of restrictive assumptions have been made. Foremost among these is the fixed product introduction date. This assumption is a good approximation of the actual practice in some industries, but not others. Postponing project introduction is an option available to many managers that is not captured in the current formulation. Other interesting relaxations include a process with more than two phases and a more heterogeneous set of development tasks.

Second, the paper has been focused on performance analysis. Additional insights might be obtained via optimization. For example, the analysis shows a trade-off between steady-state performance and robustness to variability. Calculating the most desirable mix may yield additional insight. Similarly, the analysis suggests that the sequence and timing of tool introduction matters.

In an expanded framework that captures a more heterogeneous set of tools, optimally calculating the sequence and timing of their introduction may also yield valuable insight.

6. References

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

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

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

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

Cooper, R.G. (1993). *Winning at New Products*, Reading, MA, Addison Wesley.

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

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

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

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

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

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

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

Oliva, R., Rockart, S., and Sterman, J. (1998). Managing multiple improvement efforts: Lessons from a semiconductor manufacturing site. In Fedor, D. and S. Ghosh (Eds.), *Advances in the Management of Organizational Quality* , 3:1-55., Greenwich, CT: JAI Press.

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

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

Repenning N. and J. Sterman (*forthcoming*). Getting Quality the Old-Fashion Way: Self-Confirming Attributions in the Dynamics of Process Improvement, in Scott, R. and R. Cole, *Improving Research in Total Quality Management*, Sage.

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

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

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

Sterman, J.D. (*forthcoming*). *Business Dynamics*, Chicago, IL, Irwin-McGraw Hill.

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

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

Weick, K.E. (1979). *The Social Psychology of Organizing*, Second Edition, New York, Random House.

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

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

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

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

7. Appendix

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

$$f(s) = \frac{\int_0^T (Min(K - c(t) - r(t), a^*(t))dt}{A}$$

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

$a^*=A^r/(T-t)$

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

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 $C/(1-P_D(s))$, and the desired completion rate, absent a capacity constraint would be constant throughout the model year:

$$c^* = \frac{C}{1 - P_D(s)} \cdot \frac{1}{T}$$

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

$$f(t) = \left(\frac{1}{A}\right) \cdot \int_0^T \left(MIN\left(\frac{A}{T}, Max\left(\mathrm{K} - \frac{C}{1 - P^D(s)} \cdot \frac{1}{T}, 0 \right) \right) \right) dt$$

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:

$$f(t) = \left(\frac{1}{A}\right) \cdot MIN\left(A, Max\left(\mathrm{K} \cdot \mathrm{T} - \frac{C}{1 - P^D(s)}, 0 \right) \right)$$

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)*:

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