"Past the Tipping Point: The Persistence of Fire Fighting in Product Development." Repenning, Nelson P., Paulo Gonçalves and Laura J. Black. *California Management Review* Vol. 43, No. 4 (2001): 44-63.

http://doi.org/10.2307/41166100

Past the Tipping Point:

The Persistence of Firefighting in Product Development

Nelson P. Repenning[1]
Paulo Gonçalves[2]
Laura J. Black[3]

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

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

For more information on the research program that generated this paper, visit
http://mitmgmtfaculty.mit.edu/nrepenning/

1. MIT Sloan School of Management, E53-339**,** Cambridge, MA USA 02142. Phone 617-258-6889; Fax: 617-258-7579; <nelson@mit.edu>.
2. MIT Sloan School of Management, E53-358A**,** Cambridge, MA USA 02142. Phone 617-258-5585; Fax: 617-258-7579; <paulog@mit.edu>.
3. MIT Sloan School of Management, E53-364**,** Cambridge, MA USA 02142. Phone 617-253-6638; Fax: 617-258-7579; <lblack@mit.edu>.

Past the Tipping Point:

The Persistence of Firefighting in Product Development

Abstract

One of the most common syndromes in product development is the unplanned allocation of resources to fix problems discovered late in a product's development cycle or *firefighting.* While it has been widely criticized in both the popular and scholarly literature, fire fighting is a common occurrence in most product development organizations. In this paper we try to answer three questions: (1) why does firefighting exist; (2) why does firefighting persist; and (3) what can managers do about it? The most important result of our studies is that product development systems have a *tipping point*. In models of infectious diseases, the tipping point represents the threshold of infectivity and susceptibility beyond which a disease becomes an epidemic. Similarly, in product development systems there exists a threshold for problem-solving activity that, when crossed, causes firefighting to spread rapidly from a few isolated projects to the entire development system. Our analysis also shows that the location of the tipping point, and therefore the susceptibility of the system to the firefighting phenomenon, is determined by resource utilization in steady state. Taken together, these insights suggest that many of the current methods for aggregate resource planning are insufficient and that managers wishing to avoid the firefighting dynamic must rethink their approach to managing multi-project development environments.

I. Introduction

Both managers and scholars increasingly realize the central role that product development plays in creating competitive advantage. Given this interest, it is not surprising that the design of effective product development processes has received considerable attention from academics.[1] Yet despite this wealth of information and the best intentions of many product development managers, it is quite clear that practice does not always follow theory. A growing number of studies suggest that the development processes prescribed in the literature and the actual sequences of steps used to create products are two very different things.[2] Consider for example, the case of Project X, a major development project that we observed at a large US recreational products manufacturer:

> Project X started a little late. In fact it wasn't initially scheduled in the product plan for the year in which it was launched. But an unmet market need, coupled with the postponement of another project suffering technical difficulties, meant that Project X suddenly became critically important to maintaining customer loyalty and business growth. Because it started late, the project team designing X didn't meet until well into the development cycle and initially the team members, the company's best developers, didn't have time to give X as much attention as it deserved. Furthermore, in order to match the development cycle of other projects launching at the same time as X, the team rushed through much of the project's up-front work and instead spent most of their time on actual design activities.
>
> Rapid progress on design encouraged everyone and led the marketing organization to expand the scope of Project X. When prototypes revealed that the design had some serious flaws, however, it was clear X was in trouble. Significant rework would be needed to address the design concerns and the changes to requirements. With just a few months remaining before the planned launch date, X seemed at risk. But postponing Project X would jeopardize other projects whose designs had accommodated the geometry of X. Postponing X would also wreak havoc on future product plans that were premised on X being part the product line.
>
> With so much at stake, the organization marshaled its considerable resources to make sure X succeeded. Developers worked almost non-stop for months, hand-carrying parts

to testing stations and personally visiting suppliers to resolve issues. More senior managers reviewed X's progress on a weekly basis and provided additional resources where needed. Even the vice-president of development pitched in, attending project reviews and making suggestions.

As it turned out, X ended a success. The product launched on time, and while a few minor issues required patching up once the product was in the field, on the whole the organization felt good about its effort. After all, just months earlier the entire model year was in jeopardy.

The last-minute heroics required to save Project X exemplify one of the most common and costly departures from the development processes prescribed in the literature, a phenomenon we label *fire fighting*. The metaphor of fighting fires is widely used in the management literature, typically referring to the allocation of important resources to solve unanticipated problems or "fires". In product development fire fighting describes the unplanned allocation of developers and other resources to fix problems discovered late in the development cycle. The costs associated with fire fighting are well documented and this mode of product development has been widely criticized in the product development literature.

Yet, despite its costs, it should come as little surprise that fire fighting occasionally occurs in the development of new products. Creating new products is a fundamentally uncertain task, often involving unproven technologies and processes. Further, the alternatives to fire fighting, such as letting a defective product reach the market or canceling it altogether, are often even less appealing. Indeed, many organizations show a remarkable ability to fight fires and transform troubled projects into marketable products. Further, managers often view the ability to engage in fire fighting as an important complement to more formalized development processes. As one manager at the company that created Project X said, "We are a good fourth-quarter team."

There is, however, a significant problem with this view. Our research suggests that the major problems that create the need for extra resources and departures from the standard development

process, although usually attributed to outside forces (changing customer requirements, problematic suppliers), are themselves the result of past fire fighting efforts. Further, while the ability to fight fires is often viewed as a necessary skill, required by the messy reality of developing complex products in a competitive environment, we find that even the isolated use of fire fighting can quickly spread to other projects, driving out the disciplined execution of the desired development process. In other words, organizations that resort to fire fighting in a few projects are likely to soon find that its use has completely replaced the more structured development process. In many organizations, fire fighting *is* the development process. Thus, fire fighting is not a complement to a more structured approach to new product development, but is, instead, an organizational pathology that, left unchecked, can significantly degrade an organization's ability to create high quality products.

In this paper we develop this idea by reporting the results of a system dynamics analysis of fire fighting. Our results arise from over five years of in-depth study of product development in major US manufacturing firms. Our research began with four in-depth case studies of efforts to improve product development at companies in four different industries (automotive, telecommunications, semiconductor and recreational products) from which we identified the phenomenon of fire fighting as a key impediment to performance in product development.[3] In the second phase, one of the original four companies allowed us to study its development process in even more depth. During this phase we conducted in-depth post mortem analyses for each of three consecutive model years.[4] Producing these "launch reports" entailed observing the actual launch of new products in the factory, interviewing every development team that completed a product in the given model year, reviewing problem logs kept during the development process, and collecting related quantitative data such as open concern reports. In total, the three researchers producing these reports spent over fifteen months at the research site. These data and the original four cases were then used as the basis for a series of system dynamics models targeted at explaining both the existence and persistence of fire fighting in product development.[5]

The most important insight arising from our study is that product development systems have a *tipping point*. In models of infectious diseases, the tipping point represents the threshold of infectivity and susceptibility beyond which a disease becomes an epidemic. Similarly, in product development systems there exists a threshold for problem-solving activity that, when crossed, causes firefighting to spread rapidly from a few isolated projects to the entire development system. In other words, firefighting is a *self-reinforcing* syndrome: Once it occurs in one project, it can quickly spread to others, permanently degrading the capability of the entire development system.

Practically, this finding has three implications. First, it suggests that fire fighting, while initially used only when a project gets "in trouble", can quickly become the *de facto* development process. A reliance on fire fighting has a tendency to drive out proper process execution. Second, this also implies that even a *temporary* increase in workload can initiate the firefighting dynamic and cause a *permanent* decline in system performance. While the costs of permanently overloading a development process are now well recognized, the existence of a tipping point suggests that such systems are even more fragile. Even a temporary overload can initiate a vicious cycle of costly fire fighting. Third, our analysis shows that the location of the tipping point, and therefore the susceptibility of the system to the firefighting phenomenon, is determined by resource utilization in steady state. As more work is introduced into the system, progressively smaller shocks are needed to initiate the downward spiral of increasing problems and fewer resources available for their prevention. Thus, development managers face an important trade-off between steady-state performance and the system's ability to accommodate unanticipated changes in resource requirements without descending into the firefighting cycle. Taken together, these insights suggest that many of the current methods for aggregate resource planning, while necessary, are insufficient to prevent fire fighting and that managers wishing to avoid it must rethink their approach to managing multi-project development environments.

The rest of the paper progresses as follows: In the next section we present the main insights arising from our research by developing a simple system dynamics model of fire fighting in product development. In the following section we discuss the psychology of managing and learning in such environments and explain why fire fighting is likely to be so persistent. We conclude with discussion of what firms can do about it.

II. Why does fire fighting spread?

To understand both the existence and persistence of fire fighting in product development, we have developed a number of system dynamics models.[6] While some of the models are quite complex, the most important insights can be developed in a fairly simple framework. So, to understand firefighting, consider the following stylized model of a product development system (Figure 1).

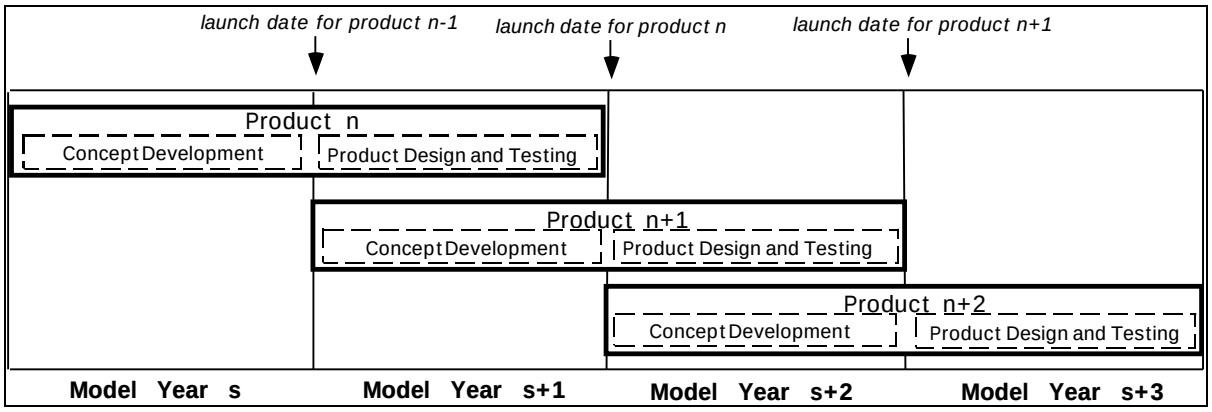

Figure 1: Overview of the Development Process

Suppose an organization has a two-step development process. First, there is a concept development phase designed to identify the customer's needs, develop the product concept, and select the supporting technologies. Second, there is a product design and testing phase, in which the concept developed in the previous phase is turned into an actual product. For simplicity we make the assumption that the organization works on only two products at once, each in a

different phase. As shown in Figure 1, in this simple model the organization launches one new product every year and at any moment in time must divide its attention between completing the design on this year's product and developing the concept for next year's product. Our modeling studies show however that our results are robust in more realistic settings.[7]

To understand the dynamics possible within this simple framework we need to capture the interconnections between the two phases. We first assume that executing each phase requires allocating development resources to complete a set of tasks. In the concept development phase, tasks include activities like documenting customer requirements and making the final selection between candidate concepts. In the design phase, the tasks are related to creating and testing the product. The primary role of concept development activities is to make the downstream design work more effective by, for example, documenting customer requirements, establishing the project scope, and demonstrating the feasibility of the chosen technologies. However, when resources are scarce, the tasks in the concept development phase can be skipped – products can be passed to detailed design with few documented customer requirements, an ambiguous scope, or unproven technologies.[8] The consequence of skipping these tasks, however, is that the probability of completing a design task correctly falls. For example, as we have often observed, an organization can skimp on documenting customer requirements but may have to make substantial changes later in the development cycle when early prototypes reveal mismatches between the initial concept and the customers' desires.[9] We capture these dynamics in figure 2.

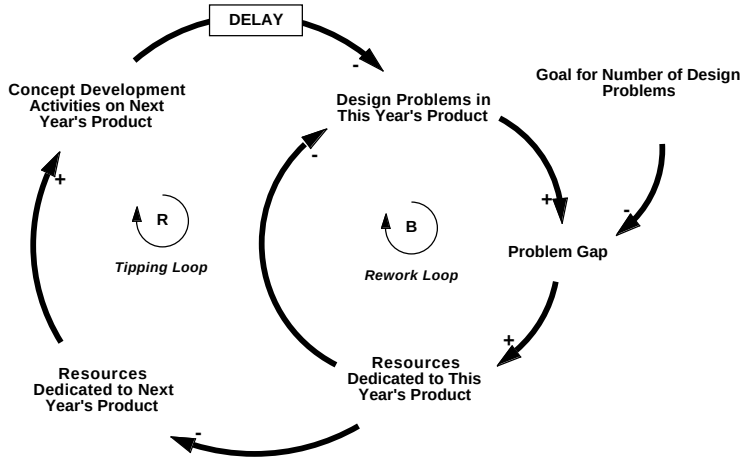

Figure 2: The Feedback Structure of Multi-Project Product Development

The inner feedback loop represents the decision process of managers allocating resources among competing projects in the development process. Our field studies suggest that, when resources are scarce, the projects nearing launch receive priority. Thus, for example, if there is a rise in problems in the design phase of the project nearest launch, then the *Problem Gap*—the accumulation of unresolved issues in design, components that don't work, bugs in the software, etc.—rises, leading the organization to allocate its resources to fixing the design problems. As the *Design Problems in This Year's Product* decrease, the *Problem Gap* falls, and fewer resources are devoted to this year's product. This is the desired balancing dynamic of the *Rework Loop*--the system acts to balance, or counteract, a change in any variable in the loop and reduce the *Problem Gap* to zero.

The often unintended consequence, however, is that allocating more developers to the design phase of the product nearest launch decreases *Resources Dedicated to Next Year's Product.* Fewer resources dedicated to the concept development phase of next year's product reduce the completion rate of the concept development activities for next year's product. This in turn increases *Design Problems in This Year's Product* when next year's product transitions to the

downstream phase. The *Tipping Loop* is a reinforcing feedback that amplifies a change in any variable in the loop. It can operate as a virtuous cycle–more resources devoted to concept development lead to fewer errors in the design phase, freeing still more resources for concept development on the *next* year's product–or it can become a vicious cycle in which more problems in the downstream phase draw resources away from the concept development activities that might prevent those problems in future products.

While simple, the model captures a set of interconnections that have repeatedly emerged from both our field studies and subsequent modeling efforts as central to understanding the firefighting phenomenon. Further, while we have conducted extensive simulation analysis on these models, the core insights can be developed graphically using the *phase plot* (which was derived from the model's structure) shown in Figure 3.[10]

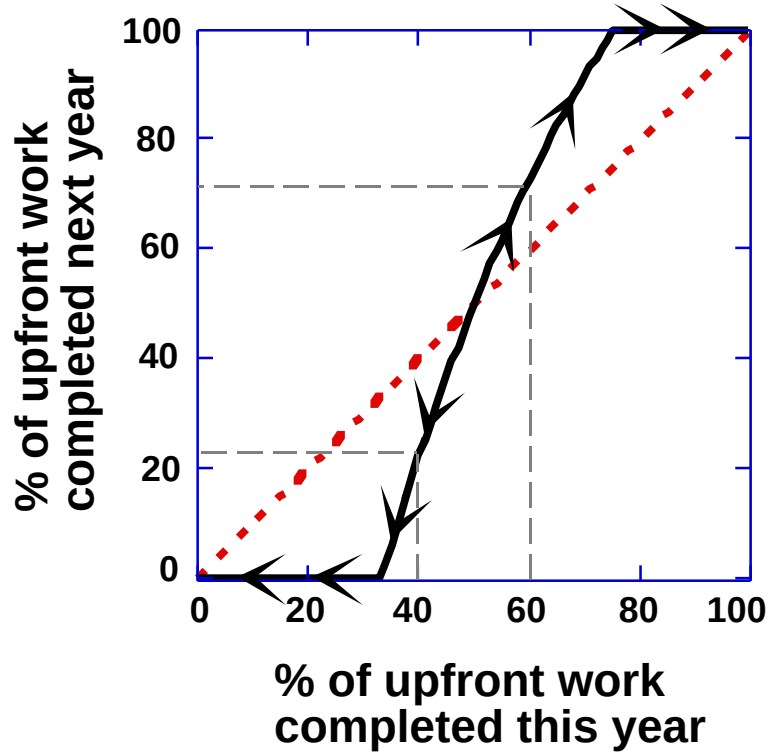

Figure 3: A Phase Plot of the Dynamics of Multi-Project Product Development

The horizontal axis of the phase plot represents the fraction of the concept development work completed in the current model year, while the vertical axis captures the fraction of the concept development work that will be completed next year. The solid, upwardly sloping line shows how, given the structure of the system, the amount of concept development work will evolve from year to year. Thus, to read the plot, start at any point on the horizontal axis, go straight up until you reach the solid line and then go straight left to the vertical axis. By relating what happens this year to what will happen next year, the phase plot provides a simple summary of the system's behavior and provides a useful tool to discuss how product development processes behave over time.

To understand the diagram and its implications for managing product development, suppose that our simulated organization accomplishes about 60 percent of its planned concept development work. In this case, to determine what will happen next year, we can, following the dotted line, read up and over to find that, if it accomplishes 60 percent of the up-front work this year, the dynamics of the system are such that it will accomplish about 75 percent of the up-front work next year. If it completes 75 percent of the early phase work next year, what will happen the year after that? Return to the horizontal axis, start at 75 percent, and again read up to the solid black line and over to the vertical axis to find that in year number three it will accomplish almost 100 percent of the concept development work. In the example so far, then, the system is driven by a *virtuous* cycle. The positive tipping loop shown in Figure 2 works in the upward direction: each year a little more concept work is done, decreasing errors in the actual design phase and freeing downstream resources to complete even more concept development work in the next year. Thus, if this cycle continues, each successive model year will require less fire fighting and the system converges to the point where the organization accomplishes all its desired up-front work every year.

In contrast to this desirable state of affairs, consider another example. Imagine this time that, for whatever reason in a particular year, our organization accomplishes only 40 percent of its planned concept development activities. Now, reading up and over, we find that instead of completing more early phase work next year, due to the greater defect rate in down stream activities, the organization actually completes less—in this case about only 25 percent. In subsequent years the situation deteriorates further in a *vicious* cycle of declining attention to up-front activities and increasing error rates in design work. In this case, the reinforcing tipping loop works in the downward direction. Here the growing need to fight fires in the downstream phase progressively reduces the completion of the early phase work until the system converges to a mode in which concept development work is not done. Here, the organization's ongoing struggle to fix problems in the project closest to launch effectively drives out the desired development process.

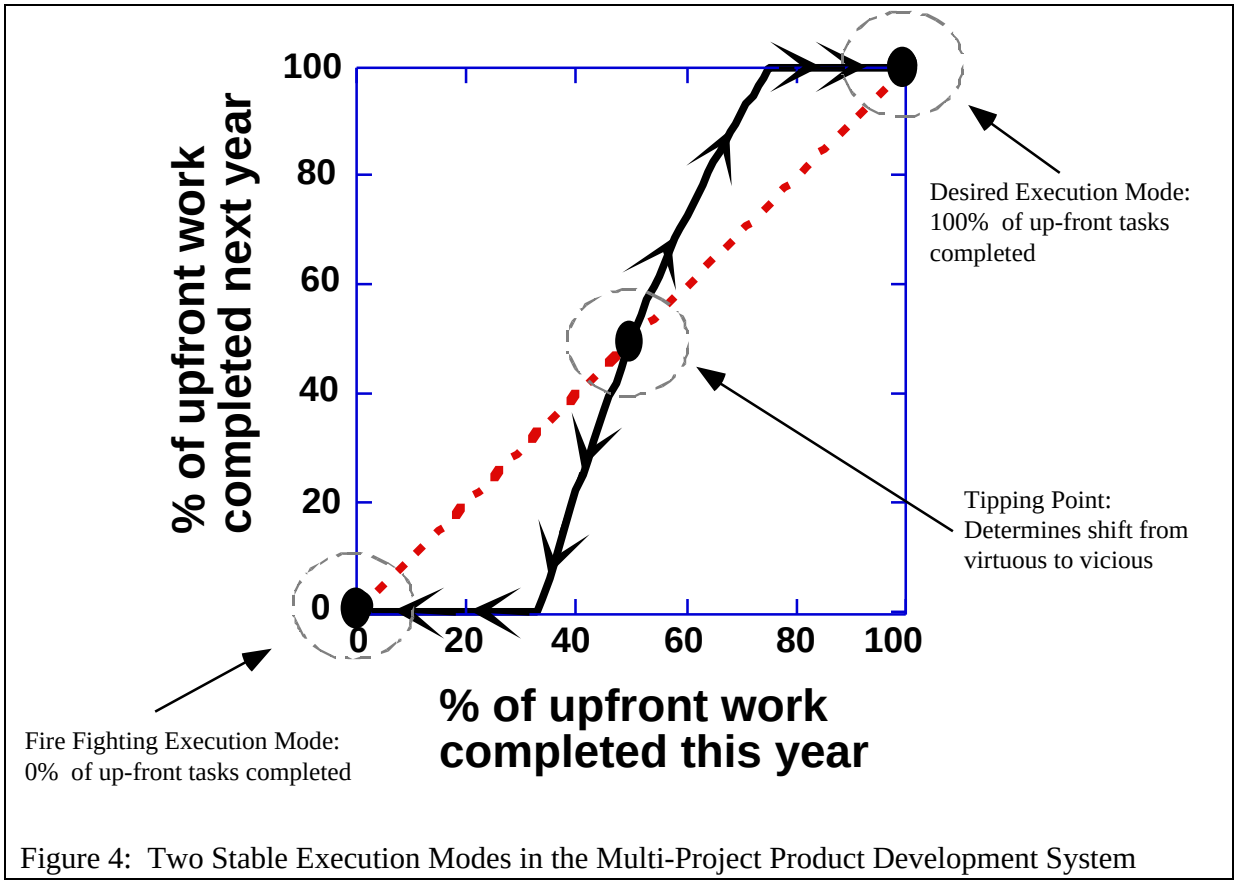

Figure 4: Two Stable Execution Modes in the Multi-Project Product Development System

The analysis offers some interesting insights into the behavior of product development systems. First, as shown in Figure 4, it suggests that such systems tend to converge to one of two execution modes (highlighted by the two solid black dots at the upper right and lower left corners). Either they will benefit from a virtuous cycle of increasing attention to up-front activities and decreasing error rates in downstream work, or they will be trapped in a vicious cycle of increased firefighting and decreased attention to the early phase work that might have prevented those crises in the first place. In other words, two product development systems, despite being identical in every way, could potentially produce very different levels of performance, depending only on the initial attention given to up-front work.

Second, Figure 4 also highlights that these two modes are separated by a tipping point (highlighted by the solid black dot at the center of the diagram where the phase plot crosses the 45-degree line). The tipping point is critical to understanding the dynamics of the system because it represents the fraction of concept development completion at which the reinforcing tipping loop shown in Figure 2 switches directions, changing from a virtuous cycle that eliminates fire fighting to a vicious circle that reinforces it. This means that, if the system starts in the desired execution mode, and for whatever reason, the fraction of concept development work falls below the tipping point, the system will never recover. Instead, it will descend in a downward spiral of increasing error rates and decreasing resource adequacy, eventually becoming trapped in the firefighting mode.

The following experiment highlights the role of the tipping point in determining the system's behavior. We simulate our model under three different scenarios.[11] First, we run the model in steady state to show its behavior in the absence of any outside shocks. In the second case, we simulate the model again, but this time, in model year one we temporarily increase the workload by 20 percent. Such an increase in workload could also represent a particularly challenging

model year or the cost of introducing a new technology. In the case of Project X, the increase represents the actual workload due to a late start and changes in the product scope. Finally, in the third case, we introduce another one-time change in workload, but this time it's a 25 percent increase. The results are shown in Figure 5.

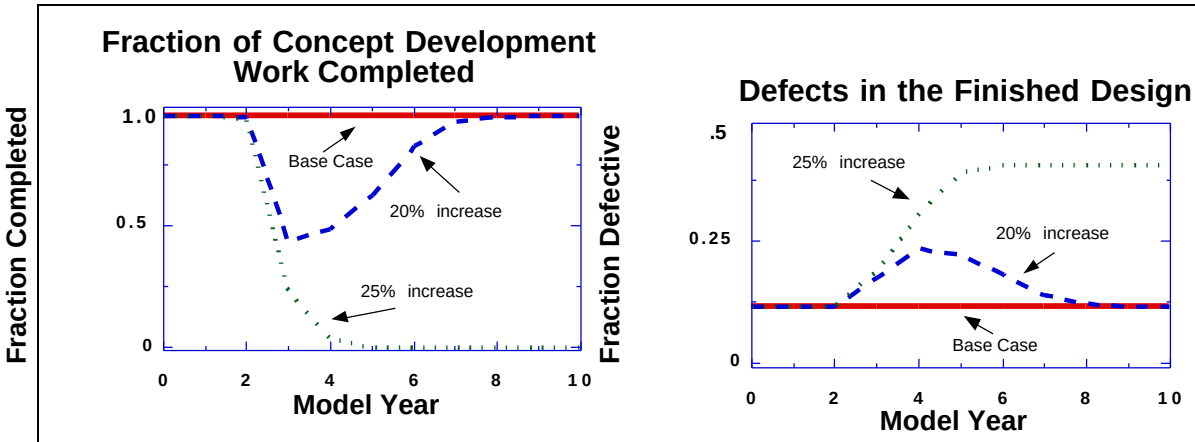

Figure 5: Three Simulated Scenarios—0%, 20%, and 25% One-Time Temporary Increase in Workload

As the figure highlights, before the change in workload, the simulated organization operates in the desired mode: It completes all the concept development work and consequently finds relatively few defects in the downstream phase. Not surprisingly, when the 20 percent increase in workload is introduced, performance declines. The extra workload means that less concept development work is done and, consequently, the quality of the final product suffers. But the decline in quality is temporary; eventually the system recovers to its initial capability. In contrast, however, the system's response to a 25 percent increase in workload differs. This is the mode of operation characterizing Project X, where much of the team's time was spent on actual design activities instead of up-front conceptual work, and, consequently, prototypes revealed serious design flaws. Unfortunately for the real organization launching X, and the organization simulated here, the system never recovers. Instead, it descends into a firefighting cycle in which up-front activities receive little attention and quality permanently declines.

Why is the system able to handle a 20 percent temporary increase in work but not 25 percent? A 20 percent increase is not quite sufficient to push the system over its tipping point. Thus, with time, the system recovers to its original capability. In contrast, the 25 percent increase pushes the system over the threshold, sending the simulated organization on a new path towards the firefighting equilibrium.

What does this mean for the management of product development systems? First, it suggests that a *temporary* increase in workload (and the consequent need to engage in fire fighting) can cause a *permanent* decline in performance. Most managers are now well aware of the costs of permanently overloading a development system. Our analysis suggests that the situation is worse—even a temporary overload can ignite the vicious cycle of self-reinforcing fire fighting. Thus, managers considering the possibility of tackling just a little more work should be aware that such efforts, while possibly successful in the short run, can jeopardize the long-term health of the development system.

The second important implication for managing product development comes in realizing that the level of resource utilization determines the location of the tipping point. The tipping point can be viewed as the balance between workload and resources beyond which an organization *cannot* accomplish all of the tasks necessary to properly execute the projects currently underway. As managers put more projects in the product plan, thereby increasing resource utilization (the percentage of people working at full capacity during all available work hours), the tipping point moves up and to the right. As steady-state resource utilization increases, smaller and smaller increases in workload (no matter how temporary) can propel the organization into the firefighting syndrome. Thus, a fully utilized product development system is also a system constantly on the verge of descent into fire fighting. The phase plots in Figure 6 depict how the location of the tipping point shifts with changes in resource utilization.

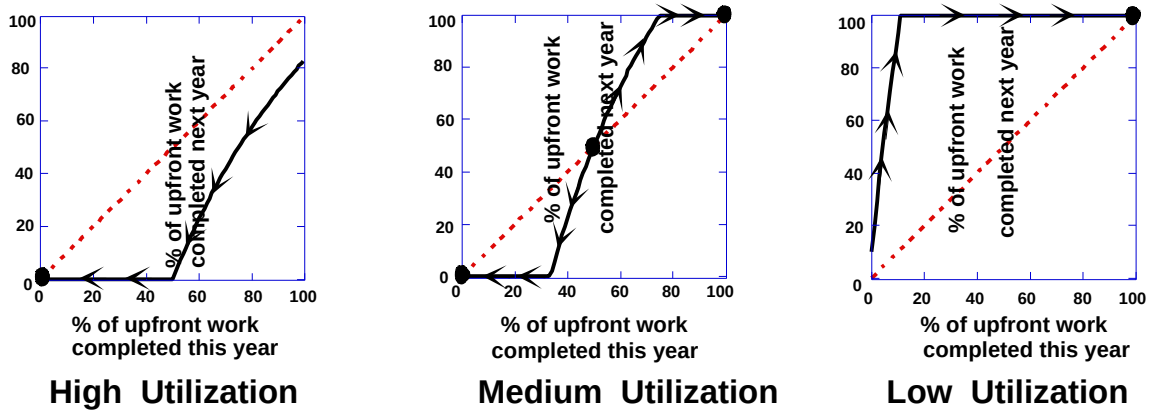

Figure 6: Resource Utilization in Steady State Affects the Location of the Tipping Point

Not surprisingly, managers frequently want to know where their development system is relative to its tipping point. While determining the exact location of the tipping point is difficult, a number of symptoms indicate that resource utilization is precariously high. For instance, interviews at one organization yielded numerous indicators of a system caught in a vicious dynamic of firefighting. From design engineering, we heard, "There are insufficient resources for the work." "Morale suffered [with ongoing problems in a high-visibility project]." "There's a lot of illness among the group." Similarly, manufacturing engineers observed, that "Manufacturing engineers are working round the clock, 24 hours a day" and that "Turnover in manufacturing engineering is high--in five years maybe 15 percent of the original staff is left."

III. Why does firefighting persist?

So far we have tackled the problem of understanding why firefighting exists. As the analysis shows, this syndrome ultimately has its roots in the fact that early-phase tasks, while highly effective in the long run, can be skipped when resources are scarce. Managers who try to "stretch" and accomplish just a little more, while producing success in the short run, often push their development systems over the tipping point and into the downward spiral of decreasing attention to up-front tasks and increasing problems in downstream projects. As mentioned in the

introduction, this scenario repeats itself in numerous organizations, so the question naturally arises, wouldn't managers eventually figure this out? The unfortunate answer to this question is that, without a detailed understanding of the dynamics such a system generates, managers are unlikely to recognize--much less overcome--the firefighting syndrome.

Consider what happens when a manager in a system like the one described above decides to take some resources away from a project in its early, concept development phase, and allocate them to fighting fires in a struggling downstream project. As the discussion above suggests, this decision has two consequences. First, with the additional resources, the troubled *project* improves. This happens quickly, the impact of the decision is easy to assess, and it has a fairly certain outcome. Later, however, these same actions, if they push the development organization over the tipping point and ignite the firefighting cycle, degrade the performance of the product development *system.* In contrast to the improved performance of the specific project, this outcome doesn't happen right away–it can take a number of years for these dynamics to unfold–and the impact of these actions is difficult to assess. Thus, managers making resource allocation decisions in such systems face a "better-before-worse" trade-off in which the positive but transient consequence of the decision to engage in fire fighting 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 psychologists, economists, and others who study decision making have reached a clear conclusion: People do not learn to manage such systems well.[12] In experiments ranging from managing a simulated production and distribution system to fighting a simulated forest fire to managing a simulated fishery, people overweight the short-run positive benefits of their decisions while ignoring the long-run negative consequences.[13] People who play these games produce wildly oscillating production rates and inventory levels, they allow the firefighting headquarters to burn down, and they destroy the fishery through over-fishing.

Applying these results to the product development context suggests both that managers will typically favor downstream projects and that they are unlikely to learn to overcome the undesirable dynamics that such a bias creates.

The problem is still worse. The fact that the health of the development system rests on managing across projects makes identifying and analyzing the source of difficulties challenging. When a development organization is caught in the firefighting cycle, people know something is wrong, but they don't necessarily know exactly what the problem is. When performance is persistently low, managers are more likely to blame the people who work for them than the structure of the development system.[14] Thus, as performance begins to decline in a downward spiral of firefighting, not only are managers unlikely to learn to manage the system better, but they are also likely to blame their designers, developers, and project managers. To make matters worse, the system provides little evidence to discredit this hypothesis. Once firefighting starts, system performance continues to decline, even if the workload returns to its initial level. Furthermore, managers will observe developers spending less and less time on up-front activities like concept development, providing powerful evidence to confirm managers' mistaken belief that developers are to blame for the declining performance. The result is one example of the more general dynamics discussed in another paper in this issue (see "Nobody Ever Gets Credit for Fixing Defects that Didn't Happen"). Management teams become increasingly frustrated with a development staff they perceive as lazy, undisciplined, and unwilling to follow the prescribed development process, and the development staff becomes increasingly frustrated with managers who, they feel, do not understand the realities of launching new products and, consequently, set unachievable objectives.

To summarize, we tip our systems because actions that appear rational at the project level– the use of fire fighting to compensate for unplanned problems–undermine effective management of the development system. And once the system has entered the firefighting zone, escape is

difficult. The firefighting syndrome persists because few actions can push the product development system back over the tipping point into a virtuous cycle of improved process execution.

IV. Policies – What to do?

If your organization is prone to these dynamics, what actions can move you onto the virtuous side of the tipping point and eliminate the need for costly fire fighting? Most important, realize that managing such systems well doesn't come naturally. As highlighted above, the "better-before-worse" behavior created by fire fighting coupled with basic human tendencies and intuitions leads decision makers down the wrong path when managing complex systems like multi-project development environments. Thus, standard managerial responses such as admonitions to "do less fire fighting" or to "focus on the process, no the product" are likely to be inadequate in combating the deeply-seated conditioning created by years of operating in the firefighting mode. Fire fighting is so prevalent precisely because it comes so naturally.

Creating a fire resistant product development system thus requires more significant, permanent changes. However, these need to be executed with considerable care. Both our field data and subsequent analysis suggest that many interventions that appear to eliminate fire fighting actually have the potential to make the situation worse rather than better. Three such strategies are particularly damaging to the long run health of the development process.

First, **don't invest in new tools and processes if you're already resource-constrained**. A common reaction to the low performance created by continuous firefighting is investment in an improved set of development tools and processes. Although successful deployment of such tools would often unequivocally help the organization, they require the development of knowledge and experience. As a consequence, introducing tools actually lowers productivity in the short run while people learn and incorporate them into existing processes. Problems arise when managers ignore this worse-before-better tradeoff. Recall that in the simulation experiments shown earlier, a temporary increase in workload was sufficient to push the system over its tipping point. Introducing a new set of tools into an already over-utilized development system creates a similar set of dynamics.[15] The increase in workload, arising from the additional training, learning, and practice time required to use the tools proficiently, further raises resource utilization, potentially pushing the system over its tipping point. Thus, if new tools are not accompanied by a reduced workload, their introduction is likely to lead to more fire fighting and a further decline in process capability.

Second, **fixing only a fraction of the problems identified late in the development cycle does not prevent the spread of fire fighting**. A second, and often observed, approach to reducing the use of fire fighting is to fix only the "important" problems. However, although fixing a fraction of problems can lead to a short-term decrease in workload, it rarely has the desired effect in the long run.[16] Leaving some problems unresolved usually means that the product leaves

customers unsatisfied, thereby increasing the pressure on the next generation of the product while simultaneously leaving even *less* time to do the up-front work that it requires. Further, and even more damaging, lower quality products can push the organization into a death spiral of declining competitiveness: low quality products generate less revenue to fund research and development, thereby limiting the ability to invest in the development of future products. Leaving defects unattended is not an effective strategy for eliminating fire fighting.

Third, **postponing problematic projects does not prevent the spread of fire fighting.** A final popular response to fire fighting is the postponement of troubled projects under the rationale that the reduced workload can be used to improve both the design of the current project (allowing it to be launched with higher quality) and next year's product concept. But because postponement does not address the root cause of over-utilized resources (postponed projects continue to consume resources until they are launched), it does little but defer the problems to the next model year. In the context of our model, this strategy amounts to doubling the quantity of outstanding downstream work in the model year following the postponement.[17] Having more projects in the downstream phase increases the need for fire fighting, reduces the attention to up-front work in subsequent generations, and further speeds the system's descent into low capability.

Thus, as is common in complex systems, many of the policies that would seem to help alleviate a problem actually exacerbate it. The problem with each of the aforementioned approaches is that, while they address the symptoms of fire fighting, they do nothing to eliminate the root cause. Any policy targeted at eliminating fire fighting must improve the balance between the outstanding workload and the available resources. Building on our field data and subsequent analysis, we find that creating a fire resistant development system requires four complementary policies.

First, **aggregate resource planning is critical to fire prevention**. Although managers are increasingly aware that resources must be managed *across* as well as *within* projects, most development organizations work on too many projects.[18] As our analysis shows, when a system is consistently overloaded, fire fighting quickly becomes the development process. Thus, there is no substitute for a portfolio-level plan that matches the number of ongoing projects to available development resources. Further, the existence of a tipping point implies that slack resources provide a valuable buffer against the uncertainties inherent in new product development. Thus, to prevent fire fighting, the aggregate project plan should allow for uncertainty by maintaining a reserve of unassigned resources. Managers that attempt to utilize fully their development resources are virtually guaranteed to spend much of their time engaged in fire fighting.

Second**, cancel projects with inadequate concepts before they reach the design phase.** While necessary, aggregate resource planning is insufficient to prevent the spread of fire fighting. As discussed above, even a single troubled project, if it draws resources dedicated to other projects, can push the system over its tipping point. Thus, beyond improving the resource planning process, preventing fire fighting also requires minimizing the chance that troubled projects actually reach the downstream phases of the development cycle. The key is to cancel projects *early,* before they have tipped the system. Canceling a troubled project shortly before its scheduled launch means that an organization pays all of the costs of fire fighting and receives none of the benefit of canceling. Our simulation experiments suggest that an aggressive policy of canceling those projects with inadequately developed concepts before they enter the design phase is very effective in preventing fire fighting.[19]

Third**, when a project does experience trouble in the later phases of the development cycle, don't try to "catch up"-- revisit the product plan instead**. Although it can clearly be minimized, the late discovery of some errors is inevitable when developing products with new technologies for new markets. These problems, if they are to be solved, constitute temporary increases in resource requirements with the potential to push the system over the tipping point. Unfortunately, in the organizations we studied, managers rarely accounted for these contingencies in the resource planning process. Despite the fact that such problems often required significant allocations of resources not captured in the initial aggregate resource plan, managers rarely updated their plans when troubles arose, implicitly assuming (as was the case in Project X) that they could "catch up" by just "working a little harder" or "squeezing a little

more." Yet, this was rarely (if ever) the case. Instead, resources were stolen from other projects in the earlier stages of the development process, trapping the development system in the firefighting syndrome. For example, despite the fact that the late discovery of defects in Project X required substantial engineering resources be pulled off other projects, the schedules for those other projects were not changed. As a consequence, those later projects were also moved to the design phase with inadequately specified concepts, creating the need for further fire fighting and, thereby, perpetuating the cycle

Thus, if a significant problem does arise late in the development cycle, ensuring that fire fighting does not spread from the troubled project to the rest of the portfolio requires that managers revisit the plans for *all* of the projects currently in progress. If for example, a given project requires three months of additional work, then the product plan must be updated to reflect the fact that those people working on the project will not be available for another three months. In our model, such a strategy amounts to either extending the model year whenever a product gets in trouble or simply skipping a model year altogether. In either case, the effective reduction in workload stabilizes the system and prevents the fire fighting required by the troubled project from spreading to the rest of the development system.

Finally, **don't reward developers for being good fire fighters**. As discussed in the introduction, many managers we interviewed came to view their ability to fight fires as an important source of competitive advantage. Not surprisingly, many organizations reward and promote engineers based on their ability to save troubled projects. Consider, for example, one senior manager's reflection on how developers in his organizations were rewarded:

> Occasionally there is a superstar of an engineer or a manager that can take one of these late changes and run through the gauntlet of all the possible ways that it could screw up and make it a success. And then we make a hero out of that person. And everybody else who wants to be a hero says "Oh, that is what is valued around here." It is not valued to

> do the routine work months in advance and do the testing and eliminate all the problems before they become problems. What is valued is being able to make a change in the last minute and ramrod it through.

Thus, to be successful, improvements in the resource planning process require complementary changes to the incentive and reward systems. While a number of changes are possible, perhaps the most important one is to simply not let project leaders jump into projects with the hope of saving them at the last minute. As the quote highlights, allowing managers to “save” troubled projects, and therefore receive accolades and benefits, creates a situation in which, for those interested in advancement, there is little incentive to execute a project properly from start to finish. While allowing such heroics may help in the short run, the long run health of the development system is better served by not rewarding them.

There are, of course, major challenges associated with implementing these changes. First, it is clear both from the field studies on which this paper is based and from other studies reported in the product development literature that admonitions to "do fewer projects" and "cancel more," though often heard in R&D organizations, rarely if ever have appreciable impact. Although people may understand and subscribe to these ideas at an abstract level, when decisions concerning specific projects must be made on a day-to-day basis, these general guidelines are simply inadequate to combat the deeply seated cognitive biases created by experience uninformed by a structural understanding of the multi-project development system. Recall the first set of simulations. In the first model year, the system ably accommodates an increased workload (even one that pushes the system beyond the tipping point), delivering the additional work with high quality. A manager making such a decision receives powerful and salient feedback that working a little harder was a good thing to do. Unfortunately, the short-run gain

comes at the expense of long-run performance. In subsequent model years, performance can progressively decline—even though workload returns to normal. Because the decline in performance occurs only with a significant delay, it is less likely to be associated with the earlier increase in workload in any manager's mind. This creates a strong belief that they can always "do just a little more."

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

Given the strength of managers' biases and the incentives facing project managers, we believe that a particularly strict and inflexible version of the cancellation policy offers the highest potential to produce significant improvement in product development. Specifically, we propose that managers establish a strict policy of allowing only a fixed number of projects into the detailed design phase and, more important, uphold a strict policy of canceling projects with unfinished concept development work.

Why might such draconian measures be needed to produce the desired outcome? First, recognize that no development system can produce beyond its steady state capacity for very long. Thus, organizations cancel and postpone projects all the time, but usually only after such projects have consumed substantial resources, thus making these interventions ineffective in eliminating the spread of fire fighting. Implementing such a policy puts a decision that is normally made implicitly and with little forethought –through the day to day actions of developers and project leaders–under the explicit control of managers. Second, as previously highlighted, the psychological biases, institutional structures, and incentive schemes that create the pathologies we studied here are deeply ingrained and unlikely to change easily. A more flexible version of the cancellation policy is subject to modifications by managers that would erode its efficacy over time. If managers are given discretion to increase the number of projects, they are likely to use it, thus putting the system in the undesirable situation from which it started.

Furthermore, such a policy gives both developers and managers a strong incentive to develop competence in evaluating projects mid-cycle and performing thorough up-front work. There is an additional reason, so far not discussed, why organizations may often fail to do up-front work: They may not know how. Organizations that do not invest in up-front activities are unlikely ever to develop significant competence in executing them. Thus, the decision to change the balance of resources not only necessitates reducing the number of projects; it may also require a

significant investment in learning. Individual managers and project leaders are unlikely to make such investments for fear of incurring the inevitable but temporary decline in performance. More senior managers may be unable to create incentives for people to undertake such learning activities.

We believe the policy we propose has the potential for reversing many of the feedbacks that otherwise work against higher performance. Managers required to cancel projects on an annual basis will need defensible criteria for doing so. Project managers will soon learn that, to survive the cut, they must demonstrate the likelihood that their project will succeed. In contrast to the situations we have observed, in which there are few incentives to do up-front work, in a world where some projects are *always* canceled before entering the design phase, project managers face strong incentives to invest in early phase activities and develop clever ways to demonstrate the efficacy of a proposed product far in advance of its detailed design. Similarly, while project managers often portray their projects in the kindest light, more senior managers, if they are required to cancel a certain number of projects, will also have a powerful incentive to develop more objective methods of determining which projects are allowed to proceed into the detailed design phase.

What do these suggested policies mean for Project X? First, given its late start, managers probably should not have agreed to undertake Project X in the first place. Without adequate time

to proceed through the concept development phase, Project X was almost guaranteed to experience serious difficulties during the design and testing phase. Second, when those problems were revealed by early prototypes, Project X probably should have been canceled. By continuing the project despite outstanding issues that required significant additional resources, managers put the entire product plan at risk. Finally, given that it did proceed, despite its major problems, managers should have revisited the product plan and adjusted the schedules for the other projects in the pipeline that were not receiving attention due to the fire fighting required by Project X.

Admittedly, the implications for Project X are grim. When faced with an urgent market need, the company's most talented developers are usually willing to give it their best shot. But, that is precisely why avoiding situations like Project X requires policies that both prevent projects from getting in trouble in the first place and, when they do experience difficulties, ensure the cure is not worse than the disease. Only by vigilantly managing the balance between work and resources will organizations be able to prevent widespread fire fighting and maintain the long-term health of their product development systems.

[1] For example, see:

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

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

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

[2] See for example:

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

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

[3]. For summaries of these case studies see:

E. Keating and R. Oliva (2000), 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;

R. Oliva, S. Rockart, and J. Sterman (1998), 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. Pp. 1-55; and

N.P. Repenning and J.D. Sterman (2000), 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. pp. 201-235.

[4]. Documentation on the "launch reports" can be found in:

C. Morrison, (2000), *Product development process assessment.* Unpublished M.Sc. thesis, Massachusetts Institute of Technology.

M. Vander de Wel, (2001). *Product development process assessment.* Unpublished M.Sc. thesis, Massachusetts Institute of Technology.

[5]. Formal models can be found in:

Repenning, N.P. (2000), "A dynamic model of resource allocation in multi-project research and development systems." *System Dynamics Review*, 16 (3), 173-212.

Black, L.J. and N.P. Repenning (2001), "Why firefighting is never enough: preserving high quality product development." *System Dynamics Review*, 17 (1): 33-62.

Repenning, N. (forthcoming), "Understanding Fire Fighting in New Product Development." *Journal of Product Innovation Management*

Pilon, K. and G. Herweg (2001), *System dynamics modeling for the exploration of manpower project staffing decisions in the context of a multi-project enterprise.* Unpublished M.Sc. thesis, Massachusetts Institute of Technology.

[6] . Repenning (2000), op. cit.; Black and Repenning (2001), op. cit.; Repenning (forthcoming), op. Cit.; and K. Pilon and G. Herweg (2001), op.cit.

[7] . See for example Pilon and Herweg (2001), op.cit.

[8] Unfortunately, according to Cooper and Kleinschmidt (1987), these up-front tasks are often skipped. See Cooper, R.G. and Kleinschmidt, E.J. (1987), "New Products: What separates winners from losers." *Journal of Product Innovation Management* 4(3): 169-184.

[9] The importance of concept development for the success of the product development process is stressed by Cooper (1993), op.cit. Also, this observation is supported by empirical studies: Cooper, R.G. and Kleinschmidt, E.J. (1986) "An Investigation into the New Product Process: Steps, Deficiencies and Impact." *Journal of Product Innovation Management* 3: 71-85.

[10] In particular, the system in Figure 2 serves as the basis for the development of a dynamic non-linear model. A reduced form of this model provides an equation relating the fraction of concept development tasks completed in a given year, the amount of work in the system, the annual capacity, and the fraction of concept development tasks completed in the previous year. For details see N. P. Repenning (*forthcoming*), op. cit.

[11] For additional documentation of the model discussed here, see N.P. Repenning (*forthcoming*), op. cit. Also, a running version of the model can be downloaded at http://web.mit.edu/nelsonr/www/.

[12] For example, see:

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

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

E. Moxnes (1998). "Misperceptions of Bioeconomics," *Management Science*, 44 (9):1234-1248.

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

[14] See Repenning, N. and J. Sterman (2000), op. cit.

[15] . The role of new tools in initiating fire fighting is formally analyzed in Repenning (2000), op. Cit.

[16] . This policy in analyzed in, Black and Repenning (2001), op. Cit.

[17] . This is also analyzed in, Black and Repenning (2001), op. Cit.

[18] S. Wheelwright and K. Clark (1995), op. cit.

[19] Black and Repenning (2001), op. cit.