Abstract

Many companies have established stage-gate processes in efforts to ensure that product development processes remain on track and on schedule. In addition, many scholars advocate that work for multiple years' new products proceed concurrently with staggered launch dates. In a situation of overlapping product development efforts with stage-gate milestones, engineers often become a scarce resource that must be allocated among projects in various phases of development. In this context, I study the effects of allocating engineers to the main product development activities of design, testing, and problem-solving. While, in keeping with a stage-gate process, practitioners and researchers often structure their work and methods as though design and testing were sequential phases, many acknowledge that design and testing emerge iteratively. In the context of multiple projects, unanticipated cycles of testing and re-design in one project can jeopardize other efforts' schedules, resources, and management attention. This paper develops a simulation model in which to explore the long-term effects of testing and prototyping in an ongoing stream of new product development. Based on research conducted at a manufacturing company, I hypothesize that actions undertaken rationally to "solve the most important problems first," such as allocating as many engineers as necessary to fix problems in the products nearest to launch, systematically undermine testing activities in other development projects. The analysis focuses on identifying policies for resource allocation that prevent systematic underinvestment in early problem identification. Policies evaluated include the timing of testing and prototype builds; priority of testing among other development activities; correcting a fraction of the defects found through testing; and postponing and canceling development efforts that fall behind schedule.

Introduction

According to a recent survey by the Product Development and Management Association, about 60 percent of U.S. companies use stage-gate processes to ensure that product development processes remain on track and on schedule, and firms identified as using "best practices" for new product development have established stage-gate processes to a greater degree. (Griffin 1997). These methods often set forth a series of cross-functional checkpoints for verifying that new product development projects are meeting certain criteria such as specification-testing, cost per unit, or readiness well before the product's scheduled introduction to market.

With much attention on within-project management such as stage-gate or milestone methods, fewer companies have implemented coherent plans to manage across-project management. As customer expectations for new and improved products rise, more companies are overlapping development of successive projects in an attempt to produce a continuing stream of new projects for their customers. Many researchers simply take for granted that work on multiple years' new products must proceed concurrently (Wheelwright and Clark 1992), and other scholars advocate that work for multiple years' new products, particularly for successive product families, proceed concurrently with staggered launch dates (Meyer, Tertzakian, and Utterback 1997).

In a situation of overlapping product development efforts with stage-gate milestones, engineers often become a scarce resource that must be allocated among projects in various phases of development. Griffin's (1997) report on best practices in product development asserted the need for "tangible and visible top management support of NPD [new product development].... This includes having a rational process at firms for allocating resources across projects...." While many acknowledge that product development organizations are often "overloaded" with a number of projects far exceeding the resources of the group (e.g., Wheelwright and Clark 1992), few offer clear advice about how to prevent undertaking too many projects (other than "Don't do it!"), and still fewer provide clues about how to recognize when a development organization has bitten off more than it can chew (Smith and Reinertsen 1998 offers a welcome exception).

Even strategies for trimming the project list regularly or assuring adequate resources for projects at initiation may not prove sufficient. As certain projects wax in unforeseen problems and others wane in resources, managers face a dynamically complex situation in which no single strategy is obviously optimal and no single action can avert with certainty snowballing troubles. As multiple ongoing projects become more common, people are finding that managing them well requires more than incrementally managing individual projects.

One manifestation of these poorly understand dynamics emerged from a program manager at one of our research sites: "We don't know what we don't know until it's *real* late," he said, explaining that critical issues in product development projects consistently surface in the last months before the scheduled market introduction. Design changes to address the critical issues then throw into furious activity development engineers, component suppliers, manufacturing engineers responsible for tooling, and people working on dependent projects, straining communications and jeopardizing product quality. By the time the product hits the assembly line on the company's annual launch date, according to one wry comment, everyone involved is holding up "the company's victory symbol--crossed fingers." This particular manufacturing company has during the last decade established a solid development method, introduced more rigorous project reviews, and improved in myriad ways its new product development process. Despite these advances, and despite increasing its engineering staff, creating and staffing a developmental purchasing group (separate from its operational purchasing group), and giving heightened focus to new product efforts, the pattern of late testing--and late changes--persists.

Acknowledging that a situation is complex does not provide the people doing product development with the tools they need to manage their work--admitting "It's a system" is not enough. Some researchers have long asserted humans' difficulty in recognizing and dealing effectively with dynamic complexity and the need for formal models to aid us in managing dynamic operations (e.g., Forrester 1961, Sterman 1989, Sterman, Repenning, and Kofman 1997). Because product development 's work-in-process, unlike manufacturing's, is not necessarily visible (especially in the early stages of development) and therefore not salient, and because information about the status of new product efforts often relies on successful translation across functional boundaries, it is likely that the dynamic complexity of multiple-project development environments poses one of the most confounding areas managers can face. Yet although multiproject development environments are increasingly important to companies' market viability and almost certainly poorly understood, few formal models exist to aid understanding of the dynamics that can emerge and under what conditions certain troubles may appear.

One exception is Repenning’s (1999) work with a stylized model that showed that, when development resources are scarce, overlapping projects become interdependent, as problems in one project can steal resources needed by another. In his model, even a temporary imbalance between work and resources--such as one bigger-than-usual launch--can lead to a permanent decline in quality. Similarly, this paper develops a simulation model with which to explore how within-project and across-project dynamics might relate to each other. Here, however, I represent testing explicitly as resource-constrained (rather than as an uncapacitated delay, as in Repenning's 1999 paper) in order to explore the long-term effects of testing and prototyping in an ongoing stream of new product development. Based on field research at a Midwest manufacturing company, I hypothesize that actions undertaken rationally to "solve the most important problems first"--such as allocating as many engineers as necessary to fix problems in the products nearest to launch--systematically undermine testing activities in other development projects.

The next section reviews customary uses of testing in product development, and the following section provides background on the field site on which this model is based. Then I describe the model's structure. In the "base case" simulation I observe that a temporary increase in workload can lead to a permanent decline in quality. I introduce structure to represent a prototype build and observe its slightly stabilizing effect. The exploration then simulates various priorities for allocating resources in an effort to discern what, if anything, might disrupt the permanent decline in quality. The surprising result is that allocations that minimize resources to testing create a more robust system in the long-run. Rather than decline gradually to a permanently lower level, product quality dips sharply but then recovers to its original high level. Analysis reveals that, if testing doesn't reveal the as-yet-unidentified problems in the product about to launch, engineers are allocated instead to early phase work on the product scheduled to launch the following year. This leads us to explore a variety of alternatives in responding to identified problems, including postponing a portion of the model year's projects; allocating resources to fix only a fraction of the defects found; and canceling incomplete work altogether.

Conventional Views of Testing and Prototyping

Practitioners and academics agree that testing and prototyping are necessary steps in creating a new product of high quality. A classic product development textbook by Ulrich and Eppinger (1995) identifies "Testing and Refinement" as the fourth of five phases portrayed diagramatically and textually as though they are chronological and sequentially dependent. Here, as in other writings on product development, testing proceeds based on the output of the "detailed design" phase, which yields drawings, computer files describing geometry and production tooling, and specifications for purchased parts (Ulrich and Eppinger, p. 17). The phase preceding testing, design, is also the one in which manufacturing engineers design tooling and design engineers define part geometry and choose materials. But if the testing phase identifies problems, few design changes can be implemented without changing the geometry, material, or production of the part: A common paradox is that we test to find errors but often neglect to allot time sufficient to fix those errors and revisit the assumptions that created them.

Most broadly, we assume that testing identifies problems, which are then fixed; therefore testing ought to improve the quality of the product. This assumption applies to various goals and levels for testing, such as field testing, reliability testing, life testing, and performance testing, and testing functionality by component as well as testing system functionality. Furthermore, we apply this assumption to prototyping as well, which almost all people working in product development include as part of the testing phase. Ulrich and Eppinger (1995) recommend at least two prototype iterations during the testing and refinement stage, and Wheelwright and

Clark (1992) point out that, since prototyping conveys information across functions, prototyping can "achieve integrated problem-solving...[and] surface broad issues that cut across disciplines." (Wheelwright and Clark, p. 146). If, however, system-level testing and prototyping succeed--that is, if they identify cross-functional issues--then project leaders may find themselves with problems whose physical and departmental dependencies are difficult to untangle, with too few resources to address the problems comprehensively, and with little to guide them in discerning the best use of the time remaining before market introduction. Without a clear-cut strategy for prioritizing problems and resource allocations, product developers lapse into a "fire-fighting" mode.

Several model-based research efforts have put forth advice about how to reduce unwanted iterations of design and testing. Eppinger's Design Structure Matrix (described in Ulrich and Eppinger 1995) seeks to order development activities so that each phase receives the information from preceding phases needed to conduct its work. Suggestions for determining the optimal amount of overlap for concurrent development activities and for streamlining design interfaces can enhance the productivity within an individual new product effort (Krishnan, Eppinger, and Whitney 1995). But normative approaches do not address the problems that arise if, as occasionally in practice, undesirable iterations in a project become necessary. Neither do these approaches offer insight on how undertaking multiple ongoing projects may alter optimal practice within a single project. In contrast, this paper explores the logical consequences of the additional rework cycles that testing and prototyping create and how those consequences play out in the context of a continuing stream of product development work.

Overview of the Field Site

In 1998 I spent three months at a midsized Midwest manufacturing company, conducting interviews and observing production activities and meetings in order to prepare an assessment of the product development process and launch (at the company's request). The report summarized 33 interviews, including 10 project teams and 15 process teams in six facilities. Interview participants represented a variety of departments, including marketing, developmental and operational purchasing groups, design engineering, data processing, cost accounting, and manufacturing. Among the top 15 themes identified as "what went wrong" during development for the model year launched, design instability ranked fourth, and late testing and insufficient testing resources ranked ninth. (Unless otherwise indicated, quotations following are from the 1998 company report.)

The data collection also included reviewing problem logs kept during development, at testing facilities, and on the assembly line, and the number of parts officially authorized by engineering at various points during development. By comparing these with previous years' records, a consistent theme emerged: The organization identifies problems in new product efforts later in the development cycle, rather than earlier. The newly surfaced problems absorb resources otherwise devoted to projects scheduled for later launch dates. Solving the urgent problems necessitates changes to product design--both design and manufacturing engineers consistently decry "late changes," design alterations that take place after a key prototype build occurring about 8 months before launch. This prototype is, according to the company's product development methodology, to signal the *end* of the design. A more recent assessment (prepared by another researcher) of the company's development and launch activities pointed out, "Builds serve as targets and launch is the only milestone. Problems are neither resolved nor expected to be resolved until launch" (company report 1999). This story of late problem identification and still later changes and of the indirect but inevitable disruption to other projects led us to investigate more thoroughly the role that testing and prototype builds can play in an ongoing stream of product development activity.

Symptoms of limited testing capacity emerge from interviewees' comments such as, "We were unable to get vehicles to put test components on," "We didn't even have equipment to measure heat--just 'felt it getting hot,'" and engineers repeatedly encountered "not enough testing equipment or people." Complaints of late decision-making include observations of late part changes, migrating scope and specification criteria, "weak follow-through on project checkpoints"; one project manager's lament, "I wish I knew we had that project earlier"; and another engineer's assertion that the "product was really managed by 'critical moments.'" One of the most persistent complaints recorded in the interviews was that engineers felt as though they were given mandates to violate the development methodology that they are supposed to adhere to in order to assure smooth product preparation and launch. Rather than denigrate the methodology prescribed, engineers expressed dismay at not being given a chance to follow the company's normative time-lines and insisted that all development groups and functions should be held equally accountable to adhering to its information and timing requirements. I suspect that the refrain of "not following methodology" does not reveal managers' and engineers' disregard for process but rather offers evidence of an underlying structural problem: that the organization is approaching iterative work with a sequentially specified process. With these themes from one company's experiences, then, I built a simulation model with which to explore the timing of testing and alternative allocations of resources that can surface critical issues earlier in the development cycle so that resources are not subsumed into crises in the work about to launch.

Description of the Model

For simplicity and tractability, this model assumes that the product development organization works on projects in two main phases during a period of two years. The company studied launches new products on an annual schedule, and it works on two major groups of projects or "model years," each at a different stage of development, at any given time. Figure 1 depicts the scheme of overlapping product development efforts.

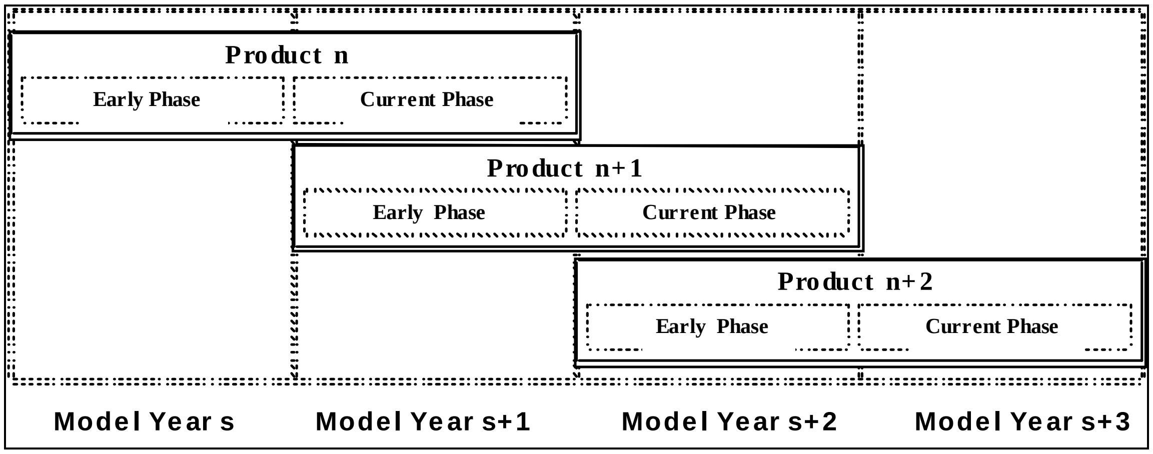


Figure 1: Overlapping Product Development Efforts

In this model the unit of work is the part. Each model year begins the "early" phase, also called Year -1, with a certain number of parts that must be designed, tested, and possibly revised in order to assure high quality. The workload of parts to be designed is akin to a company's compiling developmental part numbers assigned to components undergoing design for a portfolio of projects to hit the market at the same time--it gives a rough idea of the size of the model year's work under way. When the company launches Product *n* in Model Year *s* (market introduction takes place at a fixed time each year), then Product *n+1* transitions from the "early phase" to the "current phase," and work on Product *n+2* begins its "early phase."

The Flow of Work

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

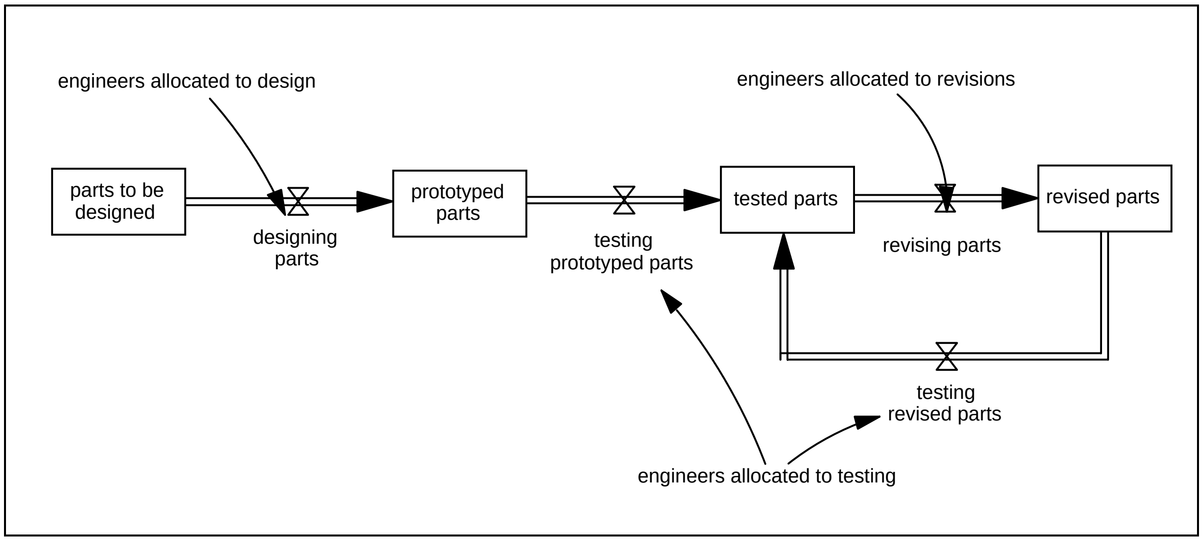


Figure 2: The Flow of Work Within a Development Phase

At each product launch, work in the current phase enters the market; work in the early phase becomes current work, and the early phase begins design of new parts for the model year to launch in 24 months. When this annual transition occurs, parts in each state of the early phase move directly to the corresponding state in the current phase. Thus parts that were tested in Year -1 do not need to be tested again in Year 0, unless testing discovered a problem that necessitates revision.

In contrast to Repenning's (1999) work, which portrayed testing as an uncapacitated delay, here testing is resource-constrained. In this model, I assume that testing takes place under two conditions: by engineers allocated to testing or by a prototype build that takes place during the current phase. When testing is resource-constrained, testing doesn't take place unless there are parts whose design or revision is complete but untried *and* engineers are allocated to testing those parts. When a prototype build occurs, all the designed parts are effectively tested within a one-week period (with no allocation of engineering resources).

Elaborating on Key Assumptions

The distinction between "early" and "current" work (I sometimes call the current phase Year 0 work) is critical for two reasons. The first concerns the probability that engineers make mistakes when designing a part. In Year -1, or early work, the probability of a design error is one-third that if the part is designed during Year 0. I believe this difference appropriately reflects the benefits of early design completion accruing from fewer complicating interdependencies among parts when the design is in an earlier stage of completion and from a longer lead time for choosing materials and processes (although dependencies, material selection, and process design are not explicitly represented in the model). For simplicity's sake, only one kind of "error" is represented in the model. It assumes that, when the simulated company launches a "defective" product, the remaining errors are cosmetic or mechanically nonoptimal but not threatening to the product's function or the user's safety.

Second, the distinction between Year -1 and Year 0 work becomes important in resource allocation. Priority for allotting engineers to the three main activities of designing, testing, and problem-solving is sharply influenced by how near a project is to its scheduled launch date. This model represents an organization in which resources are scarce and the slate of new products is ambitious. There is little slack in the system. The model depicts the company's bias for allocating more resources to the project nearest to launch by continuously estimating the total amount of work to do and allocating as many engineers as needed (or as many as are available) to the following activities in descending priority:

- Designing Year 0 parts;
- Testing Year 0 parts;
- Solving problems in Year 0 parts;
- Designing Year -1 parts;
- Testing Year -1 parts; and
- Solving problems in Year -1 parts.

Since this way of prioritizing is similar to those in the phases outlined by Ulrich and Eppinger (1995) I call this the "Textbook" Allocation. It is a rational, robust method for allocating engineers to work: If there is no work to be done in a particular activity, no engineers are assigned to it. If the organization is feeling pressure from an overload of incomplete or problematic parts when launch is imminent, it pulls engineers from other activities onto the more urgent efforts.

Additional simplifying assumptions include the following.

- Parts are treated as though they are independent; a problem in one part doesn't necessitate changes to other parts.
- Engineering resources are fungible; engineers may be allocated to any of the three activities on any part.
- There is no cost to productivity when engineers are moved from one activity to another.

The "base" workload in these simulation is 700 parts, comparable to what the field site studied actually undertakes. Diagrams and an equation listing of the model are attached as an appendix.

Most of the following simulation experiments explore the effects of a one-time increase in workload and analyze the long-term as well as short-term impacts on the quality of products launched. In each of the simulations below, the one-time (i.e., one model year's) increase in workload takes place at the beginning of the third year of simulation. Because the development cycle represented is two years, that higher-content model year launches at year 5. I believe that a one-year increase in parts-to-do poses a reasonable test of the product development system's robustness. One might consider that every fifth or sixth model year, a dramatic revision of the platform takes place, and this revision is the temporary increase in parts. Alternatively, one could think of the transient increase in workload as resulting from implementing a new development process or technology, rather than a simple increase in parts. In any case, it seems appropriate to model a surge in workload in order to understand the range of possible effects on the system and its output. Another possible interpretation is that a one-time pulse in workload results from unusual events in the company's life. For example, in the company studied, two new plants opened during one year, without reduction of the product development workload or addition of staff; even though the product development facilities did not move, all the manufacturing contacts for organizing new parts for the next model year and assuring a smooth launch were unable to give timely attention to the developing projects. Several project teams interviewed complained of poor new parts management in the plants. One product development team whose output entered production at one of the newly opened plants complained specifically of "smooth manning with a spiked workload."

To assess the "health" of the product development system, I focus on quality, proxied by a percentage of the new model year parts designed and launched correctly. Parts that were incompletely or incorrectly designed (in other words, parts with errors) detract from quality. This measure of the system is also based in fieldwork. One interview yielded the assertions that "quality is the default variable that moves" and that the team "can't move any [other] variables [including] time, people, or dollars." Another project team suggested that expectations for low-cost, on-time delivery came "at the expense of quality."

Model Results

Initial Results: Tilting Into a Regime of Lower Quality

Figure 3 depicts a series of simulations in which the temporary increase in workload is raised incrementally.

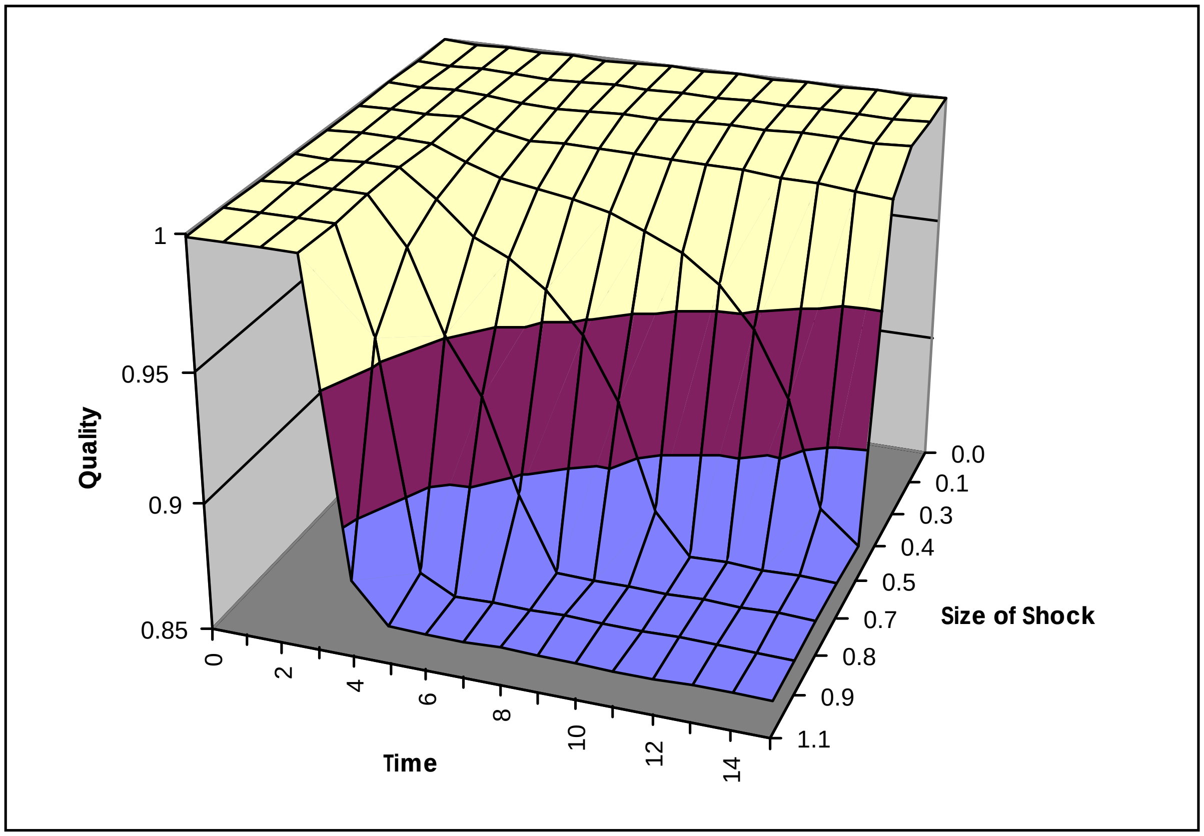


Figure 3: Quality, With a Temporary Increase in Workload in Year 3, "Textbook" Allocation with Resource-Constrained Testing

The vertical axis shows quality as a percent of parts launched without error; the horizontal axis shows time in years; and the third axis (going "into" the page) indicates the temporary increase in workload as a percentage of the base workload. If we look at the three-dimensional surface when the temporary increase in parts is 0, we see that throughout the 15 years simulated the quality of the product launched remains at 100 percent. As we follow the Size of Shock axis to about 40 percent, however, we observe a slight dip in quality about year 5, when the higher-content model year hits the market. Even more interesting, over time quality continues to decline, even though the workload of every other model year is again its original value of 700 parts. For higher increases in workload, we observe a marked decline in quality; the higher the temporary workload, the faster quality declines. Quality stabilizes, however, at about 86 percent, the maximum quality achievable when the organization conducts no work during the early phase but does all design, testing, and (not quite enough) problem-solving in the 12 months preceding launch.

An array of phase plots helps explain concisely why quality can decline permanently though the strain on the workload-resource balance is temporary. In Figure 4 , the horizontal axis shows the fraction of parts prototyped during the early phase for Model Year *s-1*, and the vertical axis shows the fraction of parts prototyped during the early phase the following year, for Model Year *s*. Points along the 45-degree line indicates an equilibrium--the company will consistently complete that fraction of early phase work year after year.

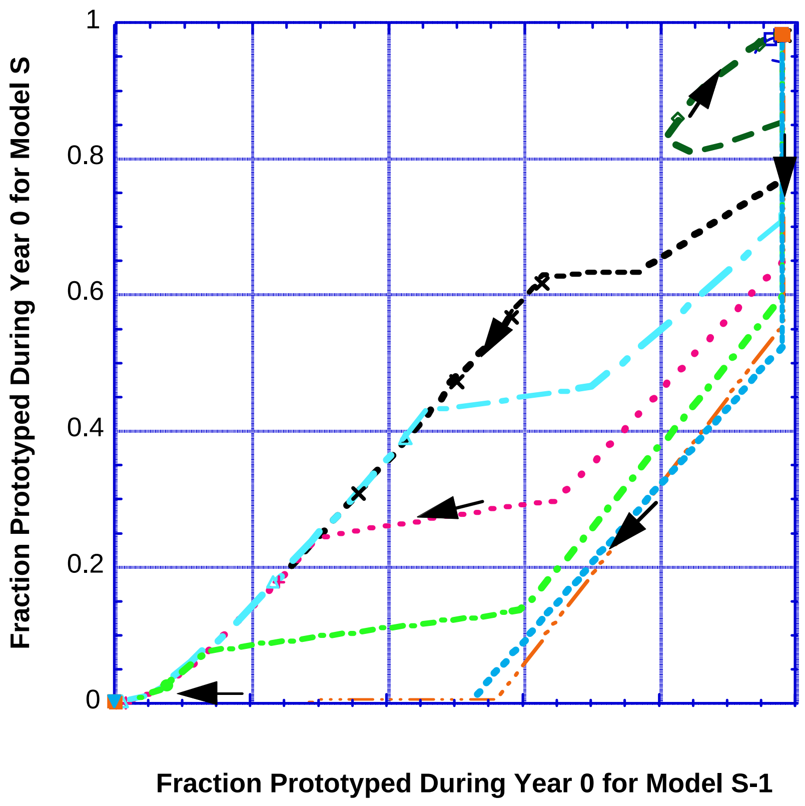


Figure 4: Phase Plots, "Textbook" Allocation With Resource-Constrained Testing

At one extreme (the top right), we can see that, if the organization designs in the early phase 100 percent of the parts for a model year, it can do the same the next year. At the other extreme (the lower left), we see that if the organization designs no parts during the early phase for one model year, it cannot "catch up" and design some parts during the early phase for the next model year.

Consider that, if no parts are prototyped during Year -1, then all the design must be completed during Year 0. To accomplish this, all resources must be allocated to design in Year 0. Then, because the probability of error in design is higher for work completed during the 12 months before launch, all resources must be devoted to solving problems identified through testing. Even with "all hands on deck," the group cannot correct all the defects before the launch date arrives, and the new model year, perhaps with missing bolts, leaky oil pans, or unevenness in paint meets the market. With the launch of that product, engineers face anew the task of designing all the new parts for the next model year within the next 12 months, since none of them were allocated to early phase work. In the mid-range, we see that when the organization completes in Year -1 more than about 75 percent of the prototyping for a model year then it can recover to complete slightly more of the prototyping in the early phase for the next model year, and even more for the following one. In this way, it returns to the equilibrium at which nearly 100 percent of the design for a model year takes form during the early phase of development. Once the early phase prototyping drops below a certain threshold, however, the percent prototyped in Year -1 declines year over year.

One might call this descent into a lower regime of quality "tilting," to indicate that the initial equilibrium, in which quality is consistently high, is unstable; with sufficient perturbation, the system moves to a more stable equilibrium. A reduction in workload can lead to 100 percent completion of the early phase work each year, while an increase in workload can lead to 0 percent completion of the early phase work each year. It is important to note that, once quality declines, even though the workload returns to its initial level, quality does not improve. It is possible for a multi-project development environment to slip into a regime of lower quality and remain there, even though engineers work as diligently as ever. Understanding why this decline can occur--and why it can persevere--necessitates exploring the dynamics resulting from resource allocation priorities.

The Dynamics of Resource Allocation

Since there is little slack in the product development system, in order to complete all the work at a high quality some work must take place during Year -1, when the percentage of work done right the first time is higher. A simplified representation of the model used in this research, shown in Figure 5, portrays the dynamic pressures (indicated by loops) that bear on resource allocation. (In this streamlined diagram, each phase's parts to be designed, parts to be tested, and parts to be revised are aggregated in the stock labeled "work to do.") Beginning at the lower left of Figure 5, we see that more Year 0 work to do leads to an increase in resources allocated to Year 0. This increases the Year 0 work rate, which reduces the accumulation of undesigned, untested, or unrevised parts in Year 0. This process of balancing resources to work is referred to as the Year 0 Work Completion Loop. A similar process exists for Year -1.

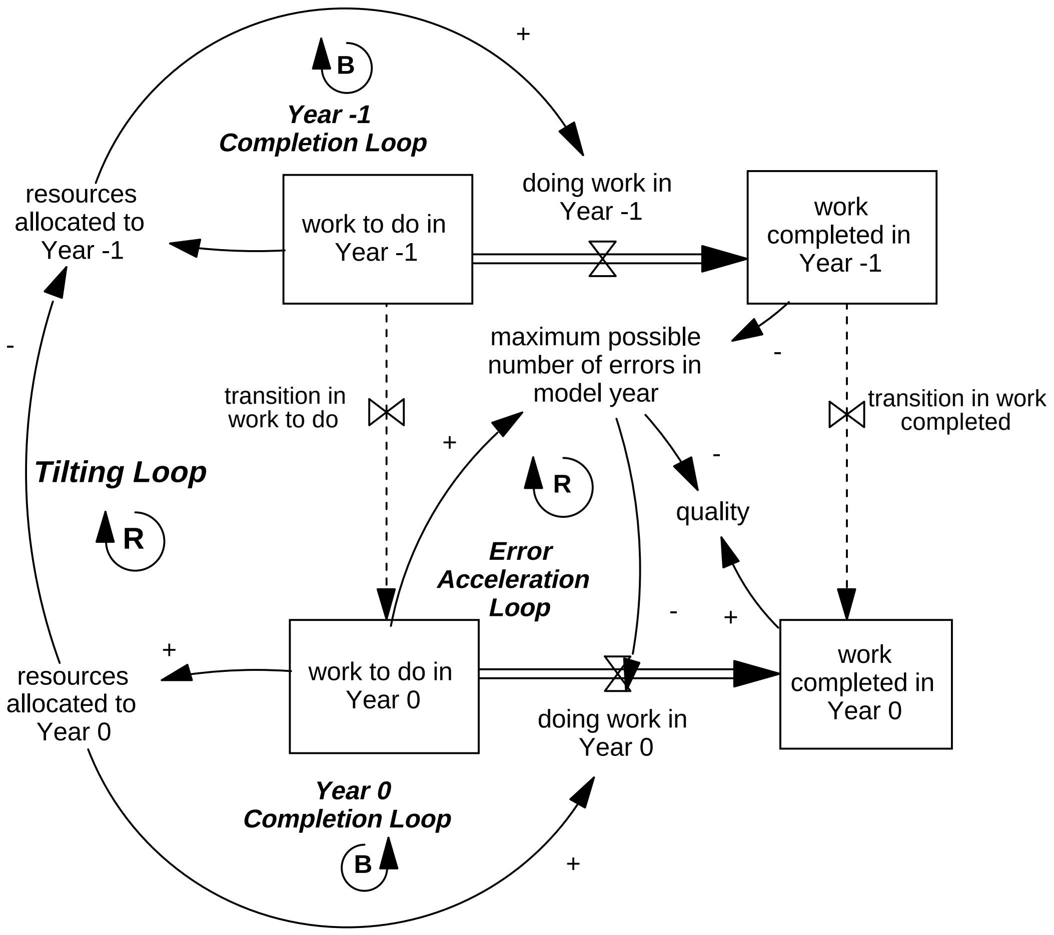


Figure 5: Dynamics Arising from Resource Allocation

As resource allocation to Year 0 increases, however, resources allocated to Year -1 necessarily decrease, since resources are finite in this model. Then less work is accomplished during Year -1. At the next model-year transition, incomplete work in the early phase suddenly becomes incomplete work in the current phase. Since Year 0 work receives priority, resource allocation to Year 0 increases again, perpetuating the lower Year -1 work rate. This phenomenon is "tilting"--when the reinforcing dynamic that shifts resources *between* phases dominates the *within* -phase completion dynamics and drives more and more work to be completed only in the last months before launch.

Figure 5 shows an additional aggravating dynamic, the Error Acceleration Loop, which can serve to destabilize further the development effort and perpetuate the lower quality output. As stated earlier, work completed in the early phase generates fewer errors than work completed in the 12 months immediately preceding launch. As incomplete work in Year 0 increases, the possible number of errors increases--the maximum number of errors in prototyped parts can result if all parts are designed during Year 0 and none are designed during Year -1 (although if parts-to-do are not designed at all, quality declines still further). When errors in prototyped parts are discovered through testing or a prototype build, engineers must be allocated to revise the defective parts. This slows the overall completion rate for work in the Year 0 phase. As the work completion rate decreases, Year 0 work-to-do is higher than it otherwise would have been and the maximum possible number of errors in the model year increases. This reinforcing feedback drives more work to be completed later in the development process.

Based on the parameters here, if the simulated company tries to complete all a model year's work within the 12 months before its launch, it does not have sufficient resources to design (with a higher probability of mistake) all the parts and then fix all the problems it creates. A one-time increase in parts per model year can "tilt" a healthy product development organization into doing a greater percentage of a model year's work during the 12 months before launch each year. Once the company is caught in the "fire-fighting" mode, since it favors completing all Year 0 activities before undertaking any Year -1 efforts, it is caught, continuing to operate in a lower-quality regime long after the workload returns to its original level.

Discussion of Simulated Policies

I test several policies to see what might ameliorate or eliminate the decline in quality that follows a temporary imbalance between workload and resources. For ease of comparison, I explore these policies using the same test--a one-year increase in workload beginning in Year 3--although other shocks to the system, such as a temporary reduction in engineers or a temporary increase in the probability of error, would serve as well. The policies initially explored include introducing a prototype build during the 12 months preceding launch; dedicating engineers to testing; and altering the priorities for allocating resources.

Policy: Prototype Build During Year Before Launch

I first introduce a prototype build during the 12 months before launch (during Year 0). The build acts as a test for all parts prototyped to date. Like the builds that take place on the manufacturing line at the field site researched, the simulated builds take place over about a one-week period, make use of all prototyped parts for the model year about to launch, and consume no (design) engineering resources. In these simulations the prototype build takes place 34 weeks, or about eight months, before launch each year.

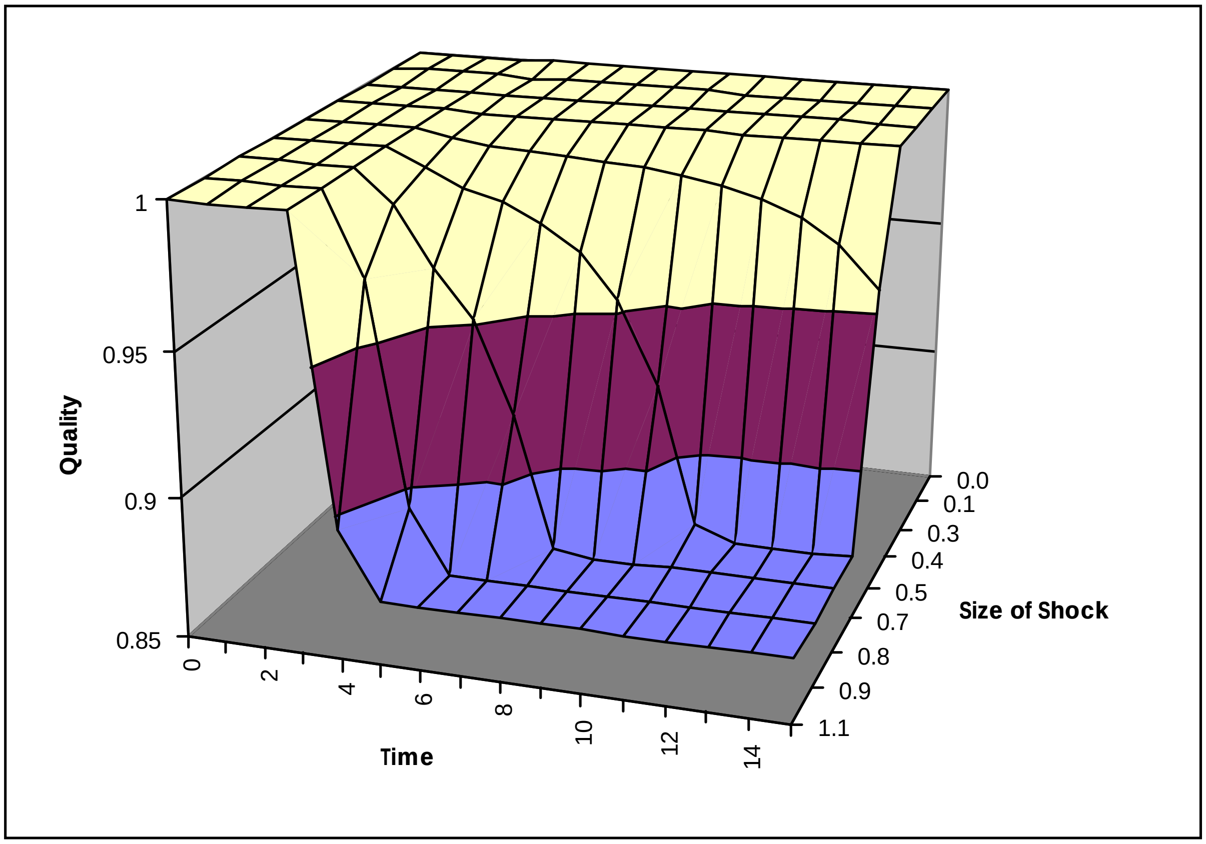


Figure 6: With Prototype Build 34 Weeks Before Launch;

Quality, With a Temporary Increase in Workload in Year 3,

"Textbook" Allocation with Resource-Constrained Testing

Comparing Figure 6 with Figure 4, we see that the build has a mildly stabilizing effect. A slightly larger shock to workload is required to tilt the system (the 100 percent surface is marginally broader). After the temporary increase, quality declines at a slightly slower rate (the slope to the lower left is more gradual) to a slightly higher equilibrium level (about 87 percent, rather than 86 percent).

Since the build acts as testing, it relieves engineers of testing responsibilities and allows more engineering-weeks to be devoted to other activities. First, when the high-content model year enters Year 0, nearly 300 parts remain to be prototyped, and all engineers are removed from other activities and allocated to Year 0 design. When the build occurs 34 weeks before launch, many problems are identified, but since some parts still need to be prototyped, no engineers immediately move to address the problems; design takes priority over problem-solving. Once Year 0 parts are prototyped, then all the engineers move to the activity of the next highest priority, problem-solving. Only after almost all problems are resolved do engineers move on to early phase prototyping.

In contrast, in the no-build scenario, no testing of Year 0 prototyped parts takes place until all parts are prototyped. As parts are tested, some engineers move to problem-solving. Testing and problem-solving proceed currently, since testing doesn't demand the entire engineering staff. Meanwhile even a few engineers are allocated to Year -1 design. As more Year 0 problems surface in testing, however, a majority of the engineering staff moves to problem-solving, leaving fewer to continue early phase prototyping.

When the high-content model year launches, most of its problems have been corrected, but little prototyping for the next model year has been accomplished during Year -1. Relatively, though, more early phase design is completed in the scenario in which the build takes place, and this leads to a less steep decline in quality when that model year hits the market than in the original scenario. Introducing a prototype build eight months before launch creates a more gradual decline in the fraction of the model year parts prototyped during the early phase and a more gradual decline in quality than the scenario without the prototype build during the transient phase--but at equilibrium in both scenarios, no early-phase prototyping takes place and quality is permanently lower. The build scenario's quality at the final equilibrium is slightly higher than the no-build arrangement, but the marginal improvement would not lead anyone to assert that introducing a prototype build could alleviate the dangers of the tilting dynamic in the model.

Timing of the Prototype Build

I explore the effects of build timing by creating a scenario in which the build takes place 16 weeks, or 4 months, before launch. In Figure 7, we see that the system is slightly less stable--a smaller shock erodes productive capability--though the final equilibrium of quality settles at 88 percent, just above both the no-build scenario and the scenario in which a build takes place 8 months before launch. Nevertheless, the tilting dynamic remains.

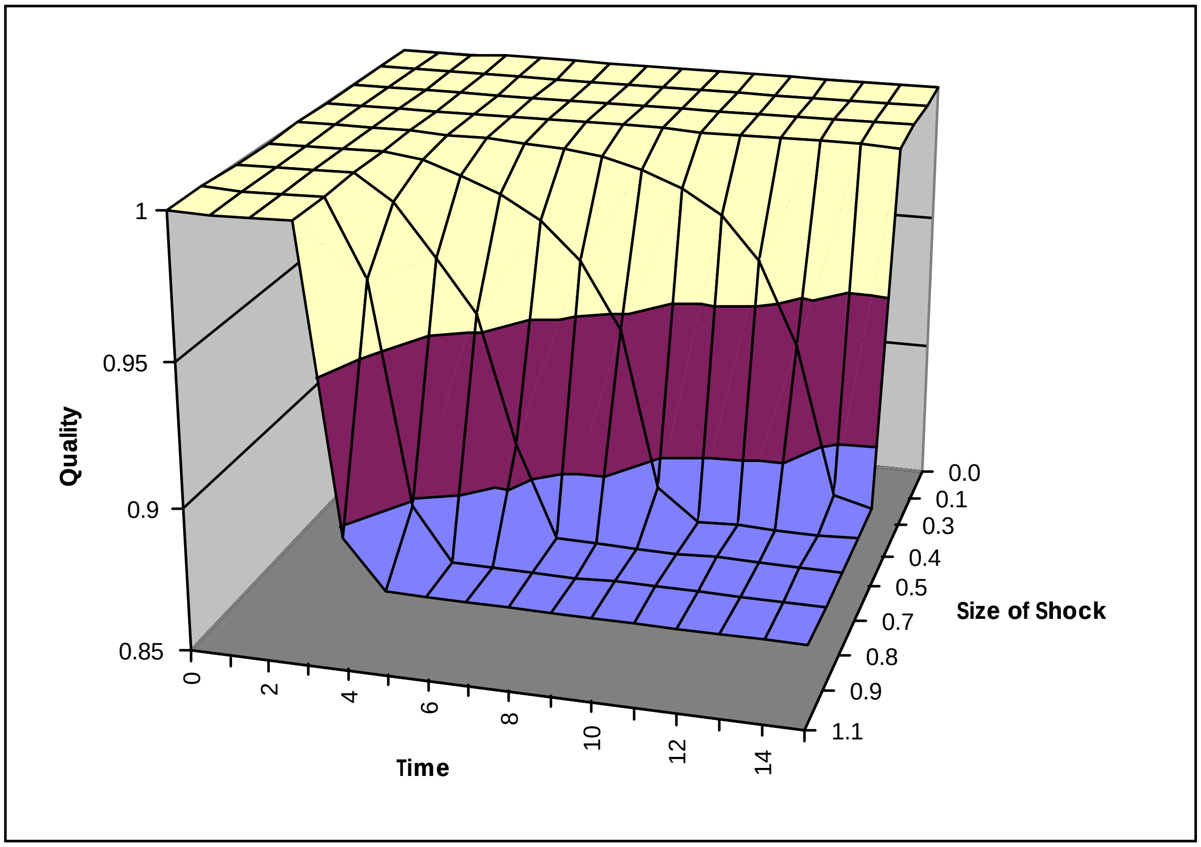


Figure 7: With Prototype Build 16 Weeks Before Launch;
Quality, With a Temporary Increase in Workload in Year 3,
"Textbook" Allocation with Resource-Constrained Testing

The slight difference in stability arises because here the prototype build is late enough that engineers complete Year 0 prototyping and begin testing and fixing the problems they find, so the organization loses to testing some of the engineering-weeks freed by the other build scenario. Then, when the build does occur in this scenario, it surfaces all remaining problems in the prototyped parts, and all engineers move to problem-solving for Year 0. By the time the high-content model year launches, most of the engineers have moved on to Year -1 prototyping, but still fewer parts are prototyped during the early phase than in either of the other simulated experiments examined thus far. This accounts for the slightly steeper face in the surface of Figure 7 showing the sharper decline in quality after the initial imbalance between work and resources.

In this model there is no distinction among types of defects; here we assume that prototype builds and testing performed by engineers are equally capable of identifying all problems. In reality, there are many different kinds of defects, and different forms of testing focus on identifying particular kinds of problems. Prototype builds are often intended to identify system-level problems and interface issues (Wheelwright and Clark 1992; Ulrich and Eppinger 1995), while testing by engineers may focus on component-level problems, material strength, or detailed fitment issues.
When projects fall behind schedule, however, builds that could surface system-level defects may in effect degenerate to fitment and component testing. This, in fact, may be the case at the field site studied, as indicated by comments such as "Builds [are] not surfacing all substantive issues--volumes and mix of [model] builds are low" and the assertion that engineers are "thinking the builds are just to give us test vehicles." Once expectations for prototype use begins to deteriorate, builds quickly become used for the wrong reasons, as engineers become more lax about having design truly completed before a prototype build takes place. "[We were] not ready for the builds," one engineer stated. Another said, "At worst, we field-test ...at the dealers."

Policy: Dedicating Resources to Testing

Next I examine the results of dedicating resources specifically to testing activities. Figure 8 shows the results of a series of simulations in which the temporary increase in workload is raised incrementally when one engineer is allocated only to testing. In this scenario, the engineer dedicated to testing is the only resource to perform testing (if other engineers are available, they work on other activities), and no matter how many engineers are needed for higher priority activities, this one engineer remains dedicated to testing.

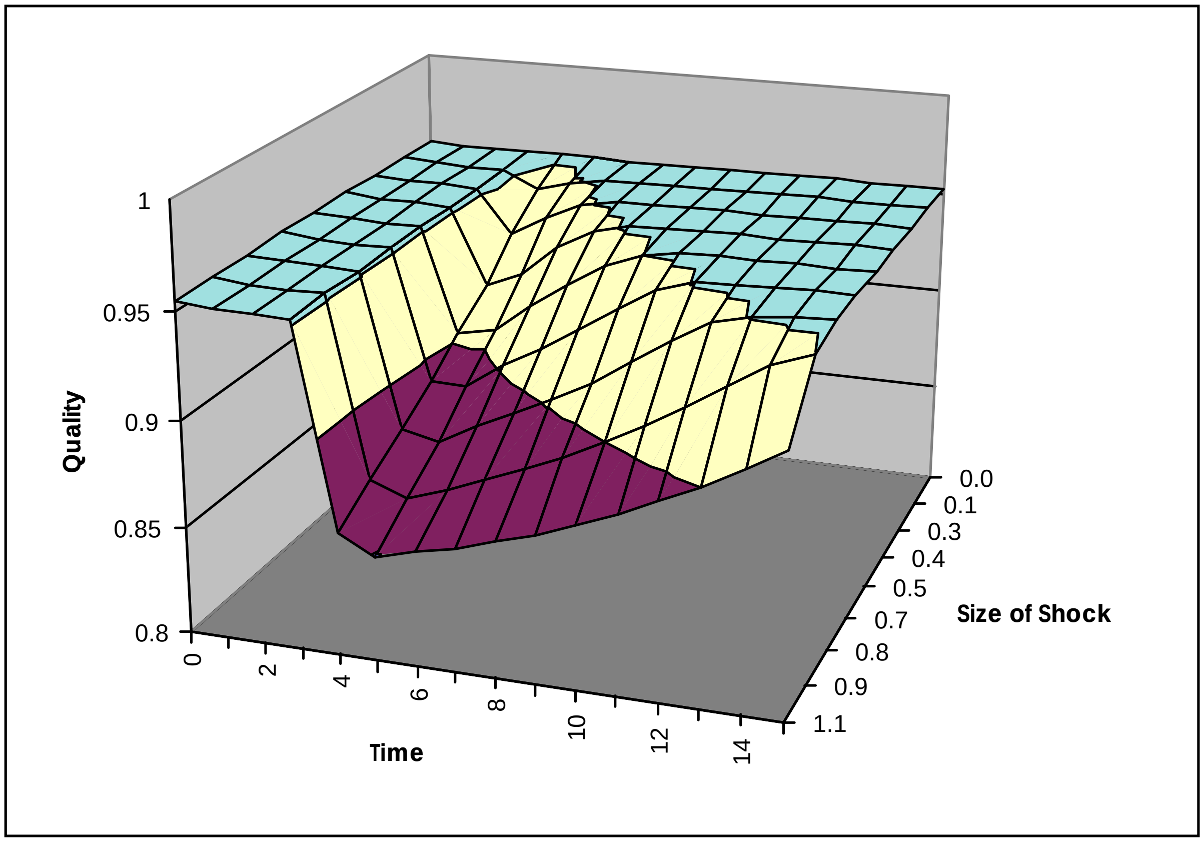


Figure 8: With One Engineer Dedicated to Testing;

Quality, With a Temporary Increase in Workload in Year 3,

"Textbook" Allocation with Resource-Constrained Testing

We note two characteristics in particular: the robustness and the overall lower quality. The system is robust, in the sense that it can recover from the temporary imbalance between workload and resources. The highest quality achievable in this scenario, however, is lower than in other scenarios (96 percent, instead of 99.9 percent). To understand how dedicating one engineer to testing brings about this result, it is helpful to compare one of the above simulation runs with situations in which no engineers are allocated to testing and two engineers are allocated to testing. Figure 9 shows a one-time 90 percent increase in parts-to-do for simulations in which zero, one, and two engineers dedicated to testing.

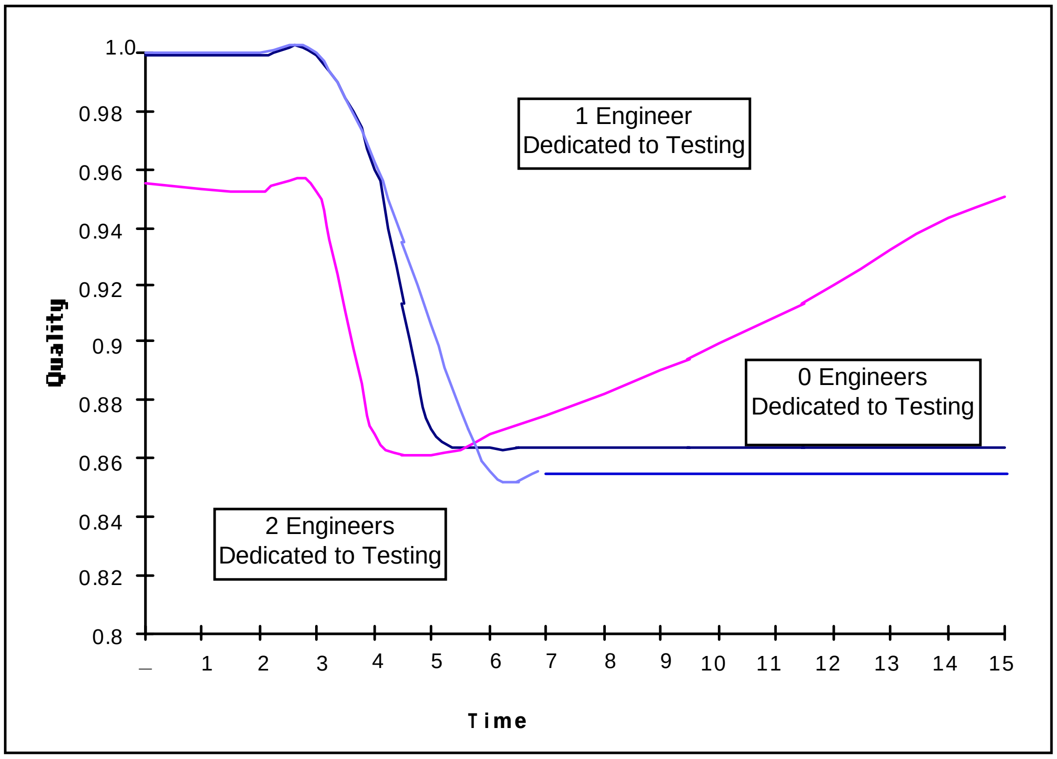


Figure 9: Comparing 0, 1, and 2 Engineers Dedicated to Testing, With a 90 Percent Temporary Increase in Workload

When two engineers are dedicated to testing, equilibrium quality is worse than when no engineers are dedicated to testing. These examples highlight the nature of the unstable equilibrium and the ways that any imbalance can work toward a permanent decline in quality. If no engineers are allocated to testing, when the surge in parts-to-do occurs, engineers design many but not all of the parts during the early phase. As the high-content model year enters Year 0, all engineers remain allocated to design until all 1450 parts are prototyped. Then all engineers move to testing Year 0 parts, and they quickly surface many problems in the prototyped parts (since some of the parts were prototyped during the current phase, the probability for error in those parts is higher than for those designed during the early phase). Then all engineers move to problem-solving in the current phase, but they are unable to correct all the errors before the high-content model year launches. At that time, the regular-size model year that has been in the early phase enters the current phase, with no parts prototyped (since all engineers were working on the parts about to launch). All engineers begin designing those parts, but with the higher probability of error of current phase prototyping, they make many mistakes, too many to correct before the model year hits the market. Then they face again a 700-part model year with no parts designed as of yet and 12 months until launch.

If one engineer is dedicated to testing, then when the surge in workload occurs fewer parts of the high-content model year are designed during the early phase, since there is one less engineer available for design activities. More parts must be designed during Year 0, when the error fraction is higher, and all remaining engineers are allocated to prototyping until all parts are designed. Since there is only one engineer allocated to testing activities, only some of the problems in the prototyped parts are identified; some of the engineers work on addressing those problems, but the rest move on to designing the next (regular-sized) model year during the early phase. When the high-content model year launches, not all of the problems in design have been identified, and not all the identified problems have been corrected. As engineers complete design of the remaining parts of the next regular-sized model year, some then move to addressing the problems identified by the lone tester, but the problems found do not consume all the engineers' time. So some begin prototyping the next model year's parts during the early phase. Each year following, more and more work is completed during the early phase, when the error fraction is lower, and quality gradually improves.

If two engineers are dedicated to testing, when the surge in workload occurs still fewer parts of the high-content model year are prototyped during Year -1, since two fewer engineers are available for design. This means that more parts must be prototyped during Year 0, when the error fraction is higher. Since two engineers are always available for testing, they quickly find all the errors in the parts designed. Again, in the weeks before launch all engineers (except the ones dedicated to testing) work on solving problems in Year 0 parts. When the high-content model year launches, the regular-size model year that has been in the early phase then enters Year 0 without any parts having been designed. Engineers cannot prototype all the parts and fix all the problems before the regular-size model year launches, and quality declines.

Hence we observe that dedicating some engineering time to testing helps in the situation above not so much because problems are identified sooner but because the policy works to constrain the number of problems identified and so limits the engineers allocated to rework. This effectively weakens the Tilting and Error Acceleration loops.

Policy: Altering the Priorities for Allocating Resources

I also varied the priorities for allocating engineering resources to the activities of design, testing, and rework. One of the more interesting results emerged from a scheme dubbed "Urgent First," since engineers are devoted first to solving problems and designing parts in Year 0, then to problem-solving and designing in Year -1, and testing in either phase receives the lowest priority. Specifically, the priorities in descending order are:

- Solving problems in Year 0 parts;
- Designing Year 0 parts;
- Solving problems in Year -1 parts.
- Designing Year -1 parts;
- Testing Year 0 parts;
- Testing Year -1 parts; and

While this clearly exaggerates the bias toward perfecting the projects closest to launch and the "fire-fighting" heroism that often accompanies working under a deadline, it serves a useful purpose, for the simulation shows a phenomenon significantly different from those observed before (see Figure 10).

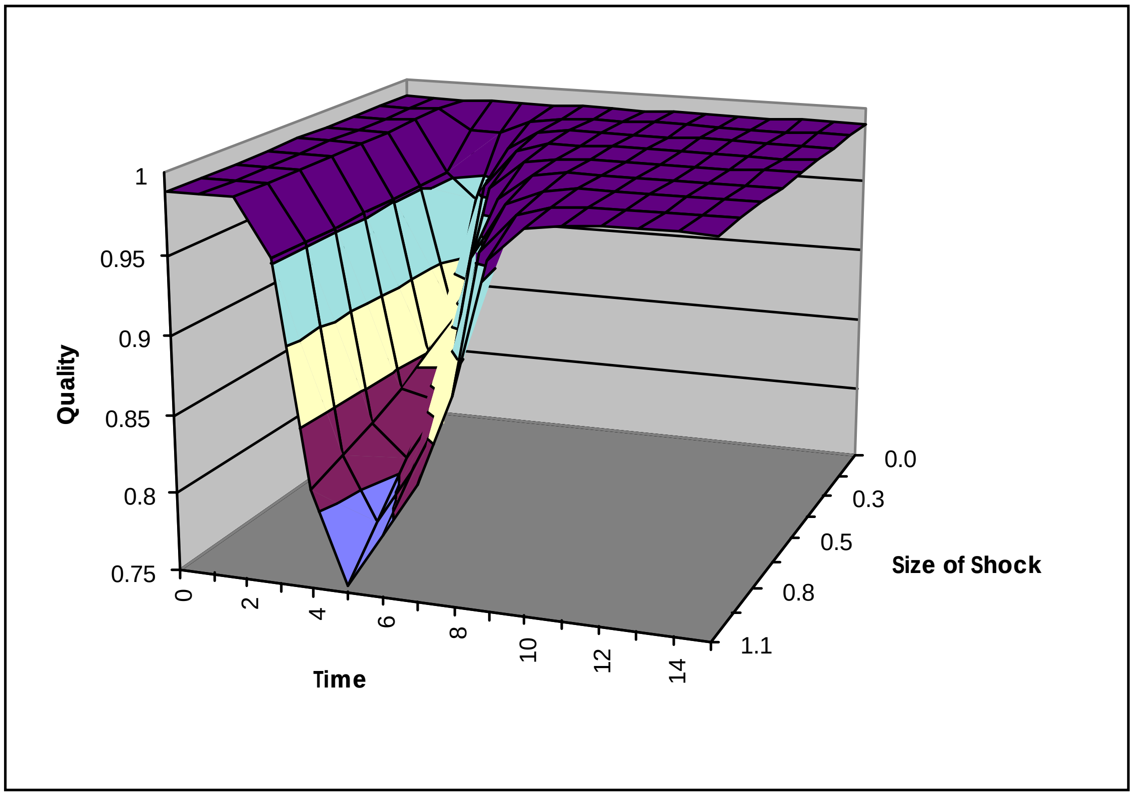


Figure 10: Quality, With a Temporary Increase in Workload in Year 3, "Urgent First" Allocation with Resource-Constrained Testing

As the high-content model year enters the market in Year 5, quality declines sharply, to 75 percent. The next model year, which contains the base workload of 700 parts, launches with quality almost as low (about 77 percent). Although quality continues to improve, it doesn't return to its original value of about 99 percent until five more model years (of 700 parts each) have entered the market.

As before, when the workload exceeds the engineering resources, the lowest-priority activities simply are not accomplished. In this case, the omitted activities are testing during each phase. Without testing, there are no identified problems to solve. If problem-solving does not absorb engineering resources, then engineers are allocated to other activities--such as designing parts during Year -1. The phase plots for the simulations under this allocation scheme are particularly revealing (see Figure 11). Clearly, no matter how large the temporary increase in workload may be, the system doggedly recovers.

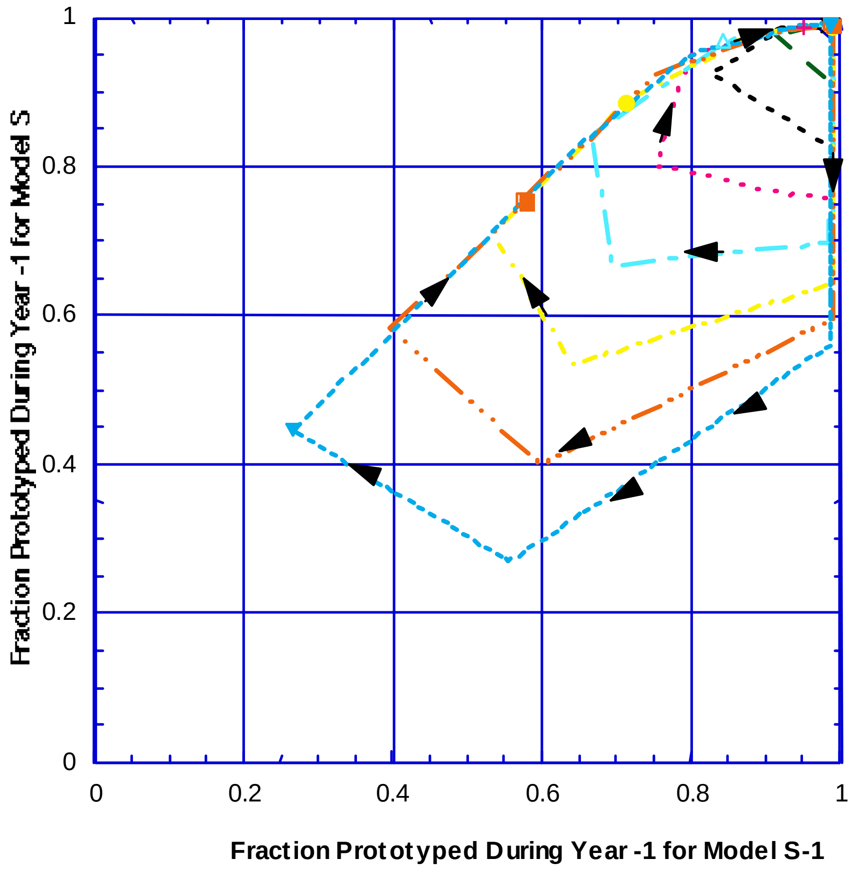


Figure 11: Phase Plot, "Urgent First" Allocation With Resource-Constrained Testing

The low priority of testing means that no problems are identified in Year 0 parts, and so fewer resources are allocated to completing the model year about to launch. In terms of the feedback loops identified in Figure 5, this weakens the Year 0 Completion Loop, making more resources available for the Year -1 work and so strengthening the Year -1 Completion Loop. This allows the reinforcing Tilting Loop to work in a virtuous direction, and in the years following the surge in workload more and more parts are designed during Year -1, thus gradually improving quality.

Based on these simulated experiments, I offer two assertions for consideration by people practicing product development. First, in an ongoing stream of new products, when projects are under development concurrently, it is possible that the effects of one project's demands--whether stemming from outsized scope or inadequate early planning or experimental process technologies--can exact their toll on other later projects whose scope, planning, and other aspects may be entirely adequate. Second, this simulated scenario leads to the counterintuitive proposal that additional testing does not necessarily improve quality--at least in the long-term. It also suggests some reinterpretation of the dynamics of the product development system. I have assumed that the goal of the product development system is to achieve high quality--that is, to complete all the work with as few errors as possible. The development phases described by the literature and practiced by companies are designed to serve that end. The "Urgent First" and "Textbook" allocations' results, however, reveal that the combination of modeled policies work together in such a way that might more accurately be described as reducing the identified (Year 0) problems to 0. Evidently there are at least two ways to achieve that goal: to fix all the problems found, or to find no problems.

In an organization where engineers are a scarce resource and people often feel pressure to choose carefully how to invest their time, it is possible that problem-solving could become esteemed as an activity more worthwhile than testing. Testing, after all, discovers problems; it brings bad news to everyone's attention. Addressing problems, however, allows engineers to bring good news to their colleagues and managers; and the closer the product is to launch, the more relieved everyone is that errors have been corrected. This can lead to rewards for "fire-fighting" that reinforce people's extraordinary efforts to improve projects close to launch rather than company-wide investment in identifying problems early. Although this model does not represent explicitly rewards within organization, data from case studies and my own experiences in organizations may permit some leeway in hypothesizing what might create and sustain certain priority schemes in an organization.

In the interviews conducted for the launch assessment, the "number one concern" expressed by engineers was low staffing. Insufficient resources posed difficulty for design engineering, manufacturing, and staff for support activities bridging engineering and manufacturing with new parts support, assembly drawings, and documentation. Others framed the problem as high workload, "overloading the product development factory," or, as one engineer said, "The harder I work, the more I have to do." High workload and low staffing point to the same imbalance between work-to-do and resources. Furthermore, many people in the company expressed some understanding of the consequences of such an imbalance. "There is time for only one priority," one team member said. "The only priority is the current model year," another offered. "What prevents people from raising issues?" queried a manufacturing engineer. "Short-term thinking" dominates, conceded one project team. Another team concluded that "the pervasive culture--we're a great fourth-quarter team--helps us and hurts us." A staff group saw only harm arising from the norm: "[The] culture helps us get through fire-fighting but hurts us overall."

Recall that one program manager said, "We don't know what we don't know until it's *real* late." These simulated scenarios and the interview data suggest that several dynamics (which reinforce each other) contribute to late identification of problems: testing as a low-priority activity among engineers; testing as an activity unrewarded by the organization; a low percentage of parts designed when the prototype builds take place; and rework from late testing sabotaging early work on subsequent product development efforts.

The Role of Prototype Builds in the Low-Priority-Testing Environment

To explore further the above assertions, I introduce a prototype build in the scenario with the "Urgent First" allocation scheme. Figures 11 and 12 depict scenarios in which a prototype build takes place 34 weeks and 16 weeks before launch, respectively.

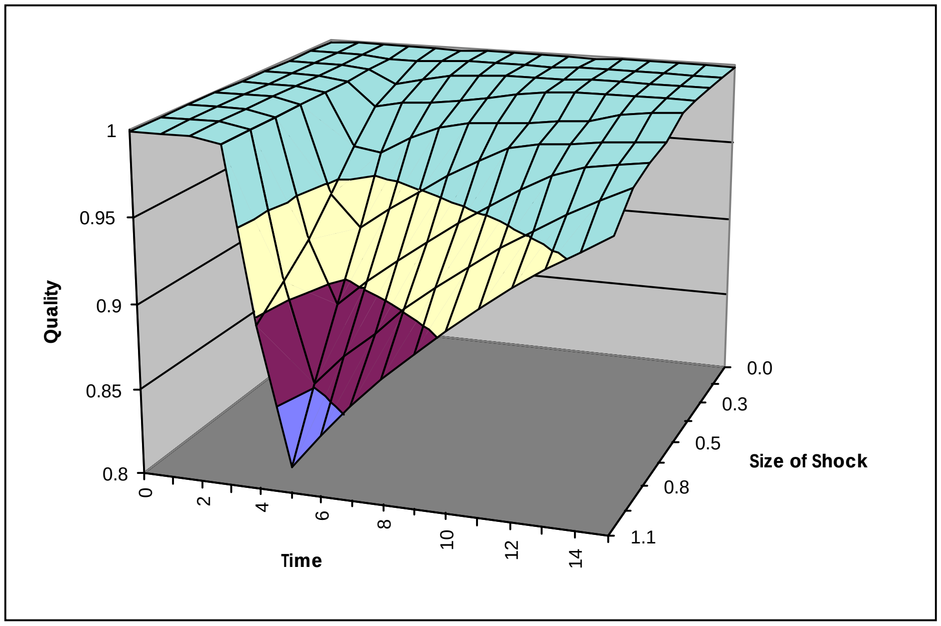


Figure 11: With Prototype Build 34 Weeks Before Launch, Quality, With a Temporary Increase in Workload in Year 3, "Urgent First" Allocation with Resource-Constrained Testing

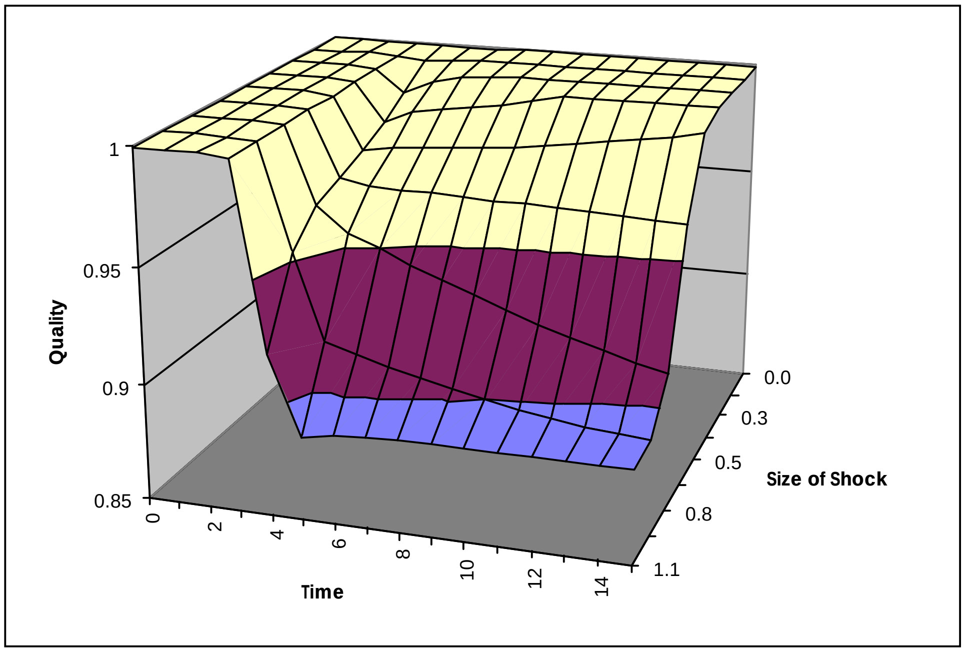


Figure 12: With Prototype Build 16 Weeks Before Launch, Quality, With a Temporary Increase in Workload in Year 3, "Urgent First" Allocation with Resource-Constrained Testing

At first glance, it may seem surprising that the earlier build does not send the system into a permanently lower equilibrium. This does not transpire because, in the model year following the temporary increase in work, relatively few parts are prototyped when the build occurs 34 weeks before launch. So even though engineers move to problem-solving activities for Year 0, there are so few problems found in the few parts designed that they soon move back to Year 0 design. When they have designed the remaining parts, (since testing is lowest priority) they move on to designing parts for the next model year during its early phase. In this way, gradually the system recovers because each year more and more work is completed during Year -1. Even when quality is at its lowest, it remains above 80 percent, a 5 percent improvement over the comparable no-build scenario.

In the scenario in which the build occurs 16 weeks before launch, quality's lowest point is yet higher--above 88 percent. But the late prototype build brings back the tilting dynamic. Here, in the year following the high-content launch, engineers have completed prototyping nearly all of the Year 0 parts when the build takes place (about four months before launch). Therefore the build surfaces many more problems. All engineers are allocated to solving Year 0 problems, and no early phase work for the next model year takes place. Hence we see the permanent decline in quality in this scenario. The system relies on prototype builds to surface all problems; when the build takes place, depending on how many parts have been through the first iteration of design, a proportional number of problems will be identified, which will then demand resources that otherwise might be devoted to preventing problems in the next model year.

Of course, one would never recommend that a company abandon testing its new products under development. The above findings simply highlight that in multi-project environments we need policies that prevent rework in one project from undermining work on other projects. In other words, we need to identify policies that can help the organization withstand pressures that lead to tilting. Since it is unreasonable to conclude that long-term quality can improve if testing activities are simply omitted, the next steps explore alternative policies for dealing with identified problems under the original "Textbook" Allocation of engineering resources:

postponing incomplete work; correcting only a fraction of the defects; and canceling incomplete work.

Policy: Postponing Incomplete Work

In the interviews conducted at the field site, I found that occasionally projects were moved from one scheduled launch date to the following model year. Often this decision emerged from one of the project review meetings taking place between four and eight months before the originally scheduled launch date. Several project teams complained that these postponement decisions were made too close to the scheduled launch, and others pointed out that postponing efforts did not always solve the project's problems and even sometimes exacerbated them. "Delaying projects doesn't solve the resource shortage," asserted one group. In one case, the decision to postpone a project was made after engineering resources for the *next* model year had already been allocated to projects, and the team whose problematic project was delayed again faced continuing problems with no resources available to solve them.

Convinced that it is worthwhile to simulate a policy of postponing incomplete work, in the model I introduce a checkpoint during Year 0 (whose timing can be varied). At the checkpoint, all parts that have not yet been designed are removed from the model year in the current phase and added to the model year in the early phase. For simplicity's sake, only the parts not yet designed are removed; parts that have been prototyped but that contain errors are not postponed. Because we assume that postponing parts incurs no cost to productivity and because we assume no interdependence among parts, the simulated results of postponing should appear much rosier than those a company adopting this policy might actually experience.

Figure 13 shows the results of postponing parts that have not been designed six months before launch. Instead of focusing on quality, however, in this three-dimensional graph we look at performance--a measure representing the quality of the parts launched times the percentage of the model year that actually enters the market.

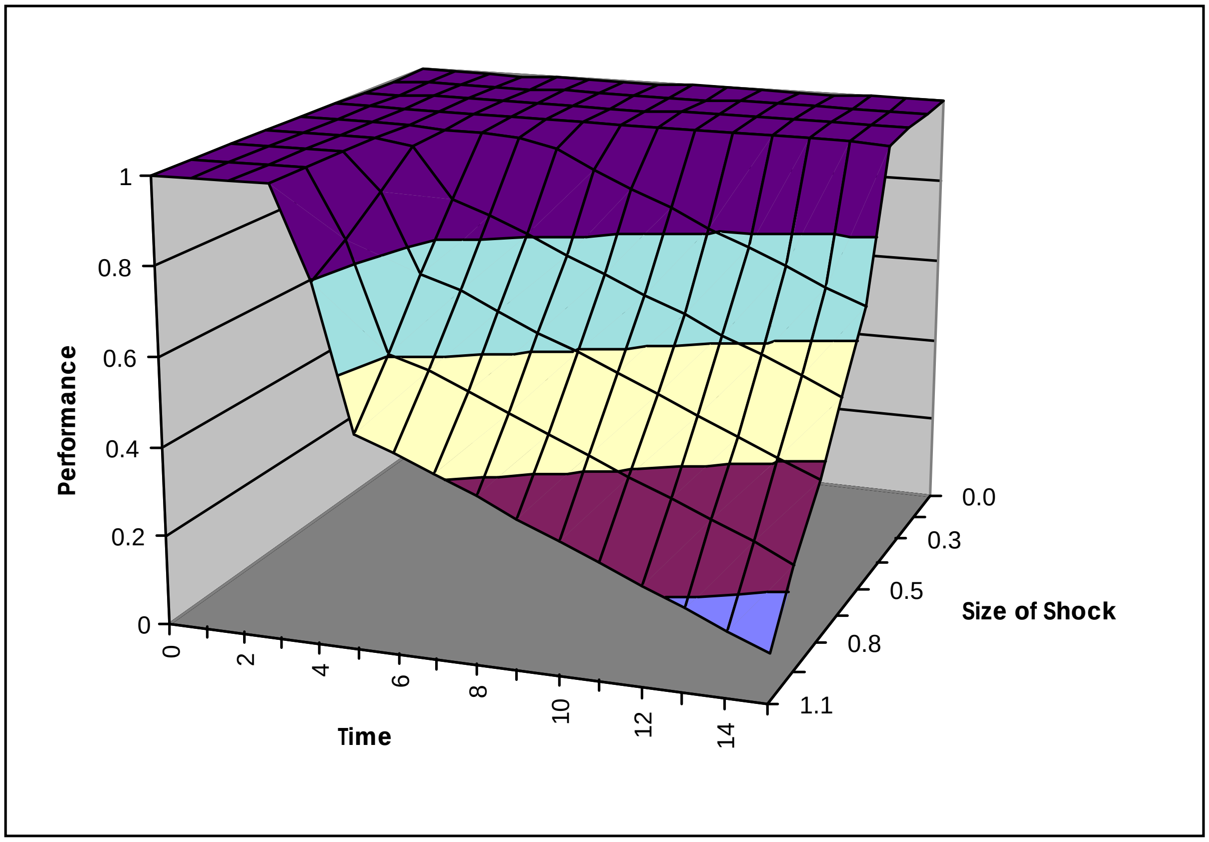


Figure 13: When Undesigned Parts are Postponed 6 Months Before Launch, Performance, With a Temporary Increase in Workload in Year 3, "Textbook" Allocation with Resource-Constrained Testing

Beginning in Year 5, the year that the high-content model year launches, parts are postponed at the checkpoint if the increase in workload exceeds about 40 percent. The greater the shock in Year 3, the greater the number of parts postponed. These postponed parts are added to the next model year's work-to-do. At the following year's checkpoint, even more parts have yet to be designed, so more parts are postponed, and the fraction of the intended model year (the base workload of 700 parts plus the postponed parts from the previous year) declines. Each year following, more and more parts are delayed and performance (as measured here) degrades further as the workload, with years' accumulation of postponed parts, grows more and more unwieldy. Always postponing and never canceling incomplete parts exaggerates the deterioration in performance; if we ignore the backlog of postponed parts, about 90 percent of the base workload of 700 parts launches each year.

If we look only at quality (see Figure 14), we see that it deteriorates only slightly.

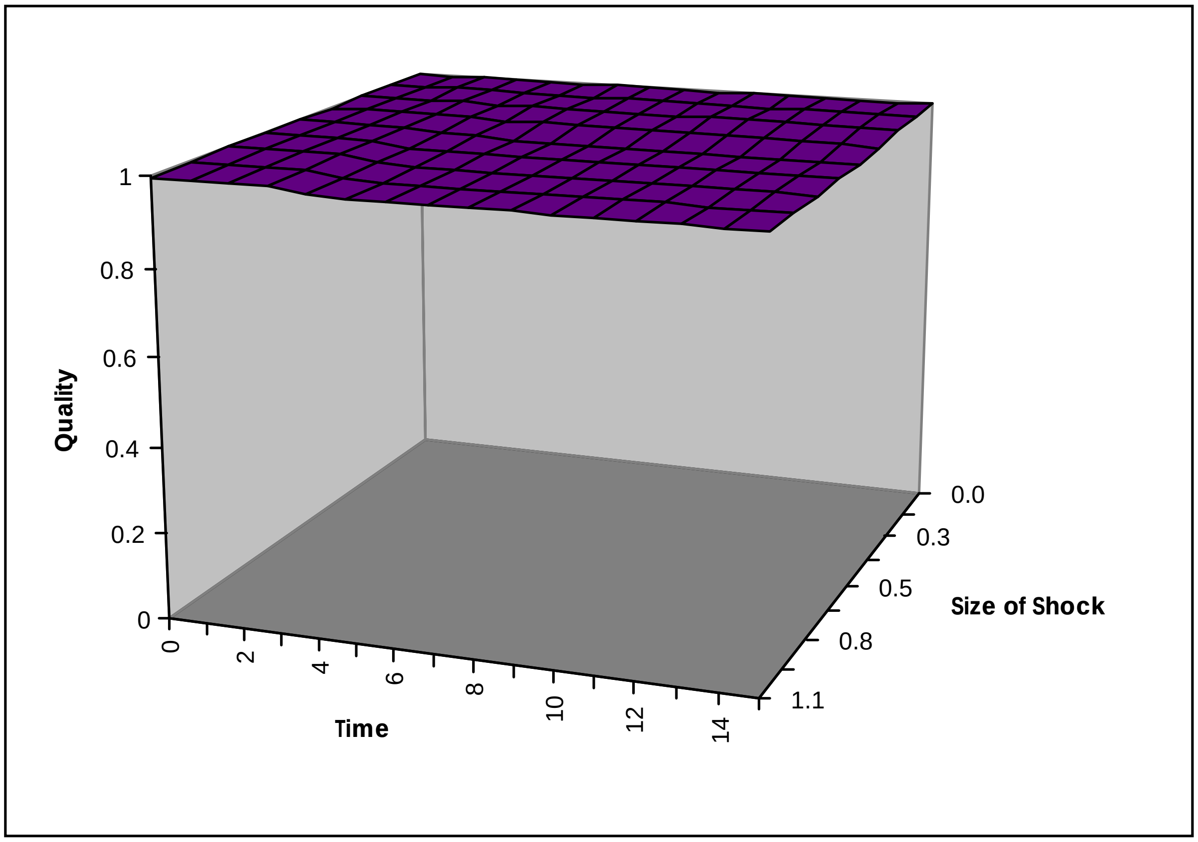


Figure 14: When Undesigned Parts are Postponed 6 Months Before Launch, Quality, With a Temporary Increase in Workload in Year 3, "Textbook" Allocation with Resource-Constrained Testing

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

I conclude that, even though the simulated experiments overestimate the benefits of a postponement policy, the outcome is not favorable. Postponing a portion of the model year does not help the engineers "catch up" after a temporary increase in workload. Furthermore, the engineers never launch the number of parts after the shock as before the shock. Even though the policy maintains high quality in the parts launched, postponing incomplete work does not allow the product development system to recover fully after the temporary imbalance between resources and work.

Policy: Fixing a Fraction of the Errors

Since the model does not distinguish among types of defects in parts under development, it may be appropriate to simulate a policy that allocates engineers to correct only the "important" errors rather than all the errors identified through testing. To simulate this policy, I generously assume that the important errors are instantly recognizable and clearly communicated to all. If the policy does not yield improved simulations under these favorable assumptions, then one can conclude that it would not help in reality, since it is not always easy to distinguish more significant defects from the less significant ones and since communicating across projects and functions is often challenging. If the policy works, then we can expect to see in the model a more severe drop in quality during the launch of the high-content model year and the year following than observed in Figure 3 above (since a majority of the errors remain uncorrected) but a return to high quality thereafter.

Figure 15 shows the result of this policy simulation. We see the same tilting phenomenon as in Figure 3, though the lower equilibrium of quality is higher (89 percent rather than 86 percent as observed in Figure 3) and on the rise, though it is nowhere near its original level of 99.9 percent even 10 years after the single outsized model year is introduced to the market. Also, while in the original "Textbook" Allocation scenario quality declines to 86 percents and remains at that level, the policy of fixing only 20 percent of the errors makes the initial decline in quality dip to about 81 percent.

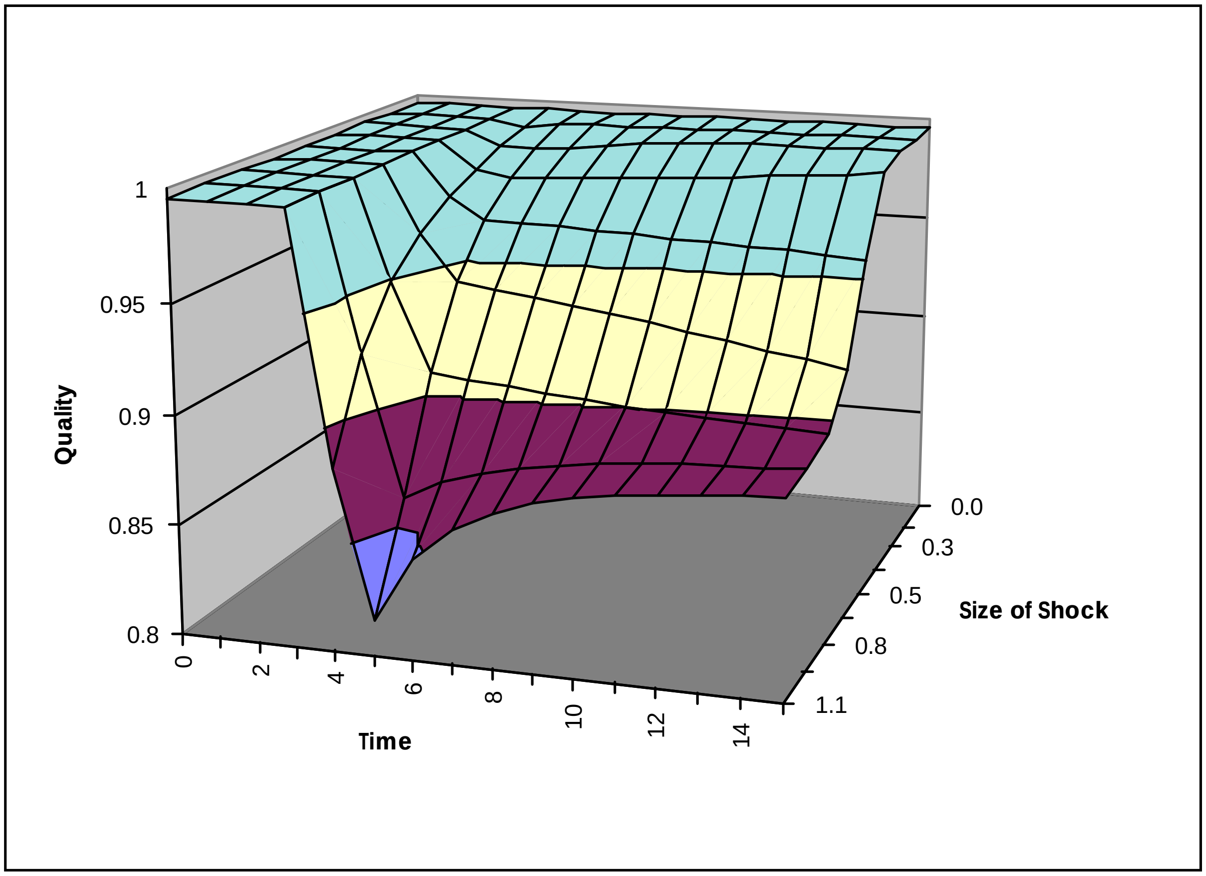


Figure 15: Fixing 20 Percent of the Problems Identified--
Quality, With a Temporary Increase in Workload in Year 3,
"Textbook" Allocation with Resource-Constrained Testing

Even though in this scenario the desired number of engineers to allocate to problem-solving in Year 0 is fewer than 50, far less than the 200 desired in the base scenario, 40-plus engineers still comprise a near majority of the engineering staff, and less than one-eighth of the possible engineering weeks are spent on early phase design. Thus, after the shock in workload, engineers consistently prototype only between 10 and 15 percent of parts during the early phase. Since so great a percentage of parts must be completed during the current phase when the error fraction is higher, engineers find many problems to correct during the 12 months before launch, and few of the next model year's parts are prototyped during Year -1.

Policy: Canceling Incomplete Work

I next simulate a policy of canceling incomplete work. Similar to the postponement policy, the canceling policy relies on a checkpoint each year, at which the number of parts in Year 0 not yet prototyped are assessed. In this scenario, however, rather than moving the undesigned parts to the early phase of the following model year, the policy simply eliminates them. Figure 16 shows the series of simulations resulting from a checkpoint each year 6 months before launch. Again, I use performance, rather than quality, as a proxy for the health of the product development system.

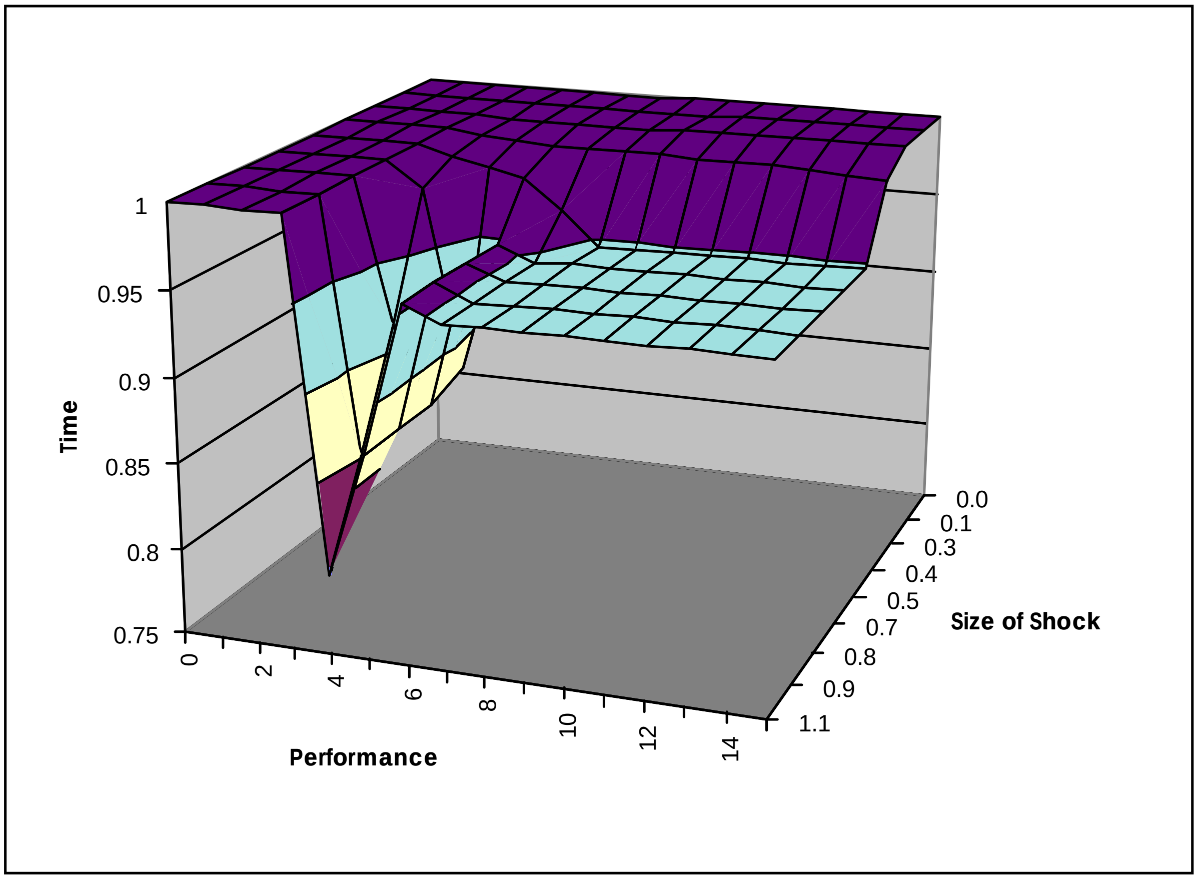


Figure 16: When Undesigned Parts are Canceled 6 Months Before Launch,

Performance, With a Temporary Increase in Workload in Year 3,
"Textbook" Allocation with Resource-Constrained Testing

For the largest shock in workload, the year that the high-content model year launches, nearly 300 parts are canceled, and performance drops accordingly, to below 80 percent. Although the system begins to recover immediately, for increases in workload exceeding about 35 percent of the usual number of parts, engineers are never again able to perform as much work during the early phase as they were before the temporary increase in work. For the larger pulses in workload, nearly 30 parts are canceled consistently each year because they have not yet been designed by the time the checkpoint occurs, six months before launch. Even though the temporary imbalance between workload and resources tilts the product development system into a lower-performance regime, the stable equilibrium is considerably higher than in other scenarios (95 percent, rather than below 90 percent).

Again, when we look only at the quality of the work launched (ignoring the percentage completed of the intended model year), we see that quality deteriorates only slightly.

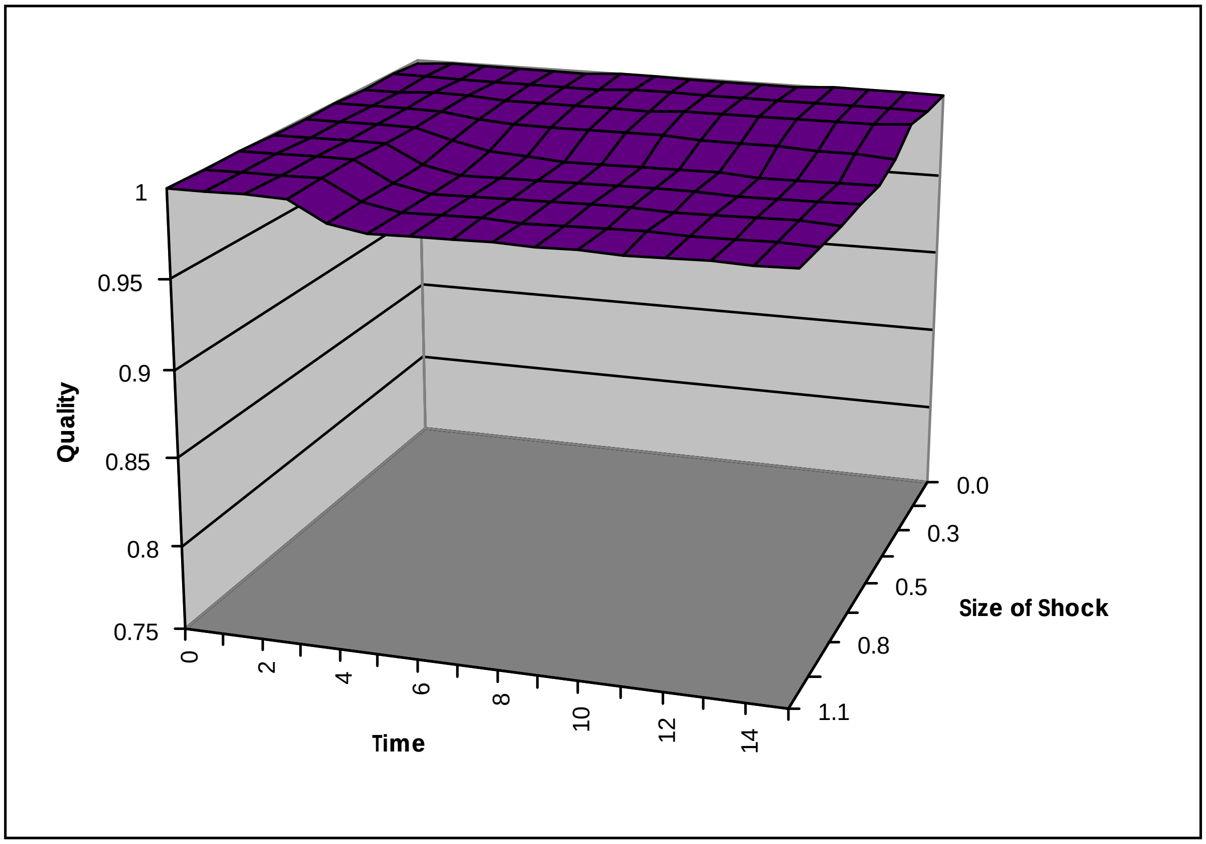


Figure 17: When Undesigned Parts are Canceled 6 Months Before Launch, Quality, With a Temporary Increase in Workload in Year 3, "Textbook" Allocation with Resource-Constrained Testing

Hypothesizing that an earlier checkpoint might allow the product development system to recover completely from a temporary imbalance between workload and resources, I move the checkpoint to 12 months before launch. If, when work from Year -1 is indexed to Year 0, there are parts that were not designed during the early phase, those parts are removed from the model year scheduled to launch in 12 months. Figures 18 and 19 show the results for performance (quality times the percent of intended model year launched) and quality (the percent of model year parts completed correctly by launch).

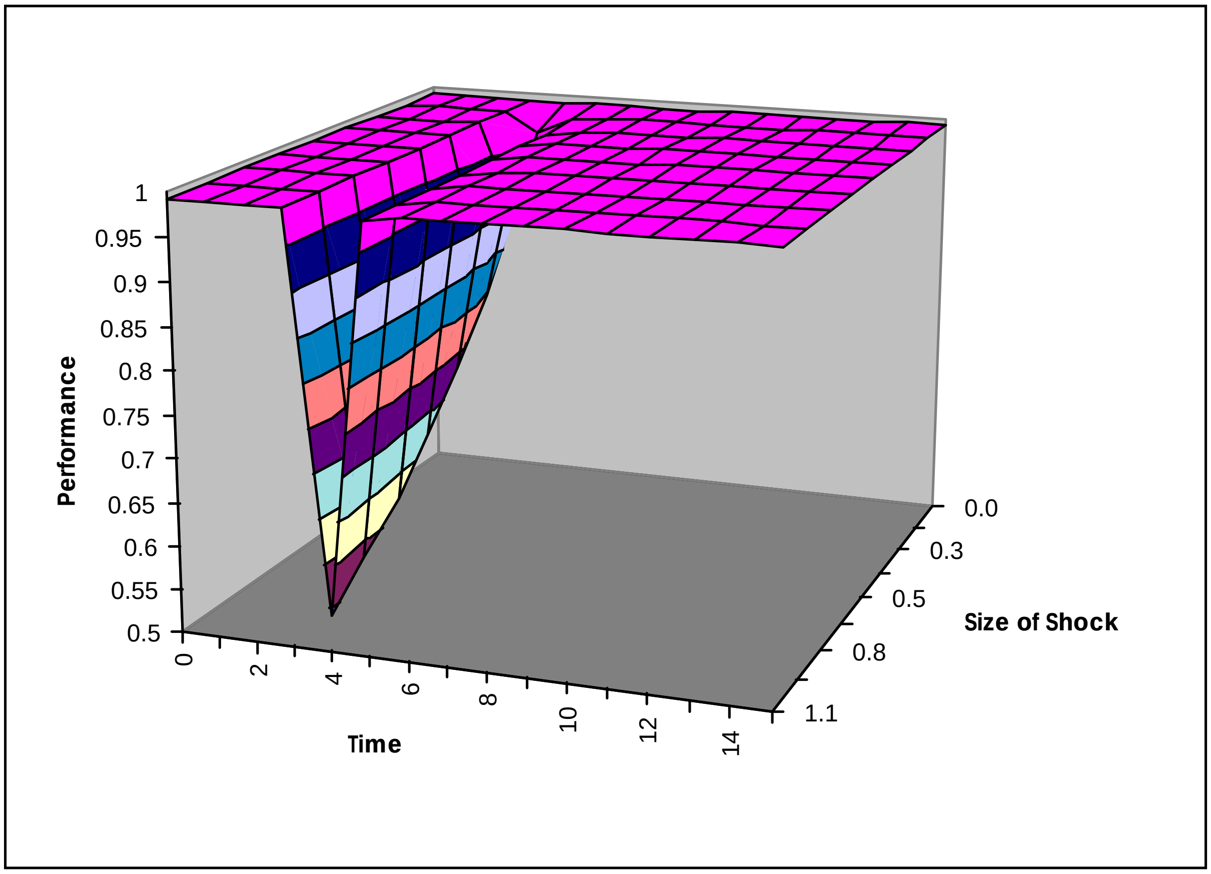


Figure 18: When Undesigned Parts are Canceled 12 Months Before Launch, Performance, With a Temporary Increase in Workload in Year 3, "Textbook" Allocation with Resource-Constrained Testing

Figure 18 depicts a significant dip in performance for even small temporary increases in workload. As Figure 19 shows, the decline in performance does not result from low quality (uncorrected defects), since quality remains at 99.9 percent, but rather from a reduction in the model year scope. But after the imbalance, the recovery is immediate and enduring.

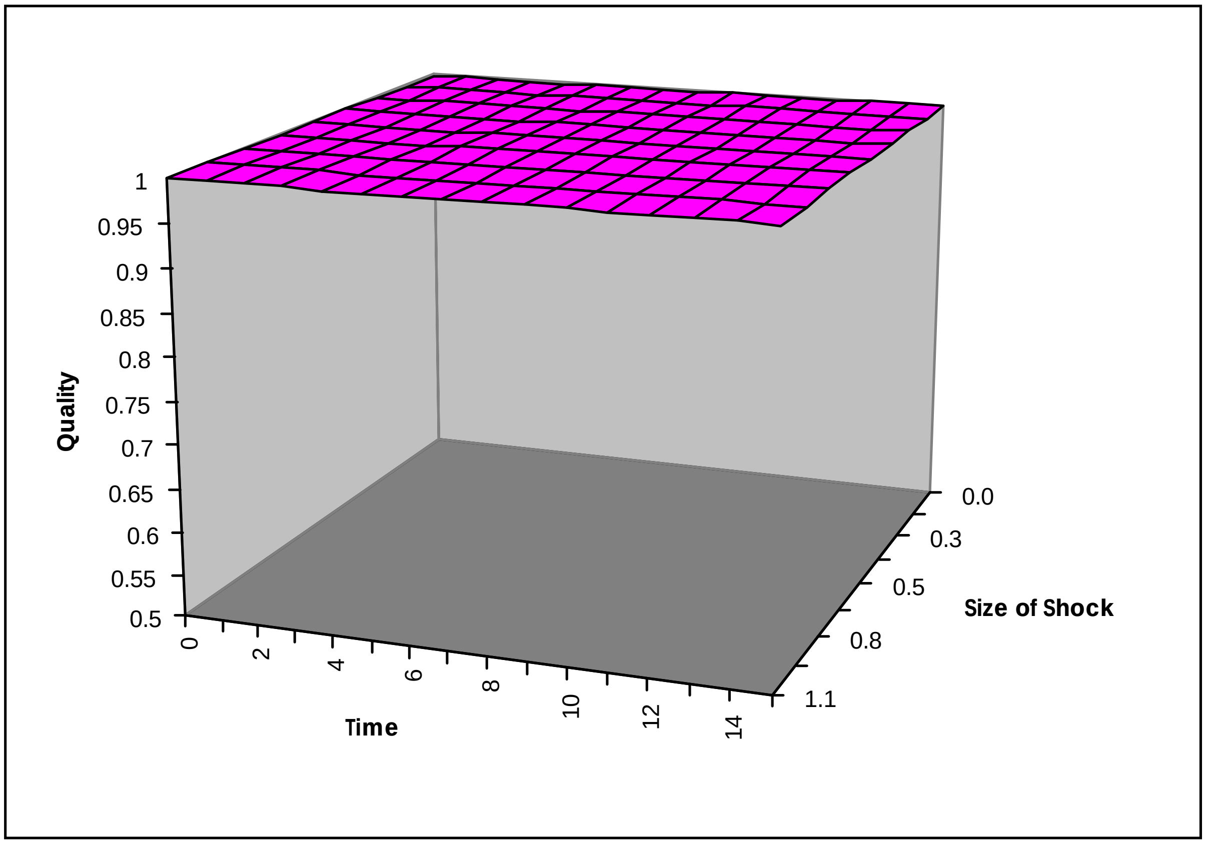


Figure 19: When Undesigned Parts are Canceled 12 Months Before Launch,
Quality, With a Temporary Increase in Workload in Year 3,
"Textbook" Allocation with Resource-Constrained Testing

I then run simulations for earlier and later checkpoints to identify how checkpoint timing affects quality. Figure 20, rather than depicting simulations over time, shows only the steady-state equilibria for a variety of checkpoint times when the temporary shock is varied. Any checkpoint ensures an equilibrium value of quality as good as or better than that of the base scenario. Clearly, however, the timing of the checkpoint and decision-making makes a significant difference in the long-term outcome: The earlier checkpoints assure long-term performance better than later checkpoints, but a checkpoint can be too early. In this model, if the decision-making checkpoint occurs more than about 45 weeks before launch, quality begins to decline below the observed maximum of 99.9 percent.

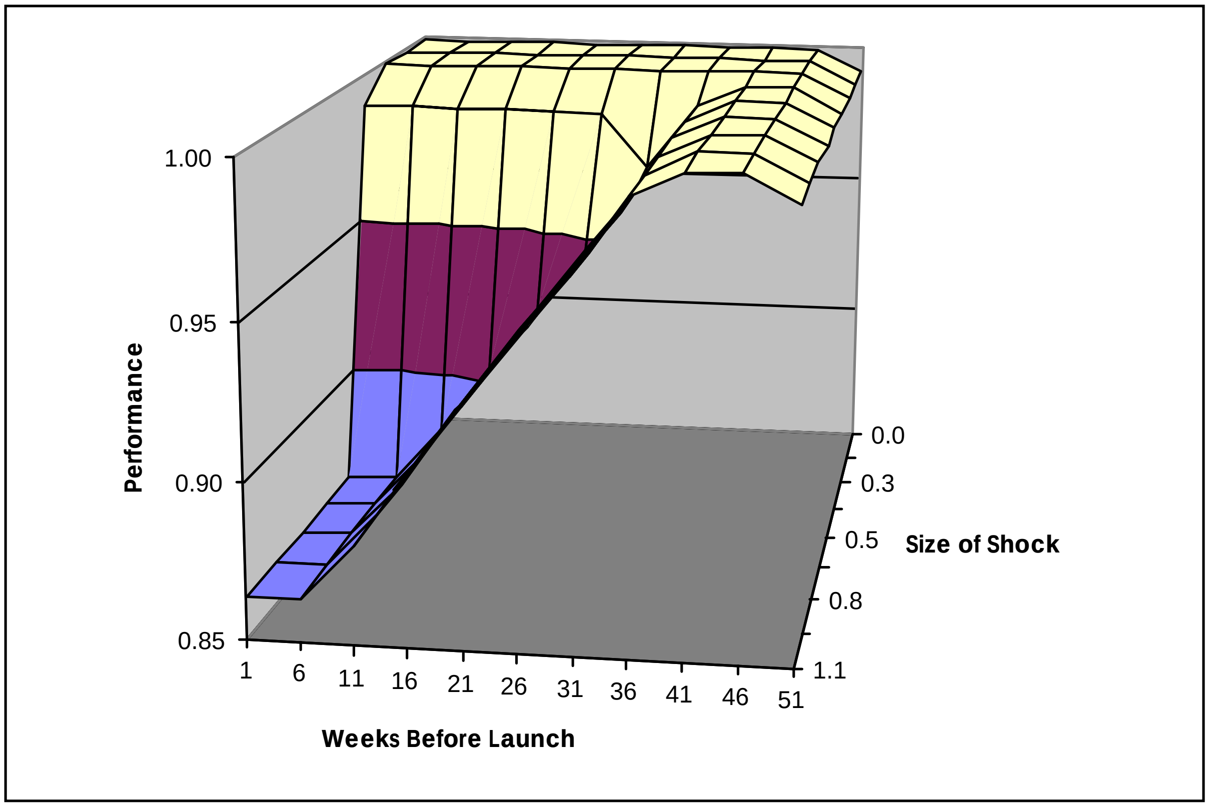


Figure 20: Steady-State Equilibria for Performance,
When the Checkpoint Timing is Varied

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

Discussion

From this series of simulation exercises, I conclude that, in an environment of overlapping multiple projects, the same number of resources can produce the same number of new products with a wide range of quality, depending on whether the workload and resources have been consistently balanced in the past. I emphasize that the variance in quality depends not on subtleties of the process methodology nor on whether engineers "care" or "try hard" in their work. It depends only on the assumption that design completed earlier in the product development process has a lower probability of error and that projects nearer to launch take precedence over projects whose market introduction is later.

The model constructed for this exploration makes critical simplifying assumptions: parts are assumed to be independent; defects are assumed to be corrected by revising only one part; engineers are easily transferred among the activities of design, testing, and problem-solving for all parts; and there is no cost in reallocating people from one activity or model year to another. While these assumptions are patently unrealistic, they err on the side of generosity. Unfavorable results observed in this model's simulations would simply be strengthened by incorporating dynamics resulting from interdependence among parts, uneven utilization of a specialized workforce, and costs in shifting people from one project to another.

In *Revolutionizing New Product Development* (1992), Wheelwright and Clark clearly state that many companies participating in their research unwittingly overload the product development system by undertaking too many projects. In one case, they estimated that the work exceeded the resources available by as much as 200 to 300 percent. While "stretch goals" have validity in some circumstances, few managers would choose systematically to undermine the capability of their product development organizations by inundating them with a workload that *ensured* low quality output.

Two points are noteworthy here: First, the turning point at which a reasonable workload becomes "overload" is not obvious to either managers or engineers at the time. Even if some suspect imbalance between work and resources, they cannot unequivocally demonstrate it at the time, because the worst deterioration in quality manifests only some time after the imbalance occurs. Second, an unusual amount of rework in the "right" number of projects can also overload the product development system. Anything that increases the likelihood of error during development--such as introducing a new technology, moving to a new facility, or even reorganizing departments--can effectively "shock" the product development system, although workload ostensibly remains constant. This is why iterations of testing and rework can play a particularly insidious role, especially if the organization plans its work as if development were a linear, sequential process. Although Wheelwright and Clark speak of the dangers (and likelihood) of overloading the development system with too many projects underway, they do not explicitly connect "additional late cycles" of product development to overloading the system. This paper points out: Testing and problem-solving can overload the development system as much as too many projects can.

Dynamically balancing work to resources can maintain the viability of long-term productive capacity. Perhaps many managers believe that they already do this; managers at the field site studied might argue that they frequently reallocate engineers to projects needing extra help in order to meet their scheduled launch. But balancing work to resources includes two critical components: first, it must assess the *total* amount of work-in-process; second, it must balance work to resources by *reducing work* to accommodate the resources available.

The total work-in-process must account for *all* projects under development (not just the one nearest launch) and account for the *actual state* of those projects in terms of percent of testing completed and percent of identified problems resolved. Managers cannot balance what they are unaware of--if testing is not complete at the component, subsystem, and system levels, then managers cannot know the workload that will emerge as problems are identified. Decision-makers basing workload estimates only on parts per model year or on the number of projects underway, without considering parts to be reworked, are more likely to overload the product development system. Therefore, as testing progresses, it makes sense to update dynamically the quantity of work-in-process based on the problems identified and as yet unresolved.

In the model, the only way unequivocally to restore balance is to eliminate identified problems or eliminate work-to-do. This explains the counterintuitive result that no testing improves long-term quality. In reality, eliminating identified problems by refusing to test during development poses a poor strategy for improving long-term quality. On the other hand, canceling incomplete work in a time-frame well in advance of launch can restore balance to the product development system while ensuring short-term and long-term quality. Canceling projects may seem an extreme move. Smith and Reinertsen (1998), however, relate an anecdote in which a company reduced its product development project list from nearly 50 projects to half a dozen. Managers have not missed the aborted projects, the authors assert, because "better opportunities have arisen since, and they were in a position to exploit them." Additionally, when testing does identify problems in development work that is near market introduction, carefully choosing which problems to solve while looking at entire portfolio of work underway can ensure that resources are available to do the critical early work on subsequent projects and guarantee continued high quality.

Conclusion

The model presented here offers a structural reason why project problems may consistently surface late in the product development cycle. In a company that undertakes multiple projects and allocates engineering resources among work in various phases of development, biases for solving the biggest problems for the nearest deadlines can effectively undermine early phase development efforts. If testing is a low-priority activity among engineers, or if a low percentage of parts are designed when prototype builds take place, then rework from late-surfacing projects may claim an increasing percentage of the engineering staff. The work produced under such conditions is often of a lower quality than that possible, and the employees producing it can become physically and emotionally fatigued from "fire-fighting." A permanent imbalance between workload and staff is not required to establish a persistent low-quality regime of output. A one-time temporary shock to the system can create permanently undesirable effects. With the model, I simulated a variety of policies in an effort to identify measures practitioners can take to alleviate pressures that lead to tilting into fire-fighting mode. The only policy that renders the product development organization robust to imbalances between workload and resources is that of canceling incomplete work well before its scheduled launch date. This results in a temporary reduction in the market introduction's size or scope but ensures consistently high quality in the product and a full return to high productive capability in the organization.

Bibliography

--------- (1998). "Launch Assessment: 1999 Model Year." Company report (name withheld in accordance with researchers' confidentiality agreement).

---------- (1999). "How the Rubber Met the Road: Product Development Process Assessment, Model Year 2000." Company report (name withheld in accordance with researchers' confidentiality agreement).

Forrester, J.W. (1961). *Industrial Dynamics.* Portland, Oregon: Productivity Press.

Griffin, A. (1997). "PDMA Research on New Product Development Practices: Updating Trends and Benchmarking Best Practices," *Journal for Product Innovation Management,* No. 14, pp. 429 - 458.

Krishnan, V., S.D. Eppinger, D.E. Whitney (1995). "A Model-Based Framework to Overlap Product Development Activities," working paper #3635, MIT.

Meyer, M.H., P. Tertzakian, and J.M. Utterback (1997). "Metrics for Managing Research and Development in the Context of the Product Family," *Management Science*, Vol. 43, No. 1, pp. 88 - 111.

Repenning, N. (1999). "The Transition Problem in Product Development, " working paper.

Smith, P.G. and D.G. Reinertsen (1998). *Developing Products in Half the Time.* New York: John Wiley & Sons, Inc.

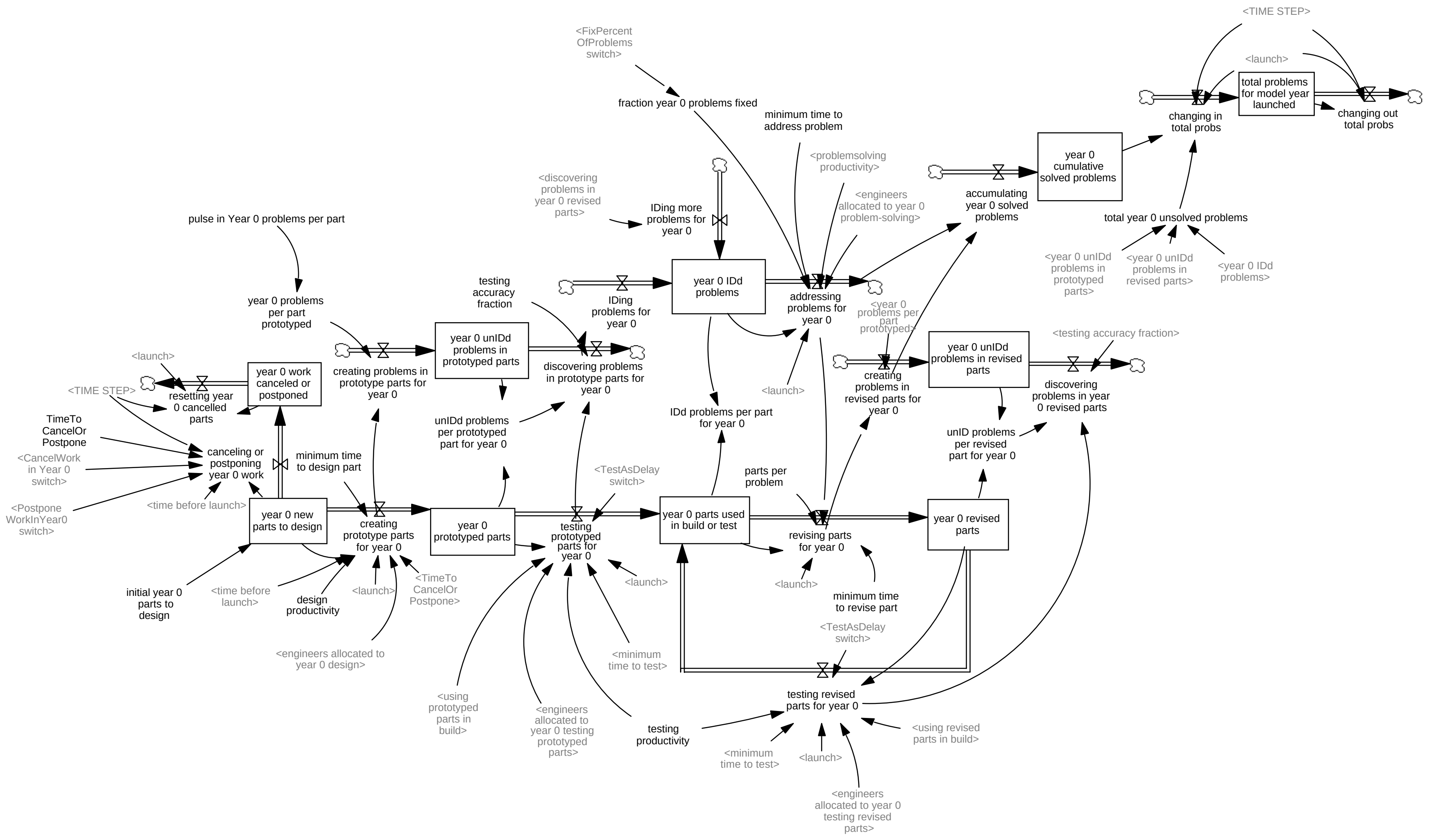
Sterman, J.D. (1989). "Modeling Managerial Behavior: Misperceptions of Feedback in a Dynamic Decision Making Experiment," *Management Science*, Vol. 35, No. 3, pp. 321-339.

Sterman, J.D., N.P. Repenning, and F. Kofman (1997). "Unanticipated Side Effects of Successful Quality Programs: Exploring a Paradox of Organizational Improvement," *Management Science*. Vol. 43, No. 4, pp. 503-521.

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

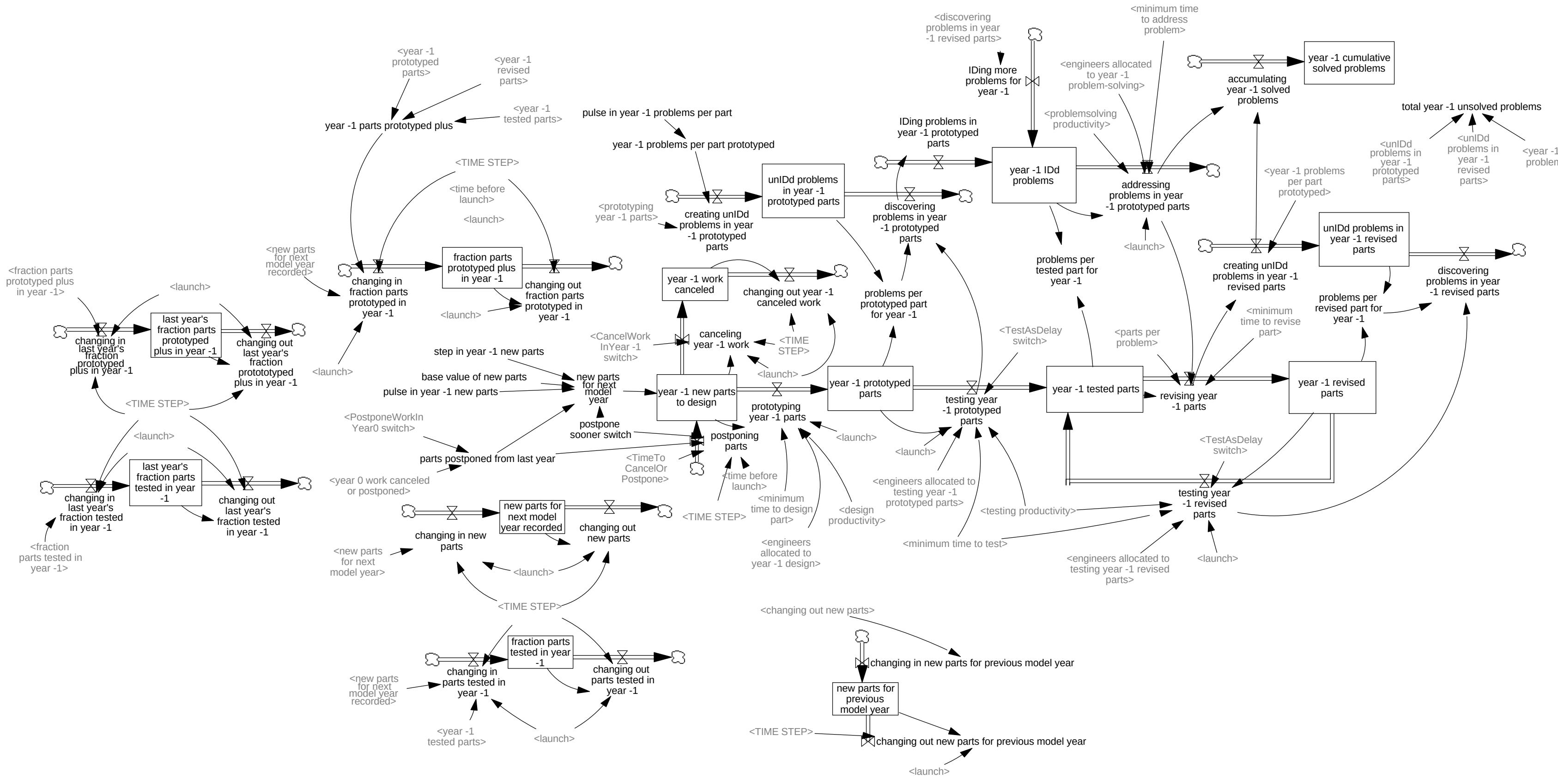
Wheelwright, S.C. and K.B. Clark (1992). *Revolutionizing Product Development: Quantum Leaps in Speed, Efficiency, and Quality*. New York: The Free Press.

Year 0 Parts and Problems



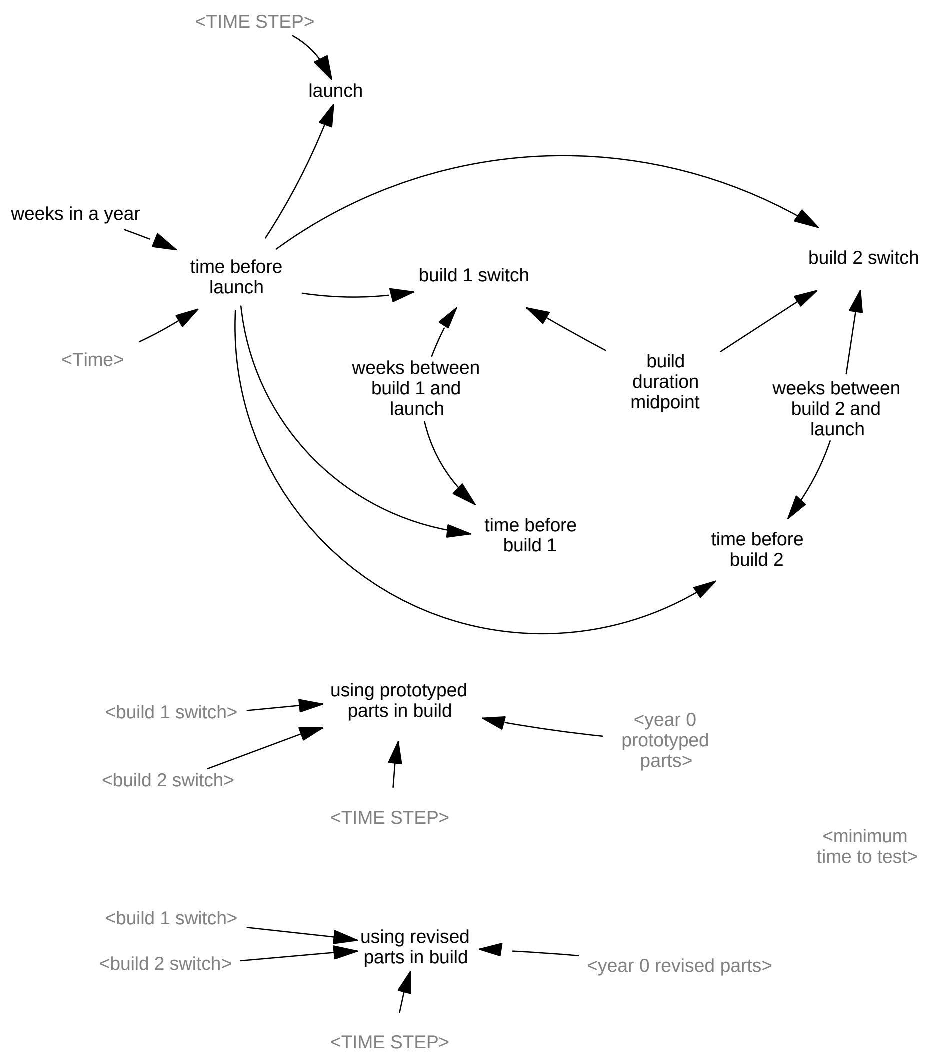
<TIME STEP>
<launch>
total problems for model year launched
changing in total probs
changing out total probs
<FixPercent OfProblems switch>
fraction year 0 problems fixed
minimum time to address problem
<problemsolving productivity>
<engineers allocated to year 0 problem-solving>
year 0 cumulative solved problems
accumulating year 0 solved problems
total year 0 unsolved problems
<year 0 unIDd problems in prototyped parts>
<year 0 unIDd problems in revised parts>
<year 0 IDd problems>
<discovering problems in year 0 revised parts>
IDing more problems for year 0
pulse in Year 0 problems per part
year 0 IDd problems
IDing problems for year 0
addressing problems for year 0
<year 0 problems per part prototyped>
testing accuracy fraction
year 0 problems per part prototyped
year 0 unIDd problems in prototyped parts
discovering problems in prototype parts for year 0
creating problems in prototype parts for year 0
year 0 unIDd problems in revised parts
<testing accuracy fraction>
discovering problems in year 0 revised parts
creating problems in revised parts for year 0
unID problems per revised part for year 0
<launch>
<TIME STEP>
year 0 work canceled or postponed
resetting year 0 cancelled parts
TimeTo CancelOr Postpone
<CancelWork in Year 0 switch>
<Postpone WorkInYear0 switch>
canceling or postponing year 0 work
<time before launch>
minimum time to design part
unIDd problems per prototyped part for year 0
IDd problems per part for year 0
<launch>
<TestAsDelay switch>
parts per problem
year 0 new parts to design
creating prototype parts for year 0
year 0 prototyped parts
testing prototyped parts for year 0
year 0 parts used in build or test
revising parts for year 0
year 0 revised parts
initial year 0 parts to design
<time before launch>
design productivity
<launch>
<TimeTo CancelOr Postpone>
<launch>
<launch>
minimum time to revise part
<TestAsDelay switch>
<engineers allocated to year 0 design>
<minimum time to test>
<using prototyped parts in build>
<engineers allocated to year 0 testing prototyped parts>
testing productivity
testing revised parts for year 0
<using revised parts in build>
<minimum time to test>
<launch>
<engineers allocated to year 0 testing revised parts>

Year -1 Parts and Problems

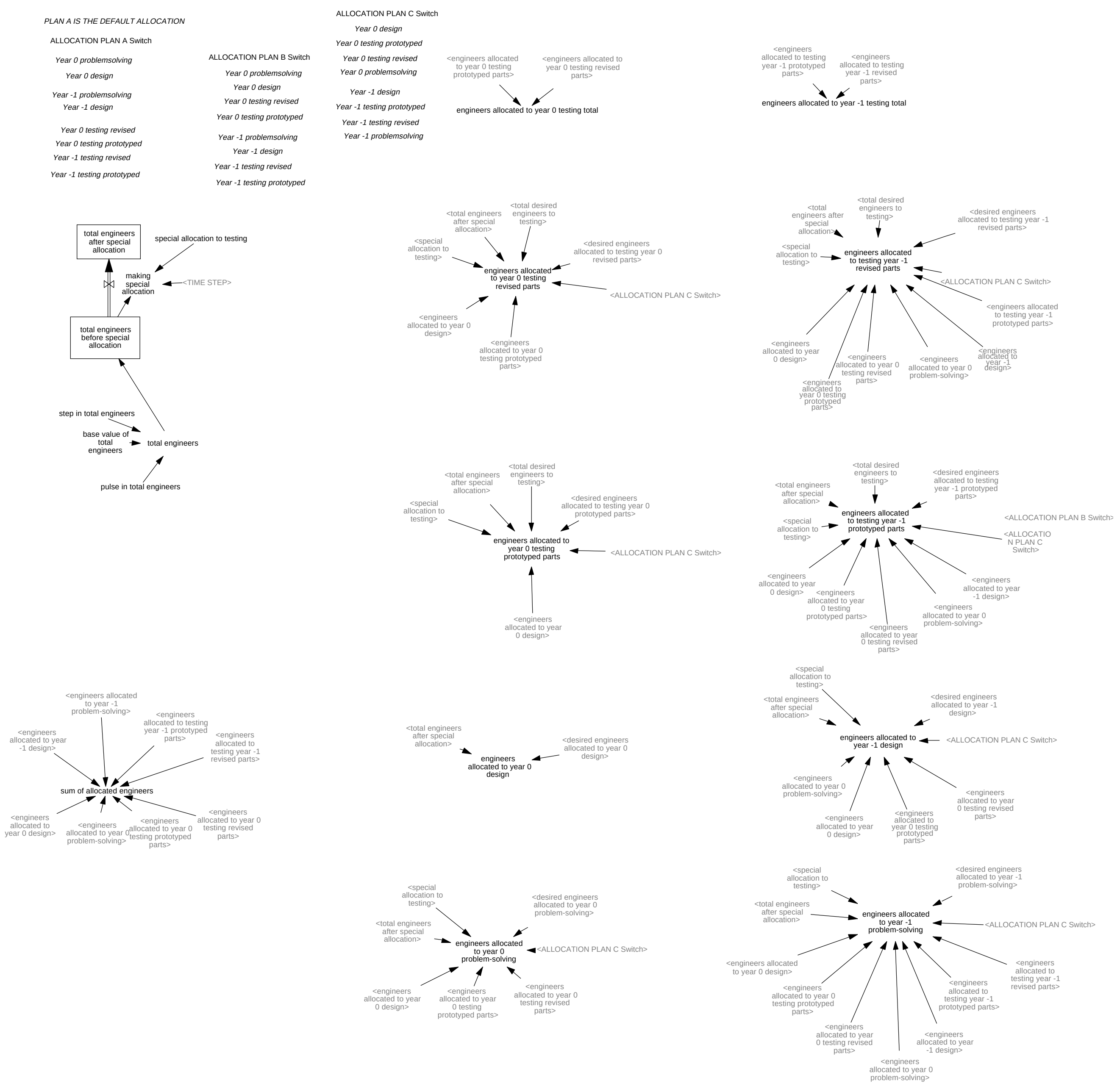


<discovering problems in year -1 revised parts>
<minimum time to address problem>
year -1 cumulative solved problems
IDing more problems for year -1
<engineers allocated to year -1 problem-solving>
accumulating year -1 solved problems
<year -1 prototyped parts>
<year -1 revised parts>
<year -1 tested parts>
year -1 parts prototyped plus
pulse in year -1 problems per part
year -1 problems per part prototyped
<problemsolving productivity>
IDing problems in year -1 prototyped parts
total year -1 unsolved problems
<unIDd problems in year -1 prototyped parts>
<unIDd problems in year -1 revised parts>
<year -1 IDd problems>
<TIME STEP>
<time before launch>
<launch>
unIDd problems in year -1 prototyped parts
year -1 IDd problems
addressing problems in year -1 prototyped parts
<prototyping year -1 parts>
creating unIDd problems in year -1 prototyped parts
discovering problems in year -1 prototyped parts
<year -1 problems per part prototyped>
unIDd problems in year -1 revised parts
<new parts for next model year recorded>
fraction parts prototyped plus in year -1
<fraction parts prototyped plus in year -1>
changing in fraction parts prototyped in year -1
changing out fraction parts prototyped in year -1
year -1 work canceled
changing out year -1 canceled work
problems per prototyped part for year -1
problems per tested part for year -1
creating unIDd problems in year -1 revised parts
discovering problems in year -1 revised parts
last year's fraction parts prototyped plus in year -1
changing in last year's fraction prototyped plus in year -1
changing out last year's fraction protototyped plus in year -1
<CancelWorkInYear -1 switch>
canceling year -1 work
<TIME STEP>
<TestAsDelay switch>
<parts per problem>
<minimum time to revise part>
problems per revised part for year -1
step in year -1 new parts
base value of new parts
pulse in year -1 new parts
new parts for next model year
year -1 new parts to design
prototyping year -1 parts
year -1 prototyped parts
testing year -1 prototyped parts
year -1 tested parts
revising year -1 parts
year -1 revised parts
<PostponeWorkInYear0 switch>
postpone sooner switch
parts postponed from last year
<TimeToCancelOrPostpone>
postponing parts
<time before launch>
<year 0 work canceled or postponed>
last year's fraction parts tested in year -1
changing in last year's fraction tested in year -1
changing out last year's fraction tested in year -1
<fraction parts tested in year -1>
new parts for next model year recorded
changing in new parts
changing out new parts
<new parts for next model year>
<TIME STEP>
<minimum time to design part>
<design productivity>
<engineers allocated to year -1 design>
<engineers allocated to testing year -1 prototyped parts>
<testing productivity>
<minimum time to test>
testing year -1 revised parts
<engineers allocated to testing year -1 revised parts>
fraction parts tested in year -1
changing in parts tested in year -1
changing out parts tested in year -1
<year -1 tested parts>
<changing out new parts>
changing in new parts for previous model year
new parts for previous model year
changing out new parts for previous model year

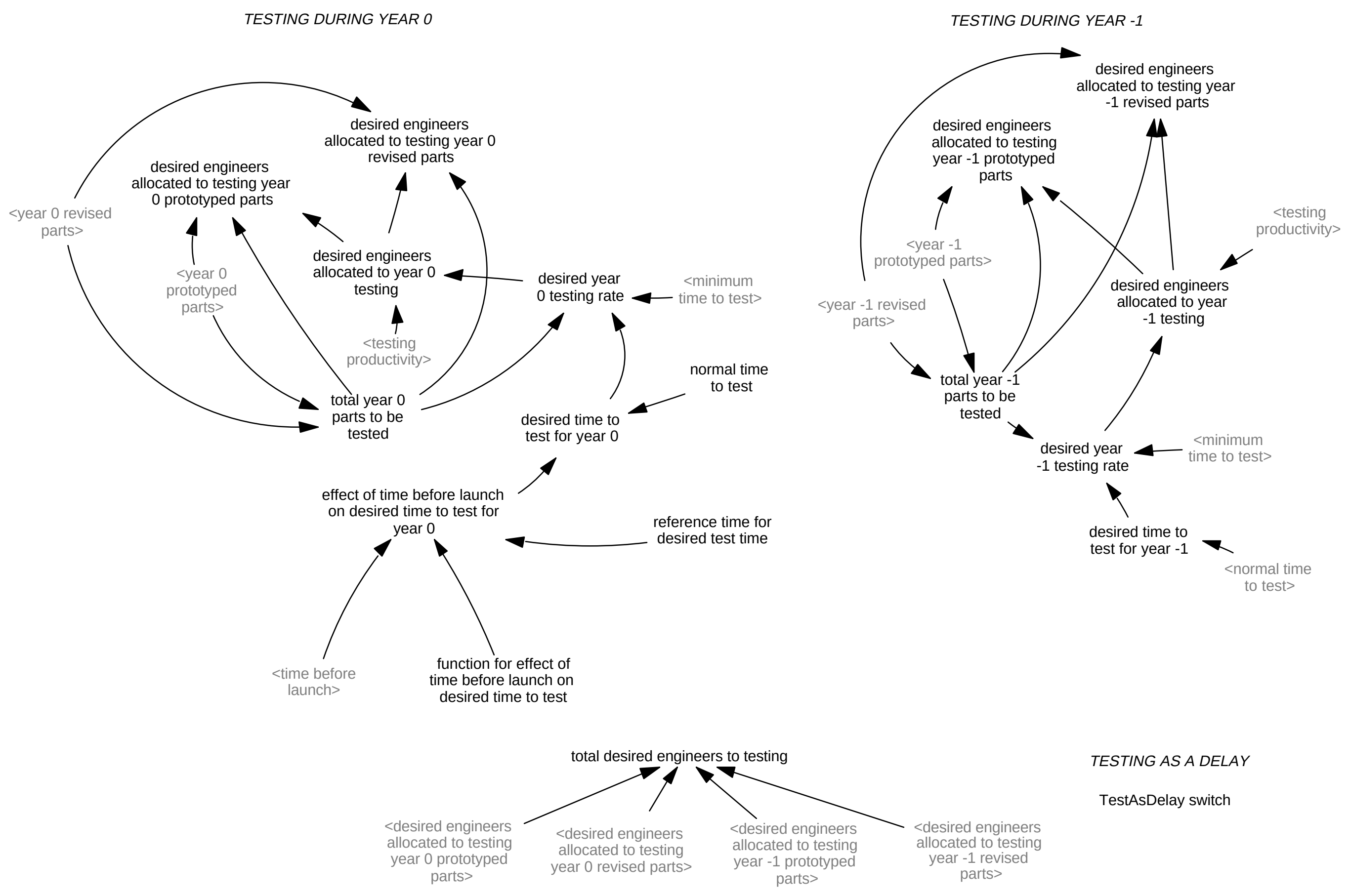
Launch Countdown



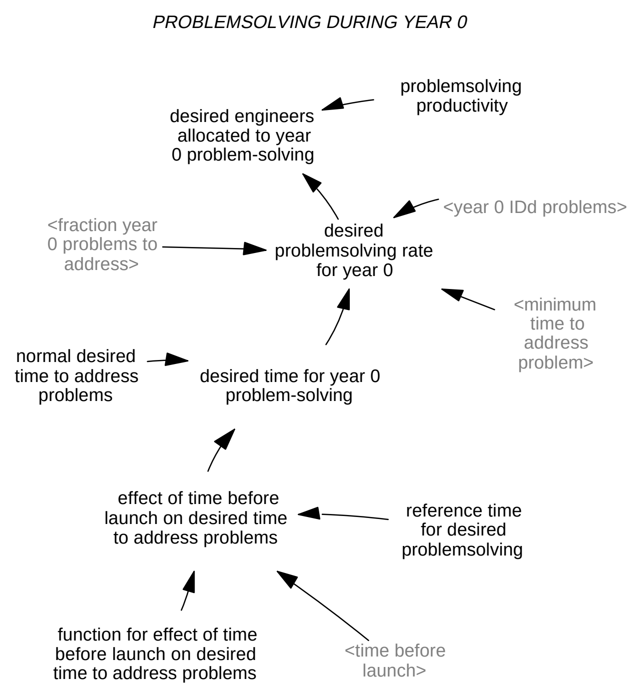
Allocating Engineers to Activities

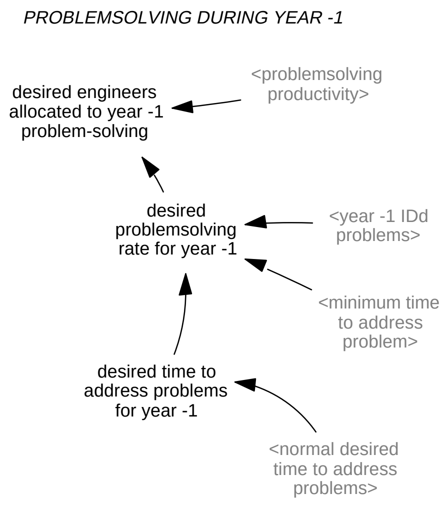

PLAN A IS THE DEFAULT ALLOCATION
ALLOCATION PLAN A Switch
Year 0 problemsolving
Year 0 design
Year -1 problemsolving
Year -1 design
Year 0 testing revised
Year 0 testing prototyped
Year -1 testing revised
Year -1 testing prototyped
ALLOCATION PLAN B Switch
Year 0 problemsolving
Year 0 design
Year 0 testing revised
Year 0 testing prototyped
Year -1 problemsolving
Year -1 design
Year -1 testing revised
Year -1 testing prototyped
ALLOCATION PLAN C Switch
Year 0 design
Year 0 testing prototyped
Year 0 testing revised
Year 0 problemsolving
Year -1 design
Year -1 testing prototyped
Year -1 testing revised
Year -1 problemsolving
<engineers allocated to year 0 testing prototyped parts>
<engineers allocated to year 0 testing revised parts>
engineers allocated to year 0 testing total
<engineers allocated to testing year -1 prototyped parts>
<engineers allocated to testing year -1 revised parts>
engineers allocated to year -1 testing total
total engineers after special allocation
special allocation to testing
making special allocation
<TIME STEP>
total engineers before special allocation
step in total engineers
base value of total engineers
total engineers
pulse in total engineers
engineers allocated to year 0 testing revised parts
engineers allocated to testing year -1 revised parts
engineers allocated to year 0 testing prototyped parts
engineers allocated to testing year -1 prototyped parts
engineers allocated to year -1 design
engineers allocated to year 0 design
engineers allocated to year 0 problem-solving
engineers allocated to year -1 problem-solving
sum of allocated engineers
<total engineers after special allocation>
<total desired engineers to testing>
<special allocation to testing>
<desired engineers allocated to testing year 0 revised parts>
<desired engineers allocated to testing year -1 revised parts>
<desired engineers allocated to testing year 0 prototyped parts>
<desired engineers allocated to testing year -1 prototyped parts>
<desired engineers allocated to year -1 design>
<desired engineers allocated to year 0 design>
<desired engineers allocated to year 0 problem-solving>
<desired engineers allocated to year -1 problem-solving>
<ALLOCATION PLAN C Switch>
<ALLOCATION PLAN B Switch>
<ALLOCATION PLAN C Switch>
<engineers allocated to year 0 design>
<engineers allocated to year -1 design>
<engineers allocated to year 0 problem-solving>
<engineers allocated to year -1 problem-solving>
<engineers allocated to year 0 testing prototyped parts>
<engineers allocated to year 0 testing revised parts>
<engineers allocated to testing year -1 prototyped parts>
<engineers allocated to testing year -1 revised parts>

Desired Allocation to Testing

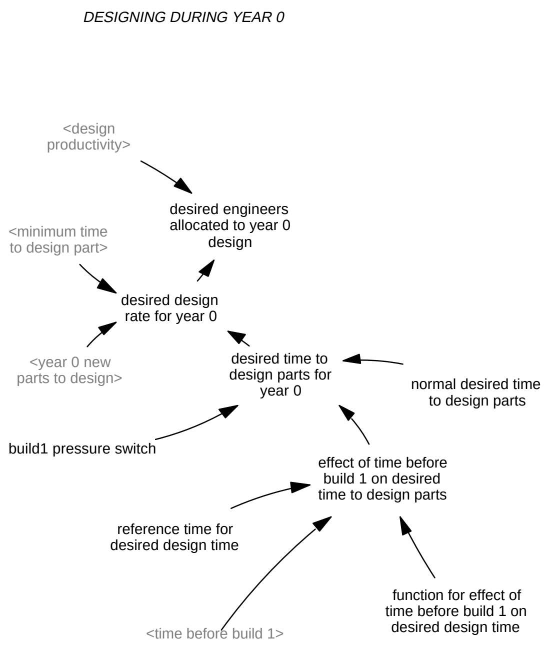

TESTING DURING YEAR 0
desired engineers allocated to testing year 0 revised parts
desired engineers allocated to testing year 0 prototyped parts
<year 0 revised parts>
<year 0 prototyped parts>
desired engineers allocated to year 0 testing
desired year 0 testing rate
<minimum time to test>
<testing productivity>
total year 0 parts to be tested
normal time to test
desired time to test for year 0
effect of time before launch on desired time to test for year 0
reference time for desired test time
<time before launch>
function for effect of time before launch on desired time to test
TESTING DURING YEAR -1
desired engineers allocated to testing year -1 revised parts
desired engineers allocated to testing year -1 prototyped parts
<testing productivity>
<year -1 prototyped parts>
<year -1 revised parts>
desired engineers allocated to year -1 testing
total year -1 parts to be tested
desired year -1 testing rate
<minimum time to test>
desired time to test for year -1
<normal time to test>
total desired engineers to testing
<desired engineers allocated to testing year 0 prototyped parts>
<desired engineers allocated to testing year 0 revised parts>
<desired engineers allocated to testing year -1 prototyped parts>
<desired engineers allocated to testing year -1 revised parts>
TESTING AS A DELAY
TestAsDelay switch

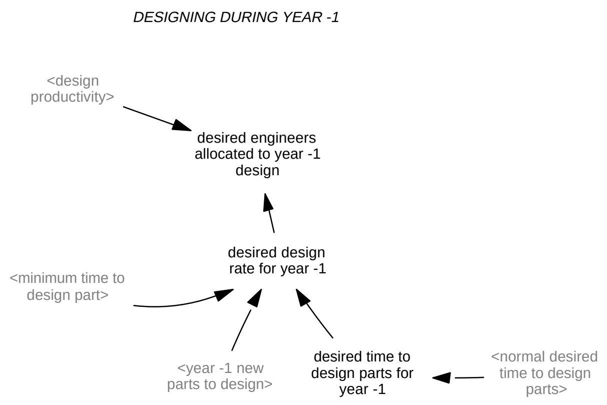
Desired Allocation to Problem-Solving


PROBLEMSOLVING DURING YEAR 0
desired engineers allocated to year 0 problem-solving
problemsolving productivity
<year 0 IDd problems>
<fraction year 0 problems to address>
desired problemsolving rate for year 0
<minimum time to address problem>
normal desired time to address problems
desired time for year 0 problem-solving
effect of time before launch on desired time to address problems
reference time for desired problemsolving
function for effect of time before launch on desired time to address problems
<time before launch>

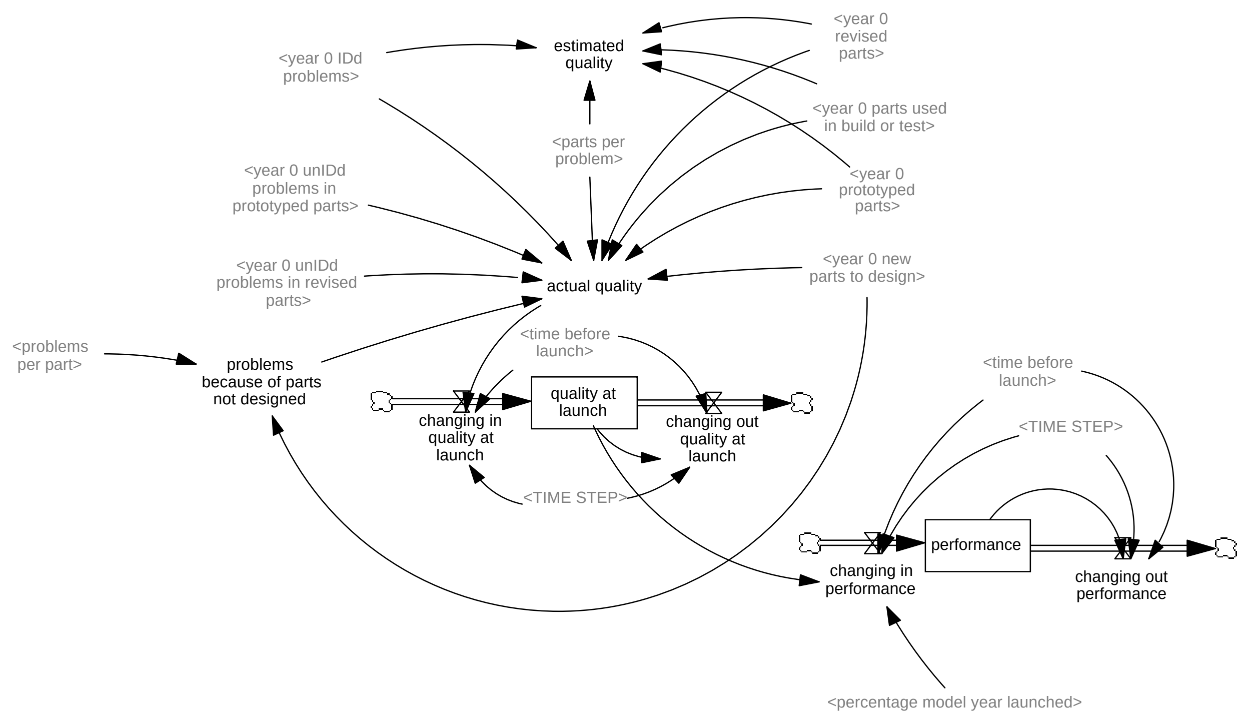

PROBLEMSOLVING DURING YEAR -1
desired engineers allocated to year -1 problem-solving
<problemsolving productivity>
desired problemsolving rate for year -1
<year -1 IDd problems>
<minimum time to address problem>
desired time to address problems for year -1
<normal desired time to address problems>

Desired Allocation to Design

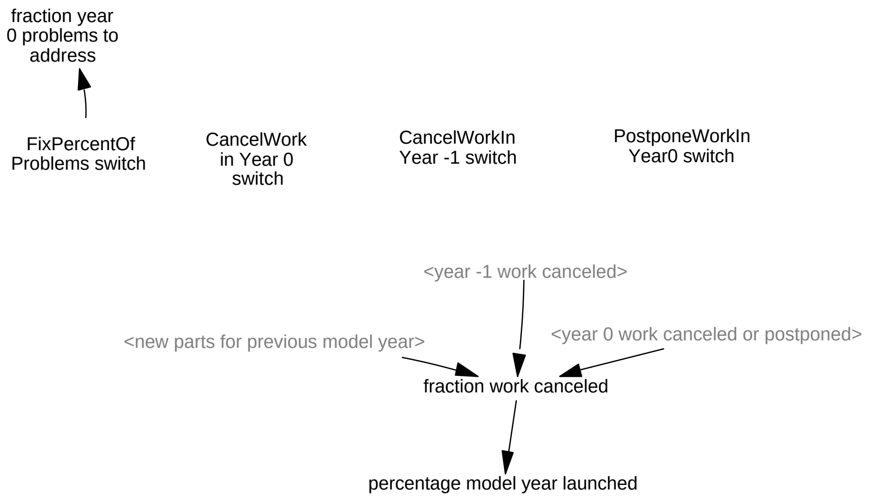

DESIGNING DURING YEAR 0
<design productivity>
desired engineers allocated to year 0 design
<minimum time to design part>
desired design rate for year 0
<year 0 new parts to design>
desired time to design parts for year 0
normal desired time to design parts
build1 pressure switch
effect of time before build 1 on desired time to design parts
reference time for desired design time
function for effect of time before build 1 on desired design time
<time before build 1>


DESIGNING DURING YEAR -1
<design productivity>
desired engineers allocated to year -1 design
desired design rate for year -1
<minimum time to design part>
<year -1 new parts to design>
desired time to design parts for year -1
<normal desired time to design parts>

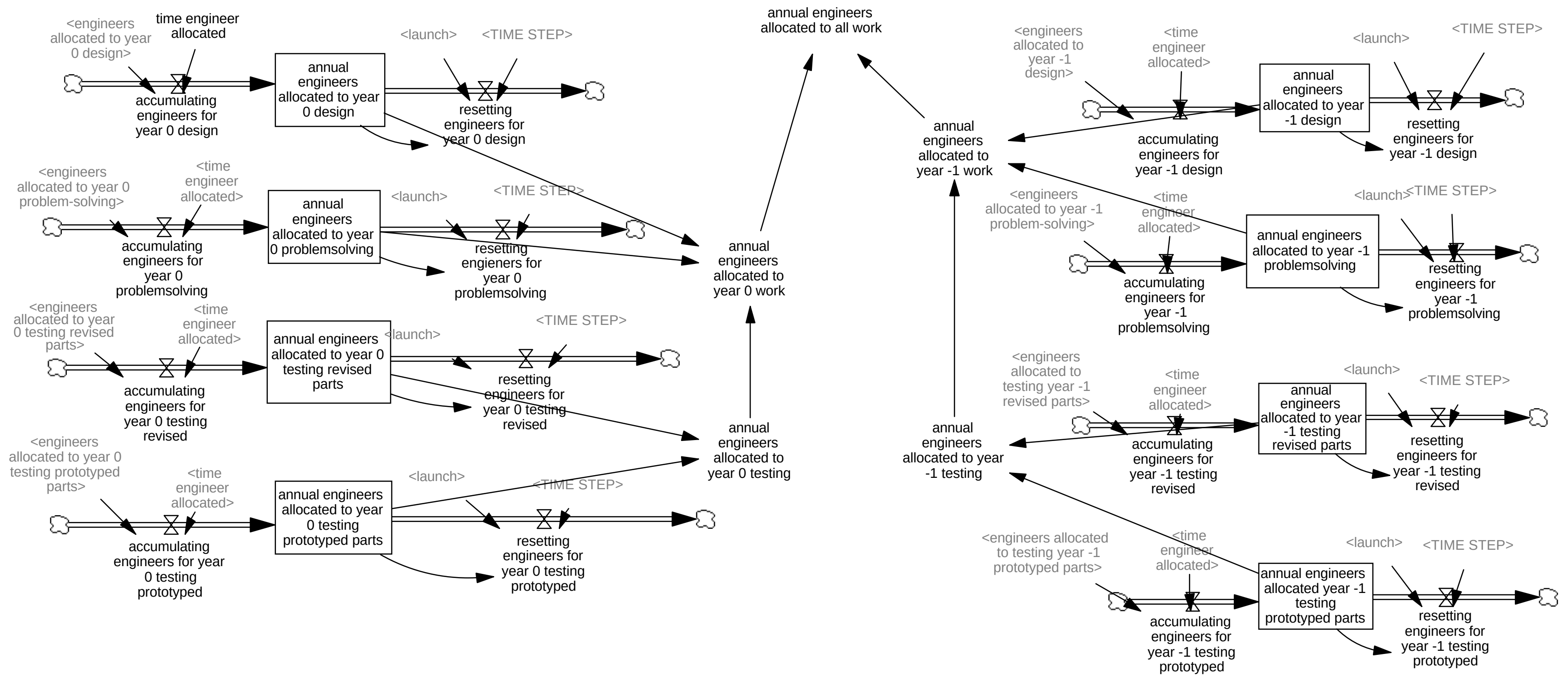
Quality and Performance


estimated quality
<year 0 IDd problems>
<year 0 revised parts>
<year 0 parts used in build or test>
<parts per problem>
<year 0 unIDd problems in prototyped parts>
<year 0 prototyped parts>
<year 0 unIDd problems in revised parts>
actual quality
<year 0 new parts to design>
<problems per part>
problems because of parts not designed
<time before launch>
quality at launch
changing in quality at launch
changing out quality at launch
<TIME STEP>
<time before launch>
<TIME STEP>
performance
changing in performance
changing out performance
<percentage model year launched>

Policy Switches


fraction year 0 problems to address
FixPercentOf Problems switch
CancelWork in Year 0 switch
CancelWorkIn Year -1 switch
PostponeWorkIn Year0 switch
<year -1 work canceled>
<new parts for previous model year>
<year 0 work canceled or postponed>
fraction work canceled
percentage model year launched

Recordkeeping of Engineering-Hours



<engineers allocated to year 0 design>
time engineer allocated
annual engineers allocated to year 0 design
accumulating engineers for year 0 design
<launch>
<TIME STEP>
resetting engineers for year 0 design
annual engineers allocated to all work
<engineers allocated to year 0 problem-solving>
<time engineer allocated>
annual engineers allocated to year 0 problemsolving
accumulating engineers for year 0 problemsolving
resetting engieners for year 0 problemsolving
annual engineers allocated to year 0 work
<engineers allocated to year 0 testing revised parts>
annual engineers allocated to year 0 testing revised parts
accumulating engineers for year 0 testing revised
resetting engineers for year 0 testing revised
<engineers allocated to year 0 testing prototyped parts>
annual engineers allocated to year 0 testing prototyped parts
accumulating engineers for year 0 testing prototyped
resetting engineers for year 0 testing prototyped
annual engineers allocated to year 0 testing
<engineers allocated to year -1 design>
annual engineers allocated to year -1 design
accumulating engineers for year -1 design
resetting engineers for year -1 design
annual engineers allocated to year -1 work
<engineers allocated to year -1 problem-solving>
annual engineers allocated to year -1 problemsolving
accumulating engineers for year -1 problemsolving
resetting engineers for year -1 problemsolving
<engineers allocated to testing year -1 revised parts>
annual engineers allocated to year -1 testing revised parts
accumulating engineers for year -1 testing revised
resetting engineers for year -1 testing revised
annual engineers allocated to year -1 testing
<engineers allocated to testing year -1 prototyped parts>
annual engineers allocated year -1 testing prototyped parts
accumulating engineers for year -1 testing prototyped
resetting engineers for year -1 testing prototyped

(001) "accumulating engineers for year -1 design"=
"engineers allocated to year -1 design"/time engineer allocated
Units: (people*Week)/Week

(002) "accumulating engineers for year -1 problemsolving"=
"engineers allocated to year -1 problem-solving"/time engineer allocated
Units: (people*Week)/Week

(003) "accumulating engineers for year -1 testing prototyped"=
"engineers allocated to testing year -1 prototyped parts"/time engineer allocated
Units: (people*Week)/Week

(004) "accumulating engineers for year -1 testing revised"=
"engineers allocated to testing year -1 revised parts"/time engineer allocated
Units: (people*Week)/Week

(005) accumulating engineers for year 0 design=
engineers allocated to year 0 design/time engineer allocated
Units: (people*Week)/Week

(006) accumulating engineers for year 0 problemsolving=
"engineers allocated to year 0 problem-solving"/time engineer allocated
Units: (people*Week)/Week

(007) accumulating engineers for year 0 testing prototyped=
engineers allocated to year 0 testing prototyped parts/time engineer allocated
Units: (people*Week)/Week

(008) accumulating engineers for year 0 testing revised=
engineers allocated to year 0 testing revised parts/time engineer allocated
Units: (people*Week)/Week

(009) "accumulating year -1 solved problems"=
"addressing problems in year -1 prototyped parts"-"creating unIDd problems in year -1 revised parts"
Units: problems/Week

(010) accumulating year 0 solved problems=
addressing problems for year 0-creating problems in revised parts for year 0
Units: problems/Week

(011) actual quality=
1-(ZIDZ(year 0 IDd problems+year 0 unIDd problems in prototyped parts+year 0 unIDd problems in revised parts+problems because of parts not designed
,year 0 parts used in build or test+year 0 prototyped parts+year 0 revised parts
+year 0 new parts to design))*parts per problem
Units: dimensionless

(012) addressing problems for year 0=
IF THEN ELSE (launch=1,0,fraction year 0 problems fixed*MIN(year 0 IDd problems
/minimum time to address problem,"engineers allocated to year 0 problem-solving"
*problemsolving productivity))

```
        Units: problems/Week

(013)   "addressing problems in year -1 prototyped parts"=
                IF THEN ELSE (launch=1,0,MIN("engineers allocated to year -1 problem-solving"
        *problemsolving productivity,"year -1 IDd problems"/minimum time to address problem))
        Units: problems/Week

(014)   ALLOCATION PLAN A Switch=0
        Units: dimensionless

(015)   ALLOCATION PLAN B Switch=0
        Units: dimensionless

(016)   ALLOCATION PLAN C Switch=1
        Units: dimensionless

(017)   annual engineers allocated to all work=
                "annual engineers allocated to year -1 work"+annual engineers allocated to year 0 work
        Units: Week*people

(018)   "annual engineers allocated to year -1 design"= INTEG (
                "accumulating engineers for year -1 design"-"resetting engineers for year -1 design",0)
        Units: Week*people

(019)   "annual engineers allocated to year -1 problemsolving"= INTEG (
                "accumulating engineers for year -1 problemsolving"-"resetting engineers for year -1
                problemsolving",0)
        Units: Week*people

(020)   "annual engineers allocated to year -1 testing"=
                "annual engineers allocated to year -1 testing revised parts"+"annual engineers allocated
                        year -1 testing prototyped parts"
        Units: Week*people

(021)   "annual engineers allocated to year -1 testing revised parts"= INTEG (
                "accumulating engineers for year -1 testing revised"-"resetting engineers for year -1
                testing revised",0)
        Units: Week*people

(022)   "annual engineers allocated to year -1 work"=
                "annual engineers allocated to year -1 design"+"annual engineers allocated to year -1
                problemsolving"+"annual engineers allocated to year -1 testing"
        Units: Week*people

(023)   annual engineers allocated to year 0 design= INTEG (
                accumulating engineers for year 0 design-resetting engineers for year 0 design,0)
        Units: people*Week

(024)   annual engineers allocated to year 0 problemsolving= INTEG (
                accumulating engineers for year 0 problemsolving-resetting engieners for year 0
                problemsolving,0)
        Units: people*Week
```

(025) annual engineers allocated to year 0 testing=
annual engineers allocated to year 0 testing prototyped parts+annual engineers allocated to year 0 testing revised parts
Units: Week*people

(026) annual engineers allocated to year 0 testing prototyped parts= INTEG (
accumulating engineers for year 0 testing prototyped-resetting engineers for year 0 testing prototyped,0)
Units: people*Week

(027) annual engineers allocated to year 0 testing revised parts= INTEG (
accumulating engineers for year 0 testing revised-resetting engineers for year 0 testing revised,0)
Units: people*Week

(028) annual engineers allocated to year 0 work=
annual engineers allocated to year 0 design+annual engineers allocated to year 0 problemsolving+annual engineers allocated to year 0 testing
Units: Week*people

(029) "annual engineers allocated year -1 testing prototyped parts"= INTEG (
"accumulating engineers for year -1 testing prototyped"-"resetting engineers for year -1 testing prototyped",0)
Units: Week*people

(030) base value of new parts=700
Units: parts

(031) base value of total engineers=85
Units: people

(032) build 1 switch=
IF THEN ELSE (time before launch>weeks between build 1 and launch-build duration midpoint :AND: time before launch < weeks between build 1 and launch+build duration midpoint, 1,0)*0
Units: dimensionless

(033) build 2 switch=
IF THEN ELSE (time before launch>weeks between build 2 and launch-build duration midpoint:AND: time before launch<weeks between build 2 and launch+build duration midpoint, 1,0)*0
Units: dimensionless

(034) build duration midpoint=2.5
Units: Week

(035) build1 pressure switch=1
Units: dimensionless

(036) canceling or postponing year 0 work=
IF THEN ELSE (time before launch=TimeToCancelOrPostpone,(CancelWork in Year 0 switch+PostponeWorkInYear0 switch)*year 0 new parts to design/TIME STEP,0)
Units: parts/Week

(037) "canceling year -1 work"=
IF THEN ELSE (launch=1,"CancelWorkInYear -1 switch"*"year -1 new parts to design" /TIME STEP,0)
Units: part/Week

(038) CancelWork in Year 0 switch=0
Units: dimensionless

(039) "CancelWorkInYear -1 switch"=0
Units: dimensionless

(040) "changing in fraction parts prototyped in year -1"=
IF THEN ELSE (launch=1,("year -1 parts prototyped plus"/new parts for next model year recorded)/TIME STEP,0)
Units: dimensionless/Week

(041) changing in IDd problems for year 0=
IF THEN ELSE (launch=1,"changing out IDd problems for year -1",0)
Units: problems/Week

(042) "changing in last year's fraction prototyped plus in year -1"=
IF THEN ELSE (launch=1, "fraction parts prototyped plus in year -1"/TIME STEP,0)
Units: dimensionless/Week

(043) "changing in last year's fraction tested in year -1"=
IF THEN ELSE (launch=1,"fraction parts tested in year -1"/TIME STEP,0)
Units: dimensionless/Week

(044) changing in new parts=
IF THEN ELSE (launch=1,new parts for next model year/TIME STEP,0)
Units: part/Week

(045) changing in new parts for previous model year=
changing out new parts
Units: parts/Week

(046) "changing in new parts for year -1"=
IF THEN ELSE (launch=1,new parts for next model year/TIME STEP,0)
Units: parts/Week

(047) changing in new parts for year 0=
IF THEN ELSE (launch=1,"changing out new parts for year -1",0)
Units: parts/Week

(048) "changing in parts tested in year -1"=
IF THEN ELSE (launch=1,"year -1 tested parts"/new parts for next model year recorded /TIME STEP,0)
Units: dimensionless/Week

(049) changing in performance=
IF THEN ELSE (time before launch=TIME STEP,quality at launch*percentage model year launched/TIME STEP,0)

Units: dimensionless/Week

(050) changing in prototyped parts for year 0=
IF THEN ELSE (launch=1,"changing out parts prototyped for year -1",0)
Units: parts/Week

(051) changing in quality at launch=
IF THEN ELSE (time before launch=TIME STEP+TIME STEP,actual quality/TIME STEP,0)
Units: dimensionless/Week

(052) changing in revised parts for year 0=
IF THEN ELSE (launch=1,"changing out rp for year -1",0)
Units: parts/Week

(053) changing in tested parts for year 0=
IF THEN ELSE (launch=1,"changing out tp for year -1",0)
Units: parts/Week

(054) changing in total probs=
IF THEN ELSE (launch=1, (total year 0 unsolved problems+year 0 cumulative solved problems)/TIME STEP,0)
Units: problems/Week

(055) changing in unIDd problems in prototyped parts for year 0=
IF THEN ELSE (launch=1,"changing out unIDd problems in pp for year -1",0)
Units: problems/Week

(056) changing in unIDd problems in revised parts for year 0=
IF THEN ELSE (launch=1,"changing out unIDd problems in rp for year -1",0)
Units: problems/Week

(057) changing in year 0 cumulative solved problems=
IF THEN ELSE (launch=1,"changing out cumulative solved problems for year -1",0)
Units: problems/Week

(058) "changing out cumulative solved problems for year -1"=
IF THEN ELSE (launch=1,"year -1 cumulative solved problems"/TIME STEP,0)
Units: problems/Week

(059) "changing out fraction parts prototyped in year -1"=
IF THEN ELSE (launch=1,("fraction parts prototyped plus in year -1"/TIME STEP),0)
Units: dimensionless/Week

(060) "changing out IDd problems for year -1"=
IF THEN ELSE (launch=1,"year -1 IDd problems"/TIME STEP,0)
Units: problems/Week

(061) changing out IDd problems for year 0=
IF THEN ELSE (Switch to postpone rather than cancel unfinished work=1,0,IF THEN ELSE (launch=1,year 0 IDd problems/TIME STEP,0))
Units: problems/Week

(062) "changing out last year's fraction protototyped plus in year -1"=
IF THEN ELSE (launch=1,"last year's fraction parts prototyped plus in year -1" /TIME STEP,0)
Units: dimensionless/Week

(063) "changing out last year's fraction tested in year -1"=
IF THEN ELSE (launch=1,"last year's fraction parts tested in year -1"/TIME STEP,0)
Units: dimensionless/Week

(064) changing out new parts=
IF THEN ELSE (launch=1,new parts for next model year recorded/TIME STEP,0)
Units: part/Week

(065) changing out new parts for previous model year=
IF THEN ELSE (launch=1,new parts for previous model year/TIME STEP,0)
Units: parts/Week

(066) "changing out new parts for year -1"=
IF THEN ELSE (launch=1,"year -1 new parts to design"/TIME STEP,0)*(1- "CancelWorkInYear -1 switch")
Units: parts/Week

(067) changing out new parts for year 0=
IF THEN ELSE (Switch to postpone rather than cancel unfinished work=1,0,IF THEN ELSE (launch=1,year 0 new parts to design/TIME STEP,0))
Units: parts/Week

(068) "changing out parts prototyped for year -1"=
IF THEN ELSE (launch=1,"year -1 prototyped parts"/TIME STEP,0)
Units: parts/Week

(069) "changing out parts tested in year -1"=
IF THEN ELSE (launch=1,"fraction parts tested in year -1"/TIME STEP,0)
Units: dimensionless/Week

(070) changing out performance=
IF THEN ELSE (time before launch=TIME STEP,performance/TIME STEP,0)
Units: dimensionless/Week

(071) changing out prototyped for year 0=
IF THEN ELSE (Switch to postpone rather than cancel unfinished work=1,0,IF THEN ELSE(launch=1,year 0 prototyped parts/TIME STEP,0))
Units: parts/Week

(072) changing out quality at launch=
IF THEN ELSE (time before launch=TIME STEP+TIME STEP,quality at launch/TIME STEP,0)
Units: dimensionless/Week

(073) changing out revised parts for year 0=
IF THEN ELSE (Switch to postpone rather than cancel unfinished work=1,0,IF THEN ELSE (launch=1,year 0 revised parts/TIME STEP,0))
Units: parts/Week

(074) "changing out rp for year -1"=
IF THEN ELSE (launch=1,"year -1 revised parts"/TIME STEP,0)
Units: parts/Week

(075) changing out tested parts for year 0=
IF THEN ELSE (Switch to postpone rather than cancel unfinished work=1,IF THEN ELSE
(launch=1,(year 0 parts used in build or test-year 0 IDd problems*parts per problem)
/TIME STEP,0),IF THEN ELSE (launch=1,year 0 parts used in build or test/TIME STEP,0))
Units: parts/Week

(076) changing out total probs=
IF THEN ELSE (launch=1,total problems for model year launched/TIME STEP,0)
Units: problems/Week

(077) "changing out tp for year -1"=
IF THEN ELSE (launch=1,"year -1 tested parts"/TIME STEP,0)
Units: parts/Week

(078) changing out unID problems in revised parts for year 0=
IF THEN ELSE (Switch to postpone rather than cancel unfinished work=1,0,IF THEN
ELSE (launch=1,year 0 unIDd problems in revised parts/TIME STEP,0))
Units: problems/Week

(079) "changing out unIDd problems in pp for year -1"=
IF THEN ELSE (launch=1,"unIDd problems in year -1 prototyped parts"/TIME STEP
,0)*(1-"CancelWorkInYear -1 switch")
Units: problems/Week

(080) changing out unIDd problems in prototyped parts for year 0=
IF THEN ELSE (Switch to postpone rather than cancel unfinished work=1,0,IF THEN
ELSE (launch=1,year 0 unIDd problems in prototyped parts/TIME STEP,0))
Units: problems/Week

(081) "changing out unIDd problems in rp for year -1"=
IF THEN ELSE (launch=1,"unIDd problems in year -1 revised parts"/TIME STEP,0)
Units: problems/Week

(082) "changing out year -1 canceled work"=
IF THEN ELSE (launch=1,"year -1 work canceled"/TIME STEP,0)
Units: part/Week

(083) changing out year 0 cumulative solved problems=
IF THEN ELSE (launch=1,year 0 cumulative solved problems/TIME STEP,0)
Units: problems/Week

(084) creating problems in prototype parts for year 0=
creating prototype parts for year 0*year 0 problems per part prototyped
Units: problems/Week

(085) creating problems in revised parts for year 0=

revising parts for year 0*year 0 problems per part prototyped
Units: problems/Week

(086) creating prototype parts for year 0=
IF THEN ELSE (launch=1 :OR: time before launch=TimeToCancelOrPostpone,0,MIN (engineers allocated to year 0 design*design productivity, year 0 new parts to design /minimum time to design part))
Units: parts/Week

(087) "creating unIDd problems in year -1 prototyped parts"=
"prototyping year -1 parts"*"year -1 problems per part prototyped"
Units: problems/Week

(088) "creating unIDd problems in year -1 revised parts"=
"revising year -1 parts"*"year -1 problems per part prototyped"
Units: problems/Week

(089) design productivity=0.2
Units: part/(Week*person)

(090) "desired design rate for year -1"=
MIN("year -1 new parts to design"/"desired time to design parts for year -1", "year -1 new parts to design"/minimum time to design part)
Units: parts/Week

(091) desired design rate for year 0=
MIN(year 0 new parts to design/desired time to design parts for year 0,year 0 new parts
to design/minimum time to design part)
Units: part/Week

(092) "desired engineers allocated to testing year -1 prototyped parts"=
"desired engineers allocated to year -1 testing"*ZIDZ("year -1 prototyped parts", "total year -1 parts to be tested")
Units: people

(093) "desired engineers allocated to testing year -1 revised parts"=
"desired engineers allocated to year -1 testing"*ZIDZ("year -1 revised parts", "total year -1 parts to be tested")
Units: people

(094) desired engineers allocated to testing year 0 prototyped parts=
desired engineers allocated to year 0 testing*(ZIDZ(year 0 prototyped parts, total year 0 parts to be tested))
Units: people

(095) desired engineers allocated to testing year 0 revised parts=
desired engineers allocated to year 0 testing*(ZIDZ(year 0 revised parts, total year 0 parts to be tested))
Units: people

(096) "desired engineers allocated to year -1 design"=
"desired design rate for year -1"/design productivity
Units: people

(097) "desired engineers allocated to year -1 problem-solving"=
"desired problemsolving rate for year -1"/problemsolving productivity

Units: people

(098) "desired engineers allocated to year -1 testing"=
"desired year -1 testing rate"/testing productivity
Units: people

(099) desired engineers allocated to year 0 design=
desired design rate for year 0/design productivity
Units: people

(100) "desired engineers allocated to year 0 problem-solving"=
desired problemsolving rate for year 0/problemsolving productivity
Units: people

(101) desired engineers allocated to year 0 testing=
desired year 0 testing rate/testing productivity
Units: people

(102) "desired problemsolving rate for year -1"=
MIN("year -1 IDd problems"/"desired time to address problems for year -1",
"year -1 IDd problems"/minimum time to address problem)
Units: problems/Week

(103) desired problemsolving rate for year 0=
MIN(fraction year 0 problems to address*year 0 IDd problems/"desired time for year 0 problem-solving",fraction year 0 problems to address*year 0 IDd problems/minimum time

to address problem)
Units: problems/Week

(104) "desired time for year 0 problem-solving"=
normal desired time to address problems*effect of time before launch on desired time to address problems
Units: weeks

(105) "desired time to address problems for year -1"=
normal desired time to address problems
Units: weeks

(106) "desired time to design parts for year -1"=
normal desired time to design parts
Units: weeks

(107) desired time to design parts for year 0=
IF THEN ELSE (build1 pressure switch=1,normal desired time to design parts
*effect of time before build 1 on desired time to design parts,normal desired time to design parts)
Units: weeks

(108) "desired time to test for year -1"=
normal time to test
Units: weeks

(109) desired time to test for year 0=
normal time to test*effect of time before launch on desired time to test for year 0
Units: weeks

(110) "desired year -1 testing rate"=
MIN("total year -1 parts to be tested"/"desired time to test for year -1","total year -1 parts
to be tested"/minimum time to test)
Units: parts/Week

(111) desired year 0 testing rate=
MIN(total year 0 parts to be tested/desired time to test for year 0,total year 0 parts to be tested/minimum time to test)
Units: parts/Week

(112) discovering problems in prototype parts for year 0=
testing prototyped parts for year 0*unIDd problems per prototyped part for year 0
*testing accuracy fraction
Units: problems/Week

(113) "discovering problems in year -1 prototyped parts"=
"testing year -1 prototyped parts"*"problems per prototyped part for year -1"
Units: problems/Week

(114) "discovering problems in year -1 revised parts"=
"testing year -1 revised parts"*"problems per revised part for year -1"
Units: problems/Week

(115) discovering problems in year 0 revised parts=
testing revised parts for year 0*unID problems per revised part for year 0
*testing accuracy fraction
Units: problems/Week

(116) effect of time before build 1 on desired time to design parts=
function for effect of time before build 1 on desired design time(time before build 1
/reference time for desired design time)
Units: dimensionless

(117) effect of time before launch on desired time to address problems=
function for effect of time before launch on desired time to address problems
(time before launch/reference time for desired problemsolving)
Units: dimensionless

(118) effect of time before launch on desired time to test for year 0=
function for effect of time before launch on desired time to test(time before launch
/reference time for desired test time)
Units: dimensionless

(119) "engineers allocated to testing year -1 prototyped parts"=
ALLOCATION PLAN C Switch*IF THEN ELSE (special allocation to testing>0,MIN
("desired engineers allocated to testing year -1 prototyped parts",special allocation to testing*"desired engineers allocated to testing year -1 prototyped parts"/total desired engineers to testing),MIN("desired engineers allocated to testing year -1 prototyped parts",total engineers after special allocation-engineers allocated to year 0 design-

engineers allocated to year 0 testing prototyped parts-engineers allocated to year 0 testing revised parts-"engineers allocated to year 0 problem-solving"-"engineers allocated to year -1 design"))
Units: people

(120) "engineers allocated to testing year -1 revised parts"=
ALLOCATION PLAN C Switch*(IF THEN ELSE (special allocation to testing>0,MIN ("desired engineers allocated to testing year -1 revised parts",special allocation to testing *"desired engineers allocated to testing year -1 revised parts"/total desired engineers to testing),MIN("desired engineers allocated to testing year -1 revised parts",total engineers after special allocation-engineers allocated to year 0 design-engineers allocated to year 0 testing prototyped parts-engineers allocated to year 0 testing revised parts-"engineers allocated to year 0 problem-solving"-"engineers allocated to year -1 design"-"engineers allocated to testing year -1 prototyped parts")))
Units: people

(121) "engineers allocated to year -1 design"=
ALLOCATION PLAN C Switch*IF THEN ELSE (special allocation to testing>0, MIN ("desired engineers allocated to year -1 design",total engineers after special allocation -engineers allocated to year 0 design-"engineers allocated to year 0 problem-solving"), MIN("desired engineers allocated to year -1 design", total engineers after special allocation-engineers allocated to year 0 design-engineers allocated to year 0 testing prototyped parts-engineers allocated to year 0 testing revised parts-"engineers allocated to year 0 problem-solving"))
Units: people

(122) "engineers allocated to year -1 problem-solving"=
ALLOCATION PLAN C Switch*IF THEN ELSE (special allocation to testing>0, MIN ("desired engineers allocated to year -1 problem-solving",total engineers after special allocation-engineers allocated to year 0 design-"engineers allocated to year 0 problem-solving"-"engineers allocated to year -1 design"), MIN("desired engineers allocated to year -1 problem-solving",total engineers after special allocation-engineers allocated to year 0 design-engineers allocated to year 0 testing prototyped parts-engineers allocated to year 0 testing revised parts-"engineers allocated to year 0 problem-solving"-"engineers allocated to year -1 design"-"engineers allocated to testing year -1 prototyped parts"-"engineers allocated to testing year -1 revised parts"))
Units: people

(123) "engineers allocated to year -1 testing total"=
"engineers allocated to testing year -1 prototyped parts"+"engineers allocated to testing year -1 revised parts"
Units: people

(124) engineers allocated to year 0 design=
MIN(desired engineers allocated to year 0 design,total engineers after special allocation)
Units: people

(125) "engineers allocated to year 0 problem-solving"=
ALLOCATION PLAN C Switch*IF THEN ELSE (special allocation to testing>0, MIN("desired engineers allocated to year 0 problem-solving", total engineers after special allocation-engineers allocated to year 0 design), MIN("desired engineers allocated to year 0 problem-solving", total engineers after special allocation-engineers

allocated to year 0 engineers allocated to year	design-engineers allocated to year 0 testing prototyped parts-0 testing revised parts))
Units: people	

(126) engineers allocated to year 0 testing prototyped parts=
ALLOCATION PLAN C Switch*IF THEN ELSE (special allocation to testing>0,MIN (desired engineers allocated to testing year 0 prototyped parts,special allocation to testing*desired engineers allocated to testing year 0 prototyped parts/total desired engineers to testing),MIN(desired engineers allocated to testing year 0 prototyped parts,total engineers after special allocation-engineers allocated to year 0 design))
Units: people

(127) engineers allocated to year 0 testing revised parts=
ALLOCATION PLAN C Switch*(IF THEN ELSE(special allocation to testing>0,MIN (desired engineers allocated to testing year 0 revised parts,special allocation to testing *desired engineers allocated to testing year 0 revised parts/total desired engineers to testing),MIN(desired engineers allocated to testing year 0 revised parts,total engineers after special allocation-engineers allocated to year 0 design-engineers allocated to year 0 testing prototyped parts)))
Units: people

(128) engineers allocated to year 0 testing total=
engineers allocated to year 0 testing prototyped parts+engineers allocated to year 0 testing revised parts
Units: people

(129) estimated quality=
1-(ZIDZ(year 0 IDd problems,year 0 parts used in build or test+year 0 prototyped parts +year 0 revised parts))*parts per problem
Units: dimensionless

(130) FINAL TIME = 1040
Units: Week

(131) FixPercentOfProblems switch=0
Units: dimensionless

(132) "fraction parts prototyped plus in year -1"= INTEG (
"changing in fraction parts prototyped in year -1"-"changing out fraction parts prototyped in year -1",0)
Units: dimensionless

(133) "fraction parts tested in year -1"= INTEG (
"changing in parts tested in year -1"-"changing out parts tested in year -1",0)
Units: dimensionless

(134) fraction work canceled=
ZIDZ(("year -1 work canceled"+year 0 work canceled or postponed),new parts for previous model year)
Units: dimensionless

(135) fraction year 0 problems fixed=
IF THEN ELSE (FixPercentOfProblems switch=1,4,1)
Units: dimensionless

(136) fraction year 0 problems to address=
IF THEN ELSE (FixPercentOfProblems switch=1,0.2,1)

Units: dimensionless

(137) function for effect of time before build 1 on desired design time
([(0,0)-(2,2)],(0,0.2),(0.25,0.25),(0.5,0.45),(0.75,0.7),(1,1),(1.25,1.25),
(1.5,1.4),(1.75,1.45),(2,1.5))
Units: dimensionless

(138) function for effect of time before launch on desired time to address problems
([(0,0)-(2,2)],(0,0.3),(0.25,0.35),(0.5,0.5),(0.75,0.75),(1,1),(1.25,1.25),
(1.5,1.4),(1.75,1.45),(2,1.5))
Units: dimensionless

(139) function for effect of time before launch on desired time to test(
[(0,0)-(2,2)],(0,0.4),(0.2,0.5),(0.4,0.7),(0.6,0.85),(0.8,0.95),(1,1),
(1.5,1),(2,1))
Units: dimensionless

(140) IDd problems per part for year 0=
ZIDZ(year 0 IDd problems,year 0 parts used in build or test)
Units: problems/part

(141) "IDing more problems for year -1"=
"discovering problems in year -1 revised parts"
Units: problems/Week

(142) IDing more problems for year 0=
discovering problems in year 0 revised parts
Units: problems/Week

(143) IDing problems for year 0=
discovering problems in prototype parts for year 0
Units: problems/Week

(144) "IDing problems in year -1 prototyped parts"=
"discovering problems in year -1 prototyped parts"
Units: problems/Week

(145) INITIAL TIME = -208
Units: Week

(146) initial year 0 parts to design=0
Units: parts

(147) initial year 0 prototyped parts=0
Units: parts

(148) "last year's fraction parts prototyped plus in year -1"= INTEG (
"changing in last year's fraction prototyped plus in year -1"-"changing out last year's
fraction protototyped plus in year -1",0)
Units: dimensionless

(149) "last year's fraction parts tested in year -1"= INTEG (

"changing in last year's fraction tested in year -1"-"changing out last year's fraction tested in year -1",0)
Units: dimensionless

(150) launch=
IF THEN ELSE (time before launch=TIME STEP,1,0)
Units: weeks

(151) making special allocation=
(total engineers before special allocation-special allocation to testing)
/TIME STEP
Units: people/Week

(152) minimum time to address problem=1
Units: Week

(153) minimum time to design part=1
Units: Week

(154) minimum time to revise part=2
Units: Week

(155) minimum time to test=4
Units: Week

(156) new parts for next model year=
base value of new parts+parts postponed from last year*(1-postpone sooner switch)
+1*(STEP("step in year -1 new parts",-1)+"pulse in year -1 new parts"
*PULSE(103,52))
Units: parts

(157) new parts for next model year recorded= INTEG (
changing in new parts-changing out new parts,new parts for next model year)
Units: parts

(158) new parts for previous model year= INTEG (
changing in new parts for previous model year-changing out new parts for previous
model year,0)
Units: parts

(159) normal desired time to address problems=3.5
Units: weeks

(160) normal desired time to design parts=5
Units: weeks

(161) normal time to test=10
Units: weeks

(162) parts per problem=1
Units: part/problem

(163) parts postponed from last year=

year 0 work canceled or postponed*PostponeWorkInYear0 switch
Units: parts

```
(164)   percentage model year launched=
                1-fraction work canceled
        Units: dimensionless

(165)   performance= INTEG (
                changing in performance-changing out performance,0)
        Units: dimensionless

(166)   postpone sooner switch=0
        Units: dimensionless

(167)   PostponeWorkInYear0 switch=0
        Units: dimensionless

(168)   postponing parts=
                IF THEN ELSE (time before launch = TimeToCancelOrPostpone-3*TIME
                        STEP,postpone sooner switch *parts postponed from last year/TIME STEP,0)
        Units: parts/Week

(169)   problems because of parts not designed=
                year 0 new parts to design*problems per part
        Units: problems

(170)   problems per part=
                1/parts per problem
        Units: problem/part

(171)   problems per part revised=0.2
        Units: problems/part

(172)   "problems per prototyped part for year -1"=
                ZIDZ("unIDd problems in year -1 prototyped parts","year -1 prototyped parts")
        Units: problem/part

(173)   "problems per revised part for year -1"=
                ZIDZ("unIDd problems in year -1 revised parts","year -1 revised parts")
        Units: problems/part

(174)   "problems per tested part for year -1"=
                ZIDZ("year -1 IDd problems","year -1 tested parts")
        Units: problems/part

(175)   problemsolving productivity=0.3
        Units: problems/(Week*person)

(176)   "prototyping year -1 parts"=
                IF THEN ELSE (launch=1,0,MIN("year -1 new parts to design"/minimum time to design
                part,"engineers allocated to year -1 design"*design productivity))
        Units: parts/Week

(177)   pulse in total engineers=0
        Units: people
```

(178) "pulse in year -1 new parts"=0
Units: parts

(179) "pulse in year -1 problems per part"=0
Units: dimensionless

(180) pulse in Year 0 problems per part=0
Units: dimensionless

(181) quality at launch= INTEG (
changing in quality at launch-changing out quality at launch,0)
Units: dimensionless

(182) reference time for desired design time=60
Units: weeks

(183) reference time for desired problemsolving=50
Units: weeks

(184) reference time for desired test time=40
Units: weeks

(185) resetting engieners for year 0 problemsolving=
IF THEN ELSE (launch=1,annual engineers allocated to year 0 problemsolving
/TIME STEP,0)
Units: (people*Week)/Week

(186) "resetting engineers for year -1 design"=
IF THEN ELSE (launch=1,"annual engineers allocated to year -1 design"/TIME STEP,0)
Units: (people*Week)/Week

(187) "resetting engineers for year -1 problemsolving"=
IF THEN ELSE (launch=1,"annual engineers allocated to year -1 problemsolving"
/TIME STEP,0)
Units: (people*Week)/Week

(188) "resetting engineers for year -1 testing prototyped"=
IF THEN ELSE (launch=1,"annual engineers allocated year -1 testing prototyped parts"
/TIME STEP,0)
Units: (people*Week)/Week

(189) "resetting engineers for year -1 testing revised"=
IF THEN ELSE (launch=1,"annual engineers allocated to year -1 testing revised parts"
/TIME STEP,0)
Units: (people*Week)/Week

(190) resetting engineers for year 0 design=
IF THEN ELSE (launch=1,annual engineers allocated to year 0 design/TIME STEP,0)
Units: (people*Week)/Week

(191) resetting engineers for year 0 testing prototyped=
IF THEN ELSE (launch=1,annual engineers allocated to year 0 testing prototyped parts
/TIME STEP,0)

Units: (people*Week)/Week

(192) resetting engineers for year 0 testing revised=
IF THEN ELSE (launch=1,annual engineers allocated to year 0 testing revised parts /TIME STEP,0)
Units: (people*Week)/Week

(193) resetting year 0 cancelled parts=
IF THEN ELSE (launch=1,year 0 work canceled or postponed/TIME STEP,0)
Units: part/Week

(194) revising parts for year 0=
IF THEN ELSE (launch=1,0,MIN(year 0 parts used in build or test/minimum time to
revise part,addressing problems for year 0*parts per problem))
Units: parts/Week

(195) "revising year -1 parts"=
IF THEN ELSE (launch=1,0,MIN("year -1 tested parts"/minimum time to revise part, "addressing problems in year -1 prototyped parts"*parts per problem))
Units: parts/Week

(196) SAVEPER = 52
Units: Week

(197) special allocation to testing=0
Units: people

(198) step in total engineers=0
Units: people

(199) "step in year -1 new parts"=0
Units: parts

(200) sum of allocated engineers=
"engineers allocated to testing year -1 prototyped parts"+"engineers allocated to testing year -1 revised parts"+"engineers allocated to year -1 design"+"engineers allocated to year -1 problem-solving"+engineers allocated to year 0 design+"engineers allocated to year 0 problem-solving"+engineers allocated to year 0 testing prototyped parts+engineers allocated to year 0 testing revised parts
Units: people

(201) Switch to postpone rather than cancel unfinished work=0
Units: dimensionless

(202) TestAsDelay switch=0
Units: dimensionless

(203) testing accuracy fraction=1
Units: dimensionless

(204) testing productivity=10
Units: part/(Week*person)

(205) testing prototyped parts for year 0=
(IF THEN ELSE (launch=1,0,using prototyped parts in build+MIN(engineers allocated to year 0 testing prototyped parts*testing productivity,year 0 prototyped parts/minimum time to test)))*(1-TestAsDelay switch)+TestAsDelay switch*(IF THEN ELSE(launch=1,0,year 0 prototyped parts/minimum time to test))
Units: parts/Week

(206) testing revised parts for year 0=
(IF THEN ELSE(launch=1,0, using revised parts in build+MIN(engineers allocated to year 0 testing revised parts*testing productivity,year 0 revised parts/minimum time to test)))*(1- TestAsDelay switch)+TestAsDelay switch*(IF THEN ELSE (launch=1,0,year 0 revised parts/minimum time to test))
Units: parts/Week

(207) "testing year -1 prototyped parts"=
(1-TestAsDelay switch)*(IF THEN ELSE(launch=1,0,MIN(testing productivity* "engineers allocated to testing year -1 prototyped parts","year -1 prototyped parts" /minimum time to test)))+TestAsDelay switch*(IF THEN ELSE(launch=1,0,"year -1 prototyped parts"/minimum time to test))
Units: parts/Week

(208) "testing year -1 revised parts"=
(1-TestAsDelay switch)*(IF THEN ELSE(launch=1,0,MIN(testing productivity* "engineers allocated to testing year -1 revised parts","year -1 revised parts" /minimum time to test)))+TestAsDelay switch*(IF THEN ELSE (launch=1,0,"year -1 revised parts"/minimum time to test))
Units: parts/Week

(209) time before build 1=
time before launch-weeks between build 1 and launch
Units: weeks

(210) time before build 2=
time before launch-weeks between build 2 and launch
Units: weeks

(211) time before launch=
IF THEN ELSE (Time<0, -(MODULO(Time,weeks in a year)),weeks in a year-(MODULO (Time,weeks in a year)))
Units: Week

(212) time engineer allocated=1
Units: Week/Week

(213) TIME STEP = 0.0625
Units: Week

(214) time to build=0.007
Units: Week

(215) TimeToCancelOrPostpone=26
Units: weeks

(216) total desired engineers to testing=
"desired engineers allocated to testing year -1 prototyped parts"+"desired engineers allocated to testing year -1 revised parts"+desired engineers allocated to testing year 0 prototyped parts+desired engineers allocated to testing year 0 revised parts
Units: people

(217) total engineers=
base value of total engineers+1*(STEP(step in total engineers,-1)+pulse in total engineers*PULSE(-1,52))
Units: people

(218) total engineers after special allocation= INTEG (
making special allocation,0)
Units: people

(219) total engineers before special allocation= INTEG (
-making special allocation,total engineers)
Units: people

(220) total problems for model year launched= INTEG (
changing in total probs-changing out total probs,0)
Units: problems

(221) "total year -1 parts to be tested"=
"year -1 prototyped parts"+"year -1 revised parts"
Units: parts

(222) "total year -1 unsolved problems"=
"unIDd problems in year -1 prototyped parts"+"unIDd problems in year -1 revised parts"
+"year -1 IDd problems"
Units: problems

(223) total year 0 parts to be tested=
year 0 prototyped parts+year 0 revised parts
Units: parts

(224) total year 0 unsolved problems=
year 0 unIDd problems in prototyped parts+year 0 unIDd problems in revised parts
+year 0 IDd problems
Units: problems

(225) unID problems per revised part for year 0=
ZIDZ(year 0 unIDd problems in revised parts,year 0 revised parts)
Units: problems/part

(226) "unIDd problems in year -1 prototyped parts"= INTEG (
"creating unIDd problems in year -1 prototyped parts"-"discovering problems in year -1 prototyped parts"-"changing out unIDd problems in pp for year -1",0)
Units: problems

(227) "unIDd problems in year -1 revised parts"= INTEG (
"creating unIDd problems in year -1 revised parts"-"discovering problems in year -1 revised parts"-"changing out unIDd problems in rp for year -1",0)

Units: problems

(228) unIDd problems per prototyped part for year 0=
ZIDZ(year 0 unIDd problems in prototyped parts,year 0 prototyped parts)
Units: problems/part

(229) using prototyped parts in build=
IF THEN ELSE (build 1 switch=1 :OR: build 2 switch=1,year 0 prototyped parts
/TIME STEP,0)
Units: part/Week

(230) using revised parts in build=
IF THEN ELSE (build 1 switch=1 :OR: build 2 switch=1,year 0 revised parts
/TIME STEP,0)
Units: part/Week

(231) weeks between build 1 and launch=34
Units: Week

(232) weeks between build 2 and launch=20
Units: weeks

(233) weeks in a year=52
Units: weeks

(234) "year -1 cumulative solved problems"= INTEG (
"accumulating year -1 solved problems"-"changing out cumulative solved problems for
year -1",0)
Units: problems

(235) "year -1 IDd problems"= INTEG (
+"IDing more problems for year -1"+"IDing problems in year -1 prototyped parts"
-"addressing problems in year -1 prototyped parts"-"changing out IDd problems for year -1",0)
Units: problems

(236) "year -1 new parts to design"= INTEG (
postponing parts+"changing in new parts for year -1"-"changing out new parts for year -1"
-"prototyping year -1 parts"-"canceling year -1 work",new parts for next model year)
Units: parts

(237) "year -1 parts prototyped plus"=
"year -1 prototyped parts"+"year -1 revised parts"+"year -1 tested parts"
Units: parts

(238) "year -1 problems per part prototyped"=
0.1+"pulse in year -1 problems per part"*PULSE(103,52)
Units: problem/part

(239) "year -1 prototyped parts"= INTEG (
"prototyping year -1 parts"-"testing year -1 prototyped parts"-"changing out parts
prototyped for year -1",0)
Units: parts

(240) "year -1 revised parts"= INTEG (
"revising year -1 parts"-"testing year -1 revised parts"-"changing out rp for year -1",0)
Units: parts

(241) "year -1 tested parts"= INTEG (
+"testing year -1 prototyped parts"+"testing year -1 revised parts"-"revising year -1 parts"
-"changing out tp for year -1",0)
Units: parts

(242) "year -1 work canceled"= INTEG (
"canceling year -1 work"-"changing out year -1 canceled work",0)
Units: parts

(243) year 0 cumulative solved problems= INTEG (
accumulating year 0 solved problems+changing in year 0 cumulative solved problems
-changing out year 0 cumulative solved problems,0)
Units: problems

(244) year 0 IDd problems= INTEG (
IDing problems for year 0+IDing more problems for year 0+changing in IDd problems for
year 0-addressing problems for year 0-changing out IDd problems for year 0,0)
Units: problems

(245) year 0 new parts to design= INTEG (
changing in new parts for year 0-changing out new parts for year 0-creating prototype
parts for year 0-canceling or postponing year 0 work,initial year 0 parts to design)
Units: parts

(246) year 0 parts used in build or test= INTEG (
+testing prototyped parts for year 0+testing revised parts for year 0+changing in tested
parts for year 0-revising parts for year 0-changing out tested parts for year 0,0)
Units: parts

(247) year 0 problems per part prototyped=
0.3+pulse in Year 0 problems per part*PULSE(103,52)
Units: problem/part

(248) year 0 prototyped parts= INTEG (
creating prototype parts for year 0+changing in prototyped parts for year 0
-testing prototyped parts for year 0-changing out prototyped for year 0,
initial year 0 prototyped parts)
Units: parts

(249) year 0 revised parts= INTEG (
revising parts for year 0+changing in revised parts for year 0-testing revised parts for
year 0-changing out revised parts for year 0,0)
Units: parts

(250) year 0 unIDd problems in prototyped parts= INTEG (
creating problems in prototype parts for year 0+changing in unIDd problems in
prototyped parts for year 0-discovering problems in prototype parts for year 0-changing out
unIDd problems in prototyped parts for year 0,0)
Units: problems

(251) year 0 unIDd problems in revised parts= INTEG (
+creating problems in revised parts for year 0+changing in unIDd problems in
revised parts for year 0-discovering problems in year 0 revised parts-changing out unID
problems in revised parts for year 0,0)
Units: problems

(252) year 0 work canceled or postponed= INTEG (
canceling or postponing year 0 work-resetting year 0 cancelled parts,0)
Units: parts