D-4685

# Sustaining Process Improvement in Product Development: The Dynamics of Part and Print Mismatches

by

Andrew P. Jones

A.B., Engineering Sciences with Environmental Studies, Dartmouth College, 1990

Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Technology and Policy
at the
Massachusetts Institute of Technology
June 1997



Signature of Author _____
Technology and Policy Program
May 1997

Certified by _____
Nelson P. Repenning
Assistant Professor of Management
Thesis Supervisor

Accepted by _____
Richard de Neufville
Technology and Policy Program

Accepted by _____
Joseph M. Sussman
Chairman, Departmental Committee on Graduate Studies

**Sustaining Process Improvement in Product Development:**
**The Dynamics of Part and Print Mismatches**

by
Andrew P. Jones

Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Technology and Policy

June 1997

## Abstract

Despite access to well-grounded methodologies and advanced simulation technologies, some manufacturing firms are having difficulty improving their product development processes. I report the results from a case study of one such firm and use the system dynamics methodology to develop a theory of what causes low quality in product development. At the theory's core, defects in designs create mismatches between parts as manufactured and the blueprints or "prints" on file in design engineering, which, in turn, create more defects. I develop a formal mathematical model to test the theory. Results show that the theory can explain high defect-creation and late defect-discovery observed in product development, suggesting that a methodology for operations and advanced design-testing technologies may be insufficient to meet improvement goals. Model runs further illustrate the system's counterintuitive behavior and provide insights into policies that will drive improvement.

Thesis supervisor: Nelson P. Repenning
Title: Assistant Professor of Management

## Acknowledgments

Thanks first to Mother and Dad for their support and teaching. My mother excels at teaching to think, and my father excels at teaching to do; I hope this paper and the research/intervention project it documents combines what I have learned from them. Thanks to my brother, Cador, and future sister-in-law, Margaret Price, for their welcoming hospitality and support when I was frazzled or Anne Fitten was out of town, and to my brother, Nathan, for giving me perspective.

Nelson Repenning was a significant collaborator in this work. He advanced it with his ideas, his energy, and, thankfully, his substantial investment in my system dynamics education. Besides an excellent teacher and thesis advisor he has been a dear friend, which was particularly welcome the fifth time we got stuck in a Midwestern airport.

Thanks to other professors: John Sterman for his leadership in the System Dynamics Group and the table function; Jim Hines for teaching me about science when I least expected it; John Ehrenfeld for his patient, principled guidance; Ali Mashayekhi for his system dynamics coaching; and Dana Meadows for sparking my interest in system dynamics with her inspiring example of its use.

Thanks to my MIT colleagues. Ed Anderson, Babette Wils, Martin Grossman, Hank Taylor, Rogelio Oliva, Scott Rockart, Omar Khan, Jeff Loiter, José Pacheco, Linda Booth Sweeney, the Uroppers and Tom Fiddaman taught me and shared the journey. Particular colleague thanks goes to Liz Krahmer for her editing and to Kevin Agatstein, who wisely quoted Jay Forrester to break me out of my worst model-analysis slump.

Thanks to team Hogtalk -- John Shibley, Marty Castleberg, Peter Senge, Lisa Forthofer, Don Kieffer, and all my patient teachers at "LaunchTech" -- for bridging academia with practical change.

Thanks to Tom Barrow, Angela Lipinski, Ginny Mumm, Nina Kruschwitz, Jean MacDonald, Kathy Sullivan, Stephen Buckley, Nan Lux and the other E60 folks for their office-life camaraderie when everything started looking too much like a Dilbert cartoon.

Finally, I am deeply grateful to Anne Fitten Glenn, my fiancée. Writing my thesis walked me to the edge of what I knew, where I could gaze over the fence at all I didn't. Anne Fitten's love afforded me a long, slow, wonderfully humbling gaze. For this, and so much more, I am dedicating this thesis to her.

## Table of Contents

# Introduction

## Process Improvement in Product Development

How can manufacturing industries improve? How can they create less pollution, operate more safely, and make high-quality products that satisfy customers and ensure a healthy economy?

One can target the ways that an organization **manufactures** products -- for example, the machines, the work schedules, the accident-response procedures, and the materials recycling plans. However, an alternative approach is to focus upstream in the process on how the organization **designs** products in the first place. Locking the firm into high-quality designs could prevent the need for reactive measures downstream in the manufacturing area. Such a design-focused approach would target the "product development" function of a manufacturing firm. This paper will focus on strategies for improving product development and, thus, help drive improvement in manufacturing industries.

## Guidance from the Literature

Total quality management (TQM), which focuses on shifting efforts from defect correction to defect prevention, can be an effective tool for driving quality improvement in many manufacturing settings (Deming 1986, Easton and Jarrell 1995, Shiba et al. 1993). The U.S. General Accounting Office found that quality improved substantially in twenty finalists from the first two years of a major quality award competition (GAO 1991).

The TQM literature suggests that product development should focus on preventing defects (a defect is a flaw in a design that could cause quality, safety, cost, or environmental problems). Adler (1995) recommends employing "coordination mechanisms" to improve the interface between design and manufacturing. Such mechanisms ensure that, as early and accurately as possible, designers know how their designs will be manufactured. For example, a design engineer creating a new mechanical steel part will know the size and shape of any adjacent or "mating" parts, how well the new part is likely to fit with these parts, how accurately a stamping machine can cut the part (e.g., whether or not the range of a part's width be several tenths of an inch or several thousandths of an inch) and how easy the part will assemble with other parts.

Particular coordination mechanisms or policies, which overlap extensively, include:

- *Concurrent Engineering*. Design engineers and manufacturing people collaborate before the completion of the design (Carter and Baker 1992, Clark and Fujimoto 1991, Nevins and Whitney 1989).
- *Computer-Aided Design and Computer-Aided Manufacturing* (CAD/CAM). High-technology computer models of parts simulate manufacturing and performance testing to prevent defect creation (Bedworth et al. 1991, Engelke 1987, Hawkes 1988).
- *Design for Manufacturability* (DFM). Design engineers emphasize whether a design can be produced efficiently, effectively, and safely as a design consideration (Ulrich and Eppinger 1994, Whitney 1988, Dean and Susman 1989).

In summary, the literature recommends that product development engineers use information about how designs will be manufactured to avoid creating defects early in the product development cycle. Thus, the leading thinkers and practitioners in the area have articulated a vision of how product development should work, and, in particular, the new technologies it will employ.

However, many firms have struggled to successfully apply TQM to product development (Thomas and Napolitan 1994, Jessen 1992). The lack of success is puzzling. The vision is laid out; it seems that a company could just purchase CAD/CAM hardware and software, hire a consultant in concurrent engineering, run the training sessions and thus improve product development. So, why isn't it working?

To understand the barriers to improving product development, I studied the experiences of one company confronting this challenge and used the insights gained to develop a theory that addresses the problem outlined above.

## Research Method

Professor Nelson Repenning and I studied a manufacturing company, fictionalized as "LaunchTech." LaunchTech manufactures a mechanical product with a high premium on both functional and cosmetic quality. Two characteristics of the company are particularly important to this thesis. First, the company makes annual changes to the product, i.e., every "model year" has new features. Secondly, the company begins manufacturing the new model year at "launch," which occurs in the same month of every year.

The research was conducted by telephone and through more than 13 trips to the company's headquarters and two manufacturing plants. We interviewed 36 different individuals from various areas of the organization, including Engineering Management (4 people), Design Engineering (3), Assembly Plant Resident Engineering (1), Manufacturing Management (5), Manufacturing Supervisors (3), Manufacturing Engineering (8), Manufacturing Operators (5), Project Management (1), Quality/Warranty/Customer Service (5), and Purchasing (1).

We collected data primarily via interviews, which we performed in a semi-structured fashion. Subjects were asked a number of standard questions. Given the nature of the research, the interviewees were not forced to stay with the standard questions. If the interviewers felt that they were pursuing a profitable avenue, the interviewee was allowed to continue in that direction. A secondary data collection method consisted of attending meetings. Notes were taken at each interview or meeting, and most interviews and many meetings were recorded. The recorded interviews and meetings were transcribed from notes and tapes, and quotes were taken from the transcripts. Several individuals were subsequently contacted to clarify issues and elaborate on particular points. Quantitative data were obtained from the company.

## Case Study - Product Development at LaunchTech

The following case study briefly explains how LaunchTech has traditionally developed new products, and what they have done to improve this process.

### Sequential Product Development

Since its founding, product development (the overall process of developing a design, from concept to design engineering to preparation for launch) at LaunchTech, like many other manufacturing companies, used a "sequential engineering" or "over the wall" process, meaning the design work and manufacturing work followed each other sequentially.

*Design*

Design engineering created new part designs for manufacture. To distinguish the upcoming year's product from the current year's, design engineers primarily made incremental changes to existing product designs -- for example, they would change existing parts to a different shape, size, or color. Every year, approximately 10% of the designs in production changed.

To produce high-quality designs, engineers needed two pieces of information. First, they needed to know whether manufacturing would be able to produce their new designs. For example, could the production processes bend metal to a certain curve, cut a part a specific width, or assemble two parts reliably? Second, because most new parts would have to fit with multiple existing parts, design engineers needed to know the shape and size of these parts as actually manufactured.

Design engineers could get the information from the manufacturing area via personal communication or printed documentation. LaunchTech was a small, informal business where most people knew each other, so people across LaunchTech relied primarily on personal communication. One manager explains,

> Getting things done in this organization. . . is more relationships than systems. It is knowing who to talk to and having some working rapport with those individuals as opposed to saying that I could put in a work order and hand it to the supervisor, and it will happen.

But in its sequential engineering process, the organization allocated limited time for manufacturing and design people to collaborate early in the product development process,

so personal communication did not always deliver the required information about manufacturing capabilities and processes to design engineering.

Thus design engineers often relied on the second source of information, printed documentation. When designing a new version of an old part, an engineer would consult the blueprint (or print) of the old part and the prints of all the connecting or "mating" parts on file. Additionally, she could make actual parts and assemble them to check that they fit properly, in a "mock-up build." However, due to processes explained later, approximately 25% of the prints of parts on file in design engineering did not match the actual parts as produced in manufacturing, which limited the engineer's ability to get reliable information about manufacturing capabilities and processes.

More specifically, mismatches disrupted the effectiveness of design engineering in two ways. First, design engineers sometimes designed parts that would fit with mating parts as shown in the prints, but would not fit when actually produced. One manufacturing engineer explains,

> Design engineering will design a part to an existing drawing -- actually, we [in manufacturing] are making that part not to an existing drawing but to a revised drawing that design doesn't know about. They send the piece down [to manufacturing], we make it, and it doesn't fit. Or it makes unwanted contact, which causes cosmetic damage.

Secondly, mismatches reduced the ability of engineers to test the manufacturability of the design with mock-up builds or computer simulation tools. One design engineer said,

> We depend on [testing in manufacturing] to finalize designs. We don't trust the database. The computer says it will fit, but we say, 'No, I want to see it fit on the production line.'

Design engineers had imperfect knowledge about manufacturing capabilities and processes, and had difficulty creating designs that were easy to manufacture reliably. The organization would then rely on the manufacturing division to correct problems or "defects" with the designs. One engineering manager said,

> When the design was done, they'd run it to the manufacturing people. We never actually thought about how it would be manufactured.

*Manufacturing*

If a completed design was difficult to manufacture reliably, then it was defective or had a "defect." If a manufacturing engineer discovered the defect, usually he could request that a design engineer change the design and fix the defect. Sometimes, however, the engineer discovered the problem late in the product development process. When launch of the new designs into production was approaching and there was not enough time for a thorough design change, the manufacturing engineer would sometimes change the design of the part so it could be manufactured reliably. By adjusting the product or process design, the engineers would contain the imminent defect so that it did not cause quality problems. However, containing a defect often added a permanent new step to the manufacturing process. One engineer noted,

> Often there are open issues at launch. Audit is asked to come up with a band-aid fix for the line. Once the band-aid is in place, no one goes back to fix the issue. The band-aid becomes the permanent fix and it should not be.

Examples of containing a defect included:

- Manufacturing a piece slightly smaller to ease assembly;
- Sanding or filing a part so that it mates properly with another part;
- Buffing or polishing a mark that a tool die puts on a part;
- Cutting a small metal block to fill a gap between two parts;
- Forcing a part on in assembly.

Over time, a culture developed around the ability to use one's metal-working experience to creatively adjust designs. One manager explained that the company has ". . . always had a reverence for tribal art -- hammer to form, file to fit, paint to match." Indeed, many engineers prided themselves on their resourcefulness. Another manager said, "Containment was fun. It was exciting to be able to say, 'The line is running today because I figured out a way to solve that problem.'"

Once someone in manufacturing adjusted designs for production, he did not always have the time, processes, or incentives to record the changes in the prints. One manufacturing engineer said, "We manipulate the hardware to meet needs for launch, but the documentation is not always complete."

Containment might prevent a defect, but, without adequate documentation, might also create a mismatch between a print and the part as manufactured.

## Focus on Improvement in Product Development

Despite the problems outlined above, the sequential process was relatively effective at producing designs for manufacturing. Indeed, in its first fifty years, LaunchTech built a proud reputation as an innovator in new designs and popular products. Then in the '70s, as the company increased capacity, product quality declined and sales fell. In the '80s, LaunchTech's leaders applied Total Quality Management to manufacturing processes, which improved quality and boosted profits dramatically.

By the late '80s, little improvement potential remained in the manufacturing area, and quality leaders began to focus on other areas of the company. Additionally, earlier quality improvement had driven revenue growth that sparked capacity expansion, and soon LaunchTech needed to create innovative designs that could attract enough customers to buy all the products their new plants would soon produce.

Thus, in the late '80s and early '90s, LaunchTech turned its improvement focus to product development. Consistent with the Total Quality Management approach, which they had successfully applied to manufacturing, LaunchTech's vision for product development focused on preventing defects early in the design process. More specifically, the vision included applying "concurrent engineering" and improving testing in design engineering.

### *Concurrent engineering*

To shift product development from sequential engineering to concurrent engineering, leaders at LaunchTech drafted a new "concurrent methodology." The methodology encouraged design engineers and manufacturing people to collaborate before the completion of the design and to meet certain deadlines in the process.

### *Improved design testing*

The new process encouraged design engineers to discover defects early in the product development cycle by testing designs more extensively. It included: (1) improving the "mock-up builds," in which actual parts are made and assembled to check for proper fit, and (2) using Computer-Aided Design and Computer-Aided Manufacturing (CAD/CAM) to simulate how parts fit together in manufacturing. One leader said,

> To improve product delivery, using modeling and up-front simulation is the only way to go. Every other way -- checking parts on the production line, for example -- is too expensive. The only question is how fast we can get there.

For the past seven years, leaders at LaunchTech have tried to implement their vision for product development. Their core implementation strategy has been to describe how product development should work once improved and encourage the engineers in the design and manufacturing areas to work in that improved mode. More specifically, leaders have distributed printed copies of the "concurrent methodology" handbook, revised the methodology based on employee feedback, and asked engineers to follow its processes. Engineers have formed cross-functional teams to improve communication across areas. Leaders have worked to improve the physical testing of parts in design engineering and the company has invested in CAD/CAM technologies. In 1996, top level managers began checking design projects for their level of completion at fixed dates in the product development cycle.

Today, in 1997, a solid concurrent methodology exists on paper, several projects are meeting deadlines, and some managers report that fewer defects are entering production. But most people at LaunchTech still report minimal benefits from the new approach, and many others accept that much of its potential has not been realized. The number of part and print mismatches remains high, and engineers are not able to prevent many defects early in the process. Manufacturing engineers continue to discover defects late in the process and resort to containing them instead of fixing them. And finally, previously steady improvement in the fraction of products returned by customers for repair has recently leveled out.

## The Problem

The first challenge of is to explain what is creating the low-quality mode of product development at LaunchTech and in many firms. What structures, policies, and interconnections within a company can sustain high defect creation and late defect discovery?

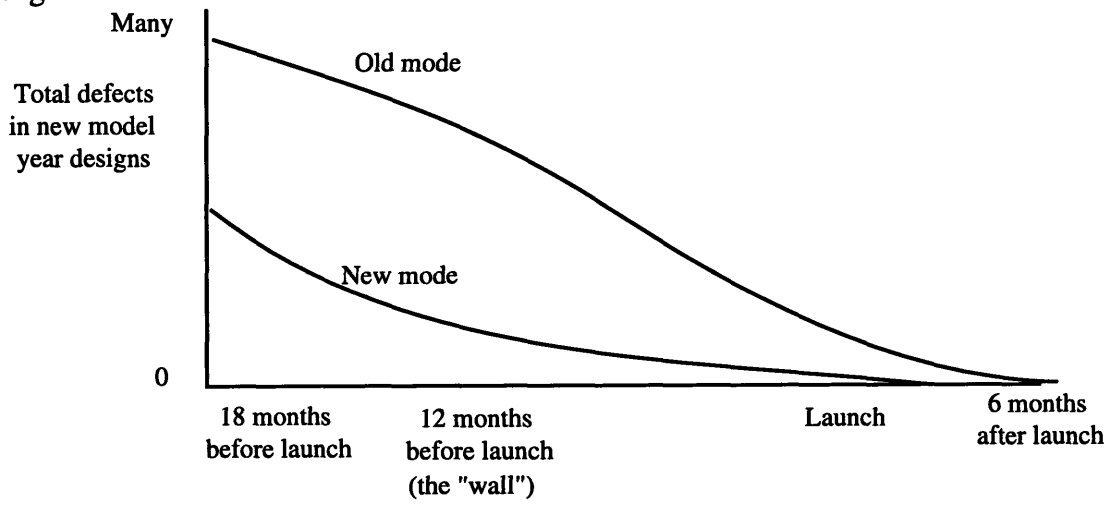
LaunchTech's difficulty implementing its "vision" for product development motivates the second endeavor. Learning about the company's long history of sequential engineering made it clear that LaunchTech was not implementing its improvement policies with a clean product development slate, but, instead, into an organization with a well-established process. For companies such as LaunchTech, the challenge is not describing the improved mode of defect prevention, but creating policies to break free from the old mode and move toward the new; the literature has not addressed strategies to drive the transition.

In summary, the purpose of this thesis is to answer two questions: What creates a low-quality mode for product development? And what policies can drive the transition to the high-quality mode?

### Reference Modes

One way of framing the two questions in a more dynamic context is through reference modes -- graphs that outline the behavior of variables over time.

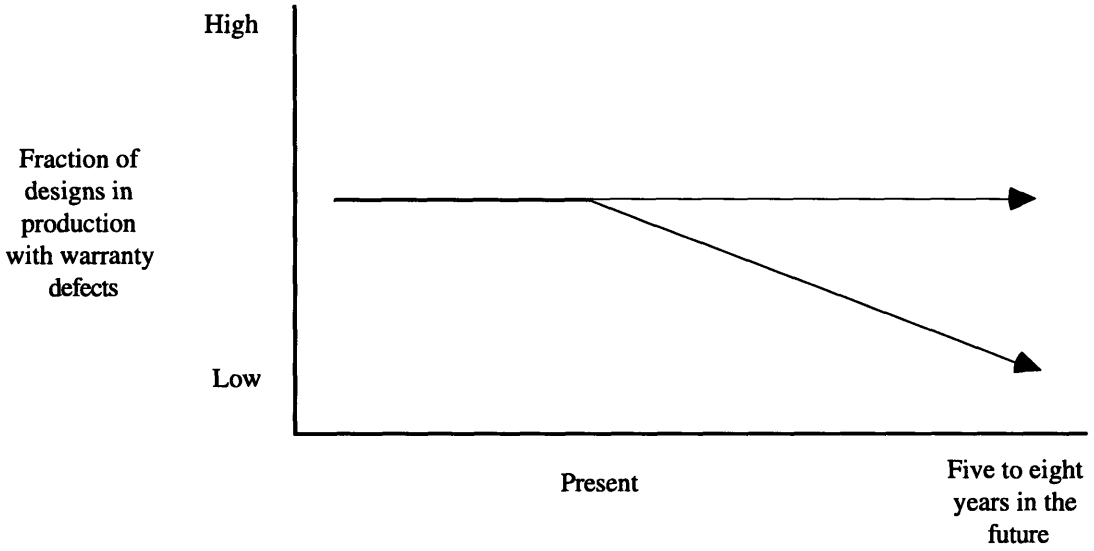
Figure 1: Defect Reference Mode



The reference mode in Figure 1 shows the behavior of *Total defects in new model year designs* for six months as design engineers work to find and fix the defects, and for eighteen months as the manufacturing people work to find and fix them.

The graphs shows two modes of operation: old and new. The top line represents the existing or old mode, in which design engineers introduce many defects into the product design process and discover and fix a small percentage in the first six months. Engineers in manufacturing discover most of the defects, yet some enter production. The bottom line represents the improved mode of operation, with less introduction, more early discovery and fixing, and fewer defects entering production. What creates the behavior in the top line? What policies will drive an organization to shift from operating in the top line mode to the bottom line mode?

Figure 2: Quality Reference Mode




The second reference mode shows a key quality metric, *Fraction of designs in production with warranty defects.* The variable is assumed to be constant at a moderate level in recent years. What policies hold greatest leverage in determining which future scenario will unfold: improvement following the bottom arrow, no improvement following the top arrow, or one of the many scenarios in between?

## The Dynamic Hypothesis

"We've gotta be singing out of the same choir book."
-- LaunchTech manufacturing engineer

The case study section of this paper presents many different dynamics operating in a manufacturing company like LaunchTech: Design engineers introduce defects, and do not discover them early in the process; the organization tends toward informal interaction rather than formal documentation; manufacturing adjusts designs and does not always document the changes; defects enter production; and part and print mismatches disrupt designing. Separately, these factors provide limited insight into the troubling dynamics observed in some firms. But when analyzing the interrelationship of the factors and the way the interrelationships evolve over time, a compelling theory emerges about barriers and opportunities in improving product development.

The next section of the thesis will describe this theory in greater detail, using causal loop diagrams to describe feedback relationships. The core of the theory follows.

As explained in the case study section, mismatches between parts and prints limit design engineers' knowledge about manufacturing processes and capabilities. More specifically, they increase the number of defects that design engineers create -- engineers design parts to fit with mating parts as shown in the prints, but they do not fit with the mating parts as actually produced. Further, mismatches stymie attempts to use simulation tools such as CAD/CAM systems and thus limit the ability of design engineers to find and fix defects early in the product development process. In these two ways, mismatches create defects.

In turn, creating defects create mismatches in three ways. First, defects lead to engineers submitting design change requests, which, with imperfect documentation, create further mismatches. Second, defects enter production and create mismatches as manufacturing machines begin making parts differently from their prints. Third, manufacturing people adjust designs to contain the defects, and, with imperfect documentation, change the part in production away from the print in the design engineering library. Newly created mismatches disrupt the design group again in a vicious cycle.

The feedback structure relating defect creation with mismatch creation helps explain the structures, policies, and interconnections in place in a company that can sustain high defect

creation and late defect discovery, as outlined in the reference modes. The following section presents the structure in greater detail. After that, a formal model will test the hypothesis presented here, plus several hypotheses for policies designed to drive the transition to an improved mode of product development.

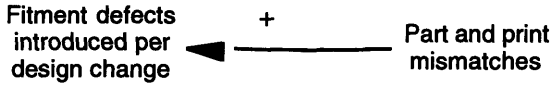
## Analysis Method

This thesis uses the system dynamics methodology for modeling complex systems. Simulation with system dynamics provides several advantages. First, assumptions are made explicit when represented by formal equations. Second, the consequences of assumptions, structures, and policies over time can be examined. System dynamics describes cause and effect relationships with stocks, flows, and feedback loops. Stocks and flows are used to model the flow of work and resources through a project. Information feedback loops are used to model decisions and policies. Time delays such as the one between discovering a defect and the defect being fixed are explicitly identified, as are non-linear relationships (Forrester, 1961).

The feedback structure in the model is captured by a causal loop diagram (Richardson and Pugh, 1981). The following section explains several elements of the language used.

*Causal connections*

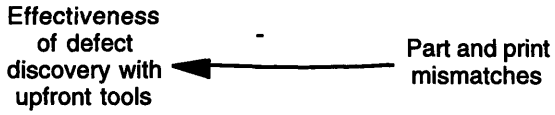
Arrows linking variables indicate that the first variable causes a change in the second variable. The positive sign indicates that the change in the second variable is in the same direction as the first variable.




For example, the link above indicates that, all else being equal, as the number of part and print mismatches goes up, the number of fitment defects (defects where two parts do not fit together) introduced per design change will go up, too. Mismatches go down, then defects go down.

$$\frac{d(\text{Fitment defects introduced per design change})}{d(\text{Part and print mismatches})} > 0$$

The arrow diagram is equivalent to the equation listed above.




The negative sign on the second arrow indicates change in the opposite direction. For example, the link shown above indicates that, all else being equal, as the number of part and print mismatches goes up, the effectiveness of defect discovery with upfront tools goes down. Mismatches down, then effectiveness up.

$$\frac{d(\text{Effectiveness of defect discovery with upfront tools})}{d(\text{Part and print mismatches})} < 0$$

The arrow diagram is equivalent to the equation listed above.

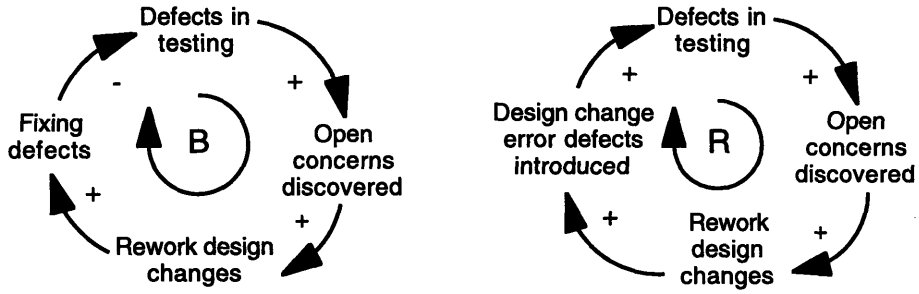
*Stocks and flows*


The clouds at either end of the diagram represent the boundaries of the model. The box, marked *Stock* represents a stock or accumulation, while the two valve-like icons marked *Inflow rate* and *Outflow rate* represent flow regulators that increase and decrease the level of the stock, respectively.

$$\text{Stock} = \int(\text{Inflow rate} - \text{Outflow rate})dt + \text{Stock}_{\text{init}}$$

The stock and flow diagram is equivalent to the equation listed above.

*Loops*



A "B" in a diagram indicates a balancing feedback loop, which seeks equilibrium. An "R" in a diagram indicates a reinforcing feedback loop, which amplifies change. The diagram above shows two examples.

### Critical assumptions

The dynamic hypothesis does not account for allocation of resources such as engineers' time and access to testing tools. For example, spending time on one task does not restrict the amount of time available for another task. Further, it is assumed that resources are not constrained.

### Introduction to the Diagram

The purpose of this section is explain the dynamic hypothesis more precisely through the use of a hybrid stock-and-flow and causal-loop diagram. The diagram will be built section by section.

The heart of the theory presented here is how, in a circular fashion, part and print mismatches effect defect creation, defect testing, and launching defects into production, which in turn effects the creation of mismatches. This thesis will start with part and print mismatches, move to explain their sources, and then close the feedback loop by explaining their effects.

### Part and print mismatches

*Part and print mismatches* are designs for which the specifications in the print do not match the part as manufactured or assembled in production. Examples include:

- a metal piece that in the print has a square corner, but when produced on the manufacturing line, an operator must round the corner with a belt sander so it fits with adjacent parts;
- a manufactured bracket that in the print is 1/4 inch width, but because of a worn tool or variations in the machine operations, the part as manufactured is often closer to 3/8 inch width;
- a thin, metal, curved part in its print has sides that run absolutely parallel, but as manufactured, has sides that bend out slightly; and,
- on the prints, the holes in a bracket and on the frame line up perfectly for a bolt. As manufactured, the frame has bent slightly after being heated, and the holes do not line up, requiring extra effort for assembly.

Just as there would be in any manufacturing organization, there is a positive number of *Part and print mismatches* at LaunchTech at present. Estimates of the percentage of designs in production that have part/print mismatches vary, with a resident engineer guessing 50%, a design engineer guessing 50%, a manufacturing engineer guessing 25-30%, and a manager reporting that "while some would guess less than 50%," a recent check found 20-30% parts not to print.

Figure 3: Part and print mismatches

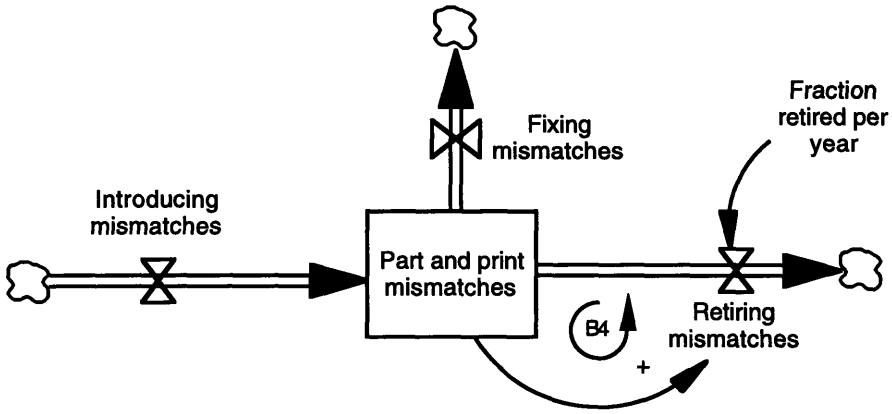



Figure 3 shows the processes that regulate the number of mismatches. The flow *Introducing mismatches* increases the stock, *Part and print mismatches*. Two outflows, *Fixing mismatches* and *Retiring mismatches*, decrease the stock.

*Retiring mismatches*

The first outflow captures the way that designers discontinue certain parts every year. When designers discontinue a part that is also a mismatch, that mismatch leaves production. Loop B4 in Figure 3 governs the retiring of mismatches, and describes how retiring will increase as mismatches increase, which will then feed back to reduce the number of mismatches. The balancing loop works similarly in the opposite direction.

*Fixing mismatches*

The stock of mismatches can also be reduced by fixing, either by changing a part to match a print or changing a print to match a part. For more discussion of fixing mismatches, see the "Fix Mismatches" policy test in the model runs section.

*Introducing mismatches*

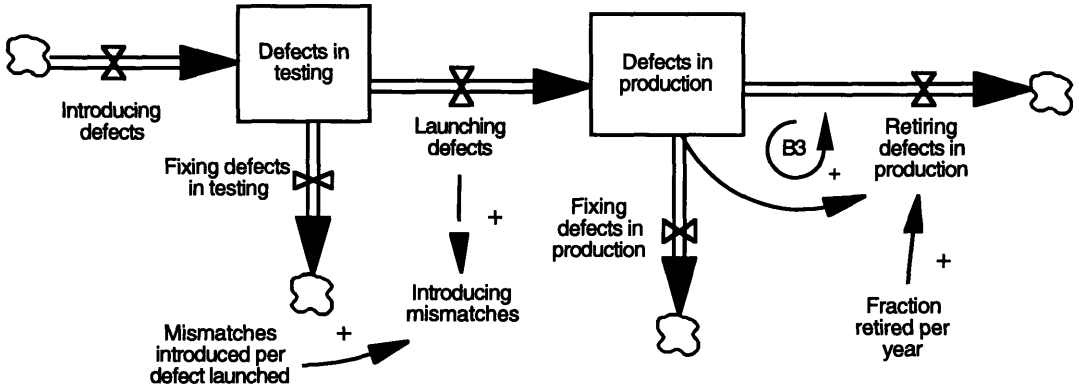
Figure 3 shows one inflow to the stock of mismatches, *Introducing defects*. Three main sources drive the inflow:

1. defects entering production;
2. the containing or adjusting of designs away from their printed specifications in order to fit parts together without documenting the change; and
3. lapses in documentation of other design changes.

The next three sections will explain each source of mismatches.

## The Defect Source of Mismatches

Figure 4: Defects




In Figure 4, *Launching defects* is one source of *Introducing mismatches*. Additionally, the number of *Mismatches introduced per defect launched* drives mismatch introduction. Before explaining the details of this connection, however, this section will describe the context of launching defects.

Launching defects is part of a wider process outlined in Figure 4. Approximately 18 months before "launch," which is the first month of production for a new "model year" of products, engineers introduce new designs into testing. Some designs will have defects, thus driving the variable *Introducing defects*. In the 18 months before launch, engineers work to discover any defects in the designs, as they sit in the stock *Defects in testing*. First the testing happens in design engineering, where engineers can see how parts fit on mock-up or prototype products, test the designs with computer tools such as CAD/CAM, or

simply check the designs by examining relevant blueprints. Then manufacturing people test the designs in the manufacturing plant by actually building the products on the production line, to discover defects. People at LaunchTech call such tests "builds." If workers discover defects, then engineers fix them, driving the variable *Fixing defects in testing*. Undiscovered defects get launched into production through the flow *Launching defects*. Once in the stock, *Designs in production*, engineers can fix them, or retire them, or they just remain in the stock.

Loop B3 governs the *Retiring of defects in production*. The more defects there are, with a fixed *Fraction retired per year*, then the more are retired. More retiring decreases the number of *Defects in production*. The loop behaves much like a drain, with the outflow higher when the level is higher, and the outflow lower when the level is lower.

More explanation about how *Launching defects* creates mismatches is necessary, however. Defects that are launched originated from designers having imperfect knowledge of manufacturing processes and capabilities, or from designers making errors. Thus, when a design created under such circumstances enters production, there is some chance that the manufacturing process will produce parts that differ from the parts as created in the blueprint by the design engineers.

Examples of manufacturing producing parts differently from the designs (particularly without extensive collaboration between manufacturing engineers and design engineers) include: Sides of curved thin metal parts splaying out when cut, large metal parts bending when heated, wire routings running shorter or longer distances, and a tool die (which create many parts) drifting from its original specifications and producing parts a different size.

Many of the defects that produce mismatches may not cause serious warranty problems in the field. Instead, they can be low-priority problems that are not fixed at a root cause level and thus remain as part of the manufacturing system. As one manufacturing engineer said,

> Some issues are handled by manufacturing engineering or the new products group, some are handled by resident or continuing engineering, and some drop into oblivion and no one does anything about them. They are swept under the rug of past sins.

### The Containment Source of Mismatches

The second source of mismatches is the active adjustment of designs in manufacturing. When an engineer discovers a defect late in the testing process or in production, she has a difficult decision to make. She would like to investigate the root cause of the problem, write an order slip requesting design engineering to change the design, and wait for the new print. But because of time constraints or different priorities, often a second possibility may be more attractive -- contain the problem by adjusting the manufacturing process so that the production line works well without the problem being solved fundamentally.

One manufacturing engineer described an example,

> The part just doesn't fit unless we belt sand a corner off. Everyone on the floor knows we have to do it for the part to fit.

Containing defects is critically important to high-quality sequential engineering work. Once the designs are on the manufacturing side of "the wall," adjustments to the design can often improve quality substantially. Problems arise when someone contains a defect by adding a step to the manufacturing process or changing the size of a part, and design engineering does not hear of the change.

Figure 5: Contained defects

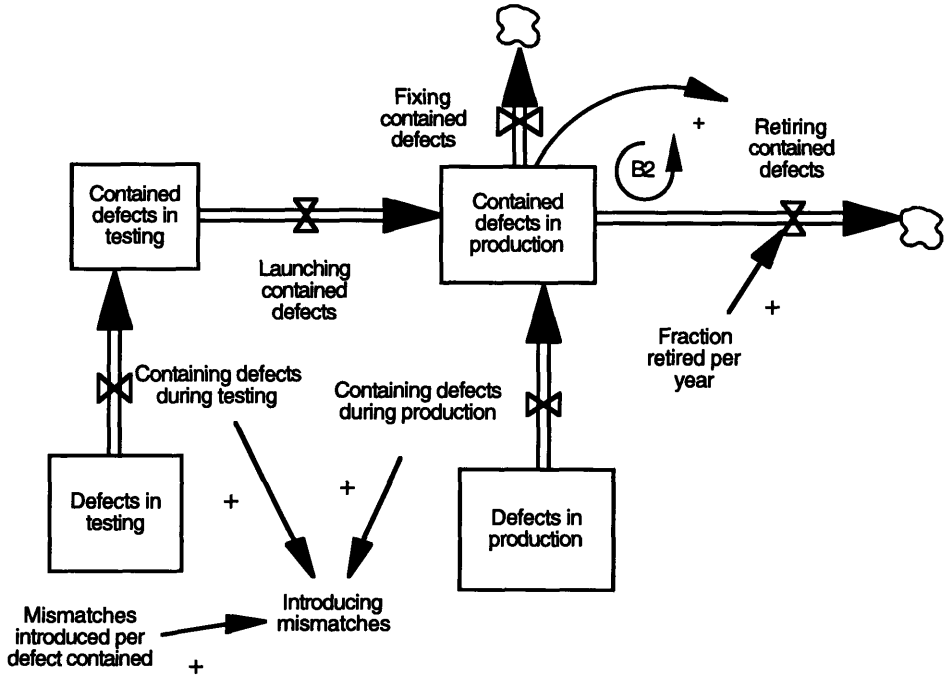


Figure 5 shows a stock-and-flow representation of containment that builds on the diagram in Figure 4. Contained defects get created by pulling defects through the flows *Containing defects during testing* and *Containing defects during production.* Much like Defects, Contained defects get launched, fixed and retired. Also like *Defects in production*, *Contained defects in production* also build up the manufacturing system.

Containment without adequate documentation can create a mismatch between a part and a print. The two upward-pointing flows in Figure 5, *Containing defects during testing* and *Containing defects during production*, drive *Introducing mismatches.*

Documentation -- captured in the diagram by *Mismatches introduced per defect contained* -- may seem to be a low priority in a high-intensity work environment; it may seem unnecessary to change the print because a part is being manufactured slightly smaller, for example. One manufacturing engineer explained that the decision to not change the print to reflect the change in how the part is being made may be one of time pressure or different priorities. He said,

> It takes only five seconds on a belt sander to take a corner off a part that doesn't fit, or three hours working with a draftsman to change the print, and sometimes we just don't have time to do it.

### The Documentation Source of Mismatches

A final source is not adequately documenting design changes, either by not completing the documentation of a design change on the print or by introducing errors to the prints through carelessness.

One reason that a change may not be adequately documented is that design engineers ignore documentation and move on to other work. One resident engineer, who works in manufacturing, said,

> Sometimes we request the change notice but it is never handled by [design engineering]. The manager of the design engineer has already moved him to work on the next model year instead of wrapping up this year. If the problem isn't a show-stopper, it isn't fixed. . . . Now I feel as though I'm following that guy around with a broom and a dustpan, trying to clean up the mess.

Design engineering finds the time to make design changes, but sometimes does not document the changes fully by changing the prints. One engineer said,

There are still change notices for print changes open from past years not as much because of time pressure, but because of differing priorities. Many design engineers think that once you get to launch, you are done. No one wants to handle the issues that come up after launch.

Figure 6: Fixing defects

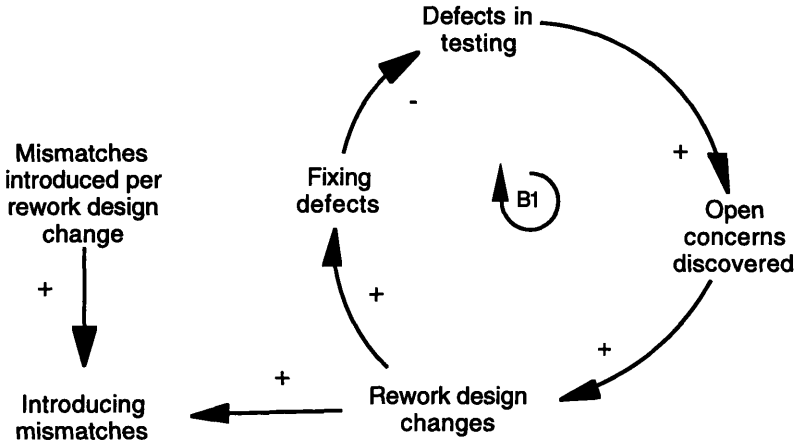
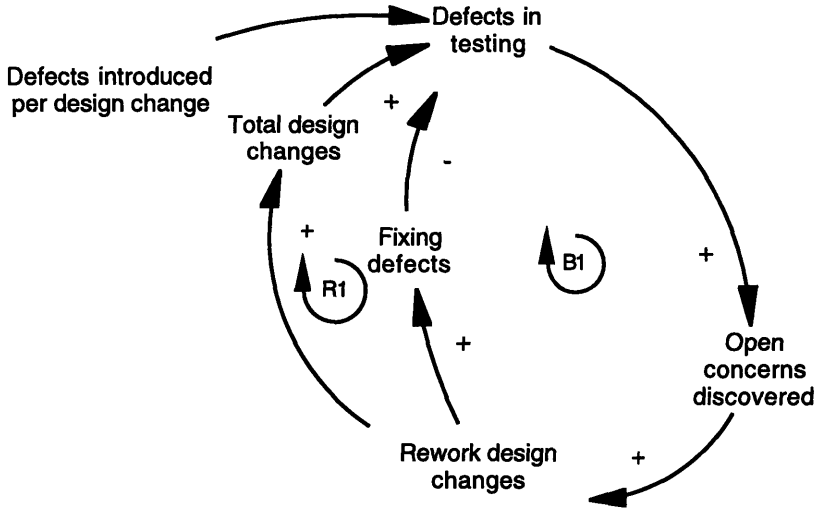



Figure 6 shows a balancing loop that drives fixing of defects and the introducing of mismatches. Starting at the top, the more *Defects in testing* there are, the more defects will be discovered in upfront testing in design engineering, or in the manufacturing builds. Such defects are catalogued as "Open Concerns" that need to be fixed. More *Open concerns discovered* lead to more *Rework design changes* for design engineering to complete. When rework clears up the open concern, then that increases *Fixing defects*, and thus decreases *Defects in testing*. This balancing loop seeks to minimize the number of defects. The loop also drives *Introducing mismatches,* as every time a design change is completed, there is a small chance that the change will not be completed adequately (for all the reasons outlined above), and a mismatch will be introduced. The variable *Mismatches introduced per rework design change* captures the chance of inadequate documentation.

*Closing Loop R1*

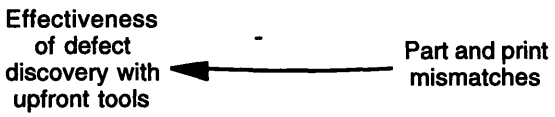
Figure 7: Unanticipated effect of rework design changes




The previous section introduced the balancing loop B1. Figure 7 introduces a second, related loop, R1. Rework design changes leads to fixing defects (and creating mismatches, as we saw earlier). But every time a change notice is introduced, there is a small chance of an unanticipated side effect from correcting a defect -- it may introduce yet another defect to the system. *Rework design changes* increases *Total design changes*, and, along with *Defects introduced per design change*, increases *Defects in testing,* thus closing the reinforcing loop R1. For every step forward in defect correction that B1 makes, the loop R1 will take a smaller step back by introducing new defects.

The previous sections have explained three main sources of mismatches, and begun to outline the feedback loops in place that drive the system under consideration. Now the focus will turn to the effects of mismatches, and the next section will introduce several important feedback loops.

## Effects of Mismatches -- Upfront defect discovery



The first effect of more *Part and print mismatches* is that they reduce the *Effectiveness of defect discovery with upfront tools*. That is, design engineers will have less success in finding defects early in the product delivery process, particularly in the "mock-up" builds and by using CAD/CAM tools.

In a mock-up build, an engineer makes actual parts and assembles them to check that they fit properly. Often, the parts are specially made for testing by following the blueprints for the parts exactly, leading an engineer to confirm that a new part fit well with the adjacent parts. But if parts are not the same as produced as they are shown in the print, then the mock-up test could not have revealed any defects.

CAD/CAM tools, as well, rely on the accuracy of the prints on file and in the computers to discover defects before builds. If a CAD tool has images of parts on the computer screen that are not accurate, then the user of the tool will not be able to discover a potential manufacturing problem before the actual builds later in the process.

The futility of upfront testing breeds additional behavioral dynamics that reduce use of the tools -- if engineers know that upfront testing is ineffective, they will abandon their faith in the value of upfront work. In particular, the organization found it difficult to complete and test high-quality designs before the testing builds on the assembly line. As one engineer notes,

> There was a time when design was almost reverse engineering what worked for engineers on the line. They would try to build the [product], and if it didn't fit, they would change the design. Design engineers would show up here [in manufacturing] with a rucksack full of prototype parts and try to make the design work.

*Closing loop R2*

Figure 8: The upfront discovery loop

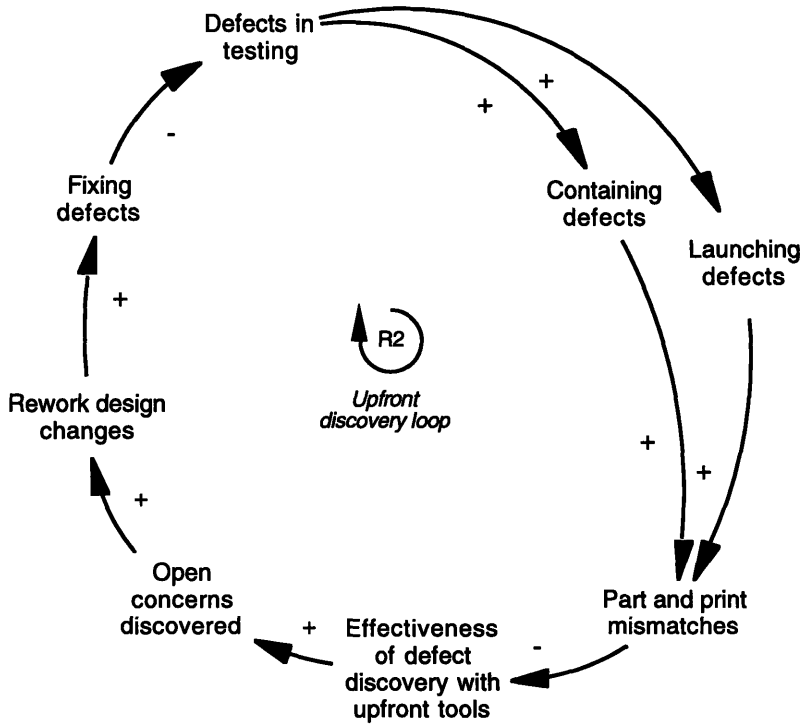
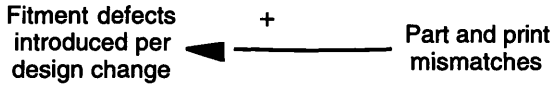



Figure 8 shows a new reinforcing loop, R2 (actually two loops in the diagram, but identified as one). More *Part and print mismatches* in the database decreases the *Effectiveness of defect discovery with upfront tools*, which then decreases the number of *Open concerns discovered.* If engineers are unaware of the defects in the system, they will not submit *Rework design changes*, which will decrease *Fixing defects* and increase the remaining number of *Defects in testing*. More defects in testing means that fewer defects will be discovered and fixed, thus increasing *Containing defects* and *Launching defects*. In this way, more mismatches feed back to create more mismatches.

More important to the policies tested in this thesis, the loop can drive improvement in the opposite direction. Fewer mismatches may increase early defect discovery, leading to more design changes happening earlier in the product delivery process. Design engineers would be clearing up more problems before any of the production builds. Engineers would fix more defects well before launch, so they would contain fewer problems, and launch fewer defects, thus creating fewer mismatches. One purpose of the mathematical model presented later in this document will to test the behavior of the R2 loop outlined here.

**Effects of mismatches -- Fitment defect introduction**




The final effect of mismatches presented here is that *Part and print mismatches* increase the number of *Fitment defects introduced per design change*. Fitment defects are design problems in which parts do not fit together well. A design engineer making a new model year change or a rework change to a part will check the drawings on file to see how the part mates with other parts. If one of the mating parts is different in production than on the print, there is some chance that a new defect will be introduced. For example, if the new part as designed mates flush with a square corner on a mating part, but someone is rounding one corner off the part with a belt sander, than the design change would introduce a new defect.

One manufacturing engineer summed up the dynamic of fitment defect introduction,

> The new production year starts, and the [products] are being made, so the mismatch [created at the past launch] is not really a problem. You move on to new issues for the next year. But a couple years later, when you try to use the prints in the file, there is a mating problem.

One design engineer adds,

> Nine out of ten times when there is something wrong on the line and you dig into it, it is because parts are not to print. [Design engineering] uses prints to come up with new designs. When you come up with a new design and base it on a current part, you use the print. Often times new parts do not mate with existing parts because the existing parts are not to print.

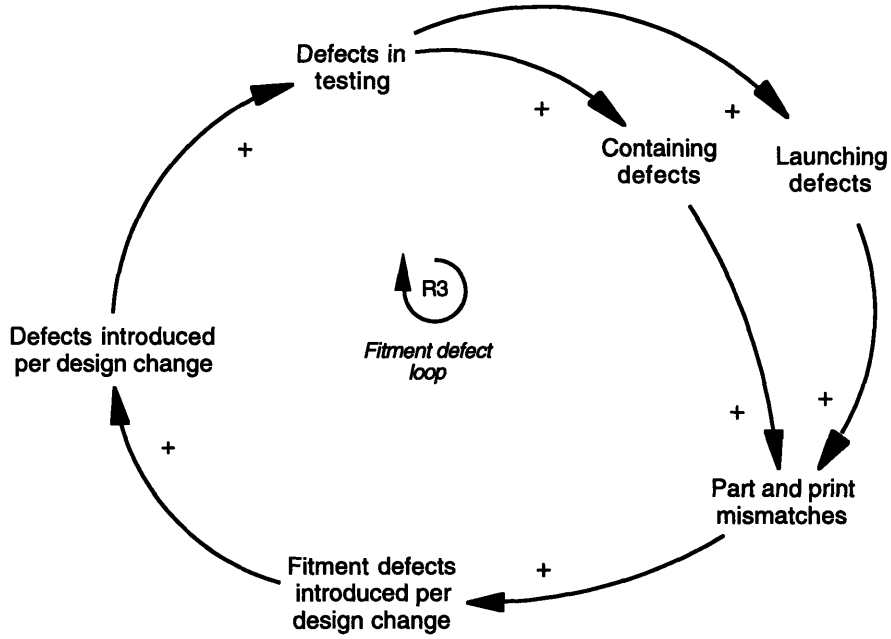
Here is a simple calculation of the possible extent of this dynamic: One engineer made a rough estimate that, while it varies widely, one design change may have approximately four mating part connections. He also estimated that if a mating part was not to print, it had approximately a 25% chance that a defect could be introduced. Taking a conservative estimate of parts not to print of 25% and doing a very rough back-of-the-envelope calculation:

$$1 \text{ design change} \times \frac{4 \text{ mating parts}}{\text{design change}} \times \frac{.25 \text{ mismatches}}{\text{part}} \times \frac{.25 \text{ defects}}{\text{mismatch}} = \frac{.25 \text{ defects}}{\text{design change}}$$

If every design change introduces .25 defects, than the 1,000 or so yearly changes introduced 250 defects. Put another way, every fix has only a 75% "efficiency."

*Closing loop R3*

Figure 9: The Fitment Defect Loop

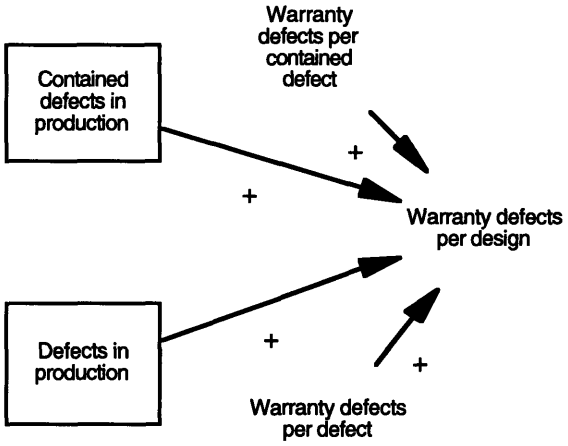



The latest connection closes a final loop, the reinforcing loop R3. Whereas loop R2 reinforced the growth or decline of *Defects* through defect discovery, the loop R3 has a similar effect by introducing new defects.

One example of the loop in operation in the direction of improvement is this: if the number of mismatches drops because of increased documentation or a fixing program, design engineers introduce fewer fitment defects with every design change. In the next model year, there are fewer defects. Later, the organization launchs fewer defects and fewer contained defects into production, both of which cause less of an increase in the number of mismatches, which, given retirement of mismatches, is a net decrease in mismatches. Thus, fewer mismatches feed back to decrease the number of mismatches.

## Product Quality

Figure 10: Product quality



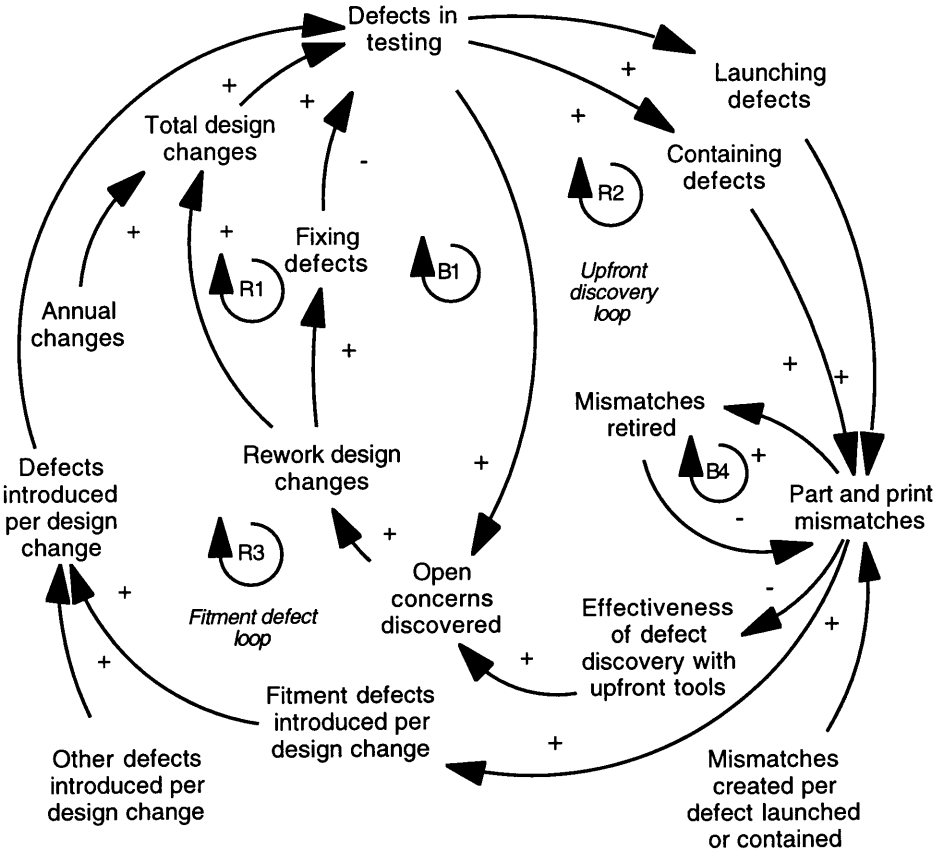

*Defects in production* and *Contained defects in production* together determine *Warranty defects per design.* The effect of *Defects* is obvious -- if parts do not fit together or if a tool die cannot manufacture a part reliably, quality will suffer. The effect from *Contained defects* is more subtle, and relies on the fraction of *Warranty defects introduced per contained defect* -- examples include:

- Additional steps included in the production process such as buffing a part or belt-sanding a part creating disruption on the line,
- The chance that a containment does not work, such as not completely polishing out a mark that a tool die makes on a part,
- More difficult assembly creating problems, such as a person having to crawl under the product to put on a part, and occasionally introducing a defect, and
- Manufacturing a part slightly smaller to ease assembly, but the looser fit leads to the reliability problems in the field.

The measurement of the variable, *Warranty defects per contained defects*, is debatable. A later section in the thesis contains a discussion of the theory's sensitivity to the variable.

## The Complete Diagram

Figure 11: Complete causal diagram




The structure shown above spells out a hypothesis for the cause of the problematic behavior seen in many firms and in the case of LaunchTech.

The next step in the thesis will develop a formal model of the hypothesis and test it using the system dynamics methodology.

## Testing the Hypothesis

### Base run

The base run of the model captures the low-quality, old mode of product development. All the relevant parameters and equations are listed in the "Model Documentation" section of the thesis.

*Model Runs*

Figure 12

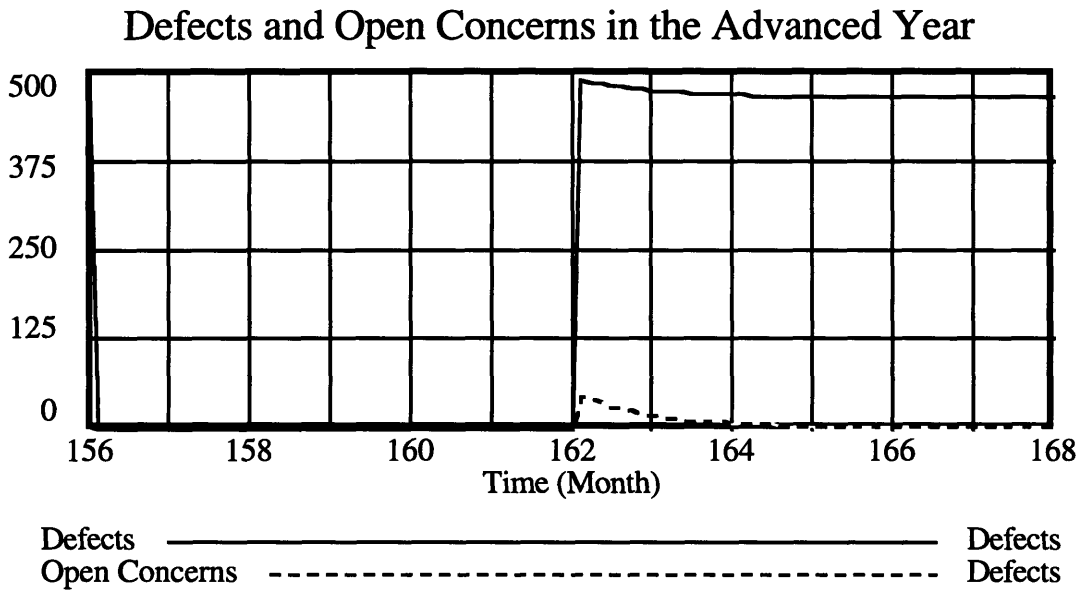



Figure 12 shows the base case behavior for defects and open concerns (defects that have been discovered) in the advanced year, which runs from 24 to 12 months before new product launch. Engineering introduces 1000 designs into testing in month 162, 18 months before product launch (which will occur, for this model year, in month 180). With the designs come 486 defects; the figure shows their arrival as a sharp increase in month 162. Engineers immediately discover 10% of the defects with tools such as computer simulation, and the discovery creates a small number of open concerns for engineers to fix with rework design changes. Fixing an open concern fixes a defect, so as engineers fix the open concerns between months 162 and 165, the total number of defects falls slightly. Because engineers discover so few defects, and because the new design changes that fixed the defects also sometimes introduced a new defect, 463 defects pass (in month 168) into the next stage, the current year. Overall, the product delivery process introduces a large number of defects and does not discover and fix many in the early months.

Figure 13

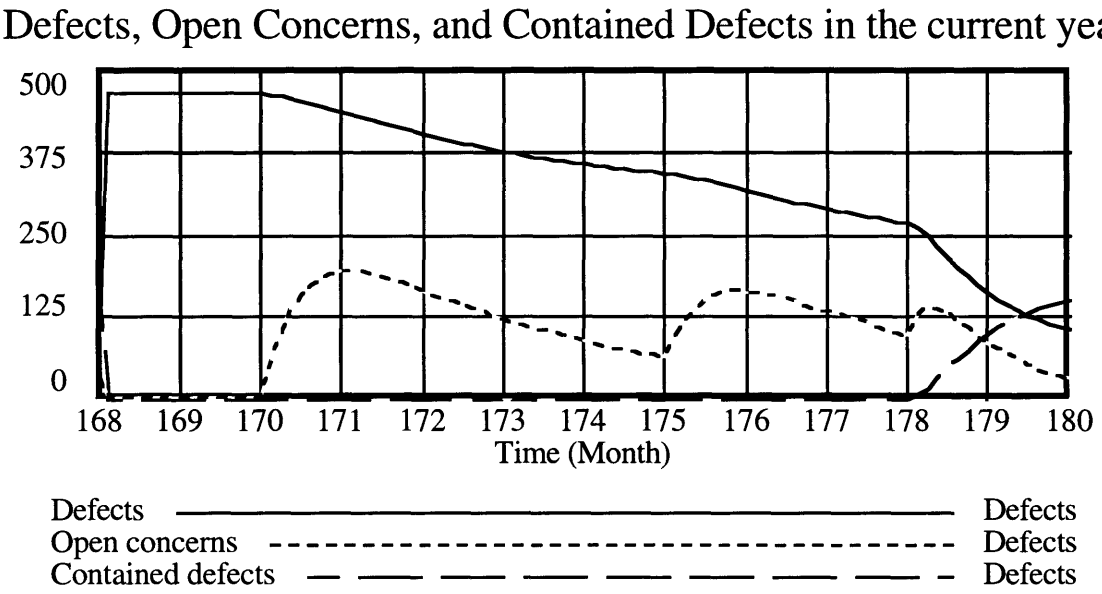



Figure 13 shows several types of defects in the current year, which spans the 12 months before new product launch in month 180. In month 168, 463 defects remain in testing. The defects are undiscovered and untouched until month 170, when manufacturing people run Build 1, and discover 60% of the defects. With discovery, the number of open concerns grows, pushing manufacturing engineers to submit rework design changes to ask design engineers to fix the defects. After a delay, the concerns get fixed, so the number of concerns falls. Fixing an open concern fixes a defect, so, starting in month 170, the number of defects falls too. The rate of decrease of defects (the solid line) from month 171 to month 175 is not as great as the rate of decrease of open concerns (the checked line), however, because rework design changes create some new defects as well, as illustrated by loop R1 in Figure 7.

Manufacturing builds more test products on the line with Build 2 in month 175 and Build 3 in month 178. In both cases, the builds spark defect discovery and thus increase the number of open concerns. In the final two months of the year, open concerns and defects are decreasing at a faster rate, because engineers are quickly containing defects more than they are fixing the root causes. Despite working hard to clear up defects in the final months, engineers do not handle defects by the time of launch, in month 180. The long-dashed line shows the stock of contained defects growing in the final months.

Overall, engineers in the organization are fixing many defects late in the product delivery process, containing many, but still delivering some into production.

Figure 14

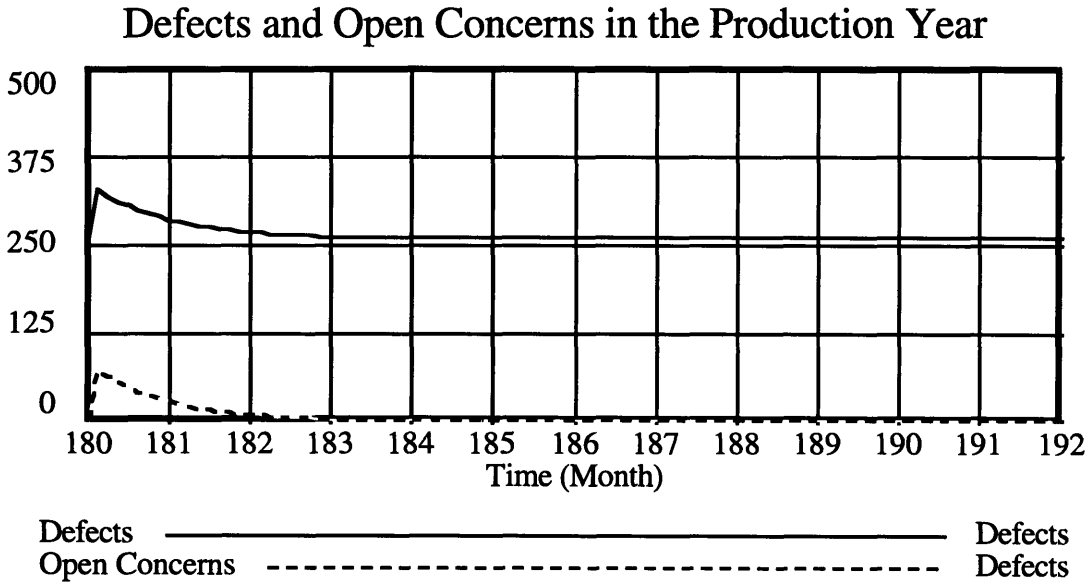



Figure 14 shows (in the solid line) the defects in the production year, the twelve months after launch. Before month 180, there were 260 defects in production, defects that had built up over multiple years. In month 180, the number of defects jumps up from the previous value as new defects arrive into production at launch. Engineers and line operators discover 90% of the defects in launch, so the number of open concerns (the checked line) rises as well. Both stocks decrease in the first few months as engineers either contain or fix the discovered defects, and defects in production fall to 260 once again.

Figure 15

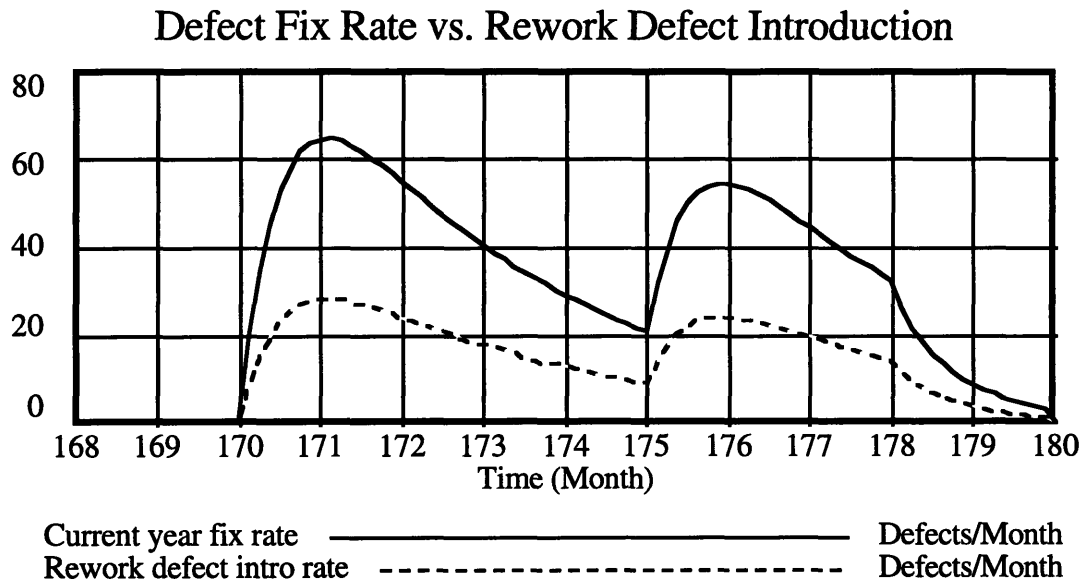
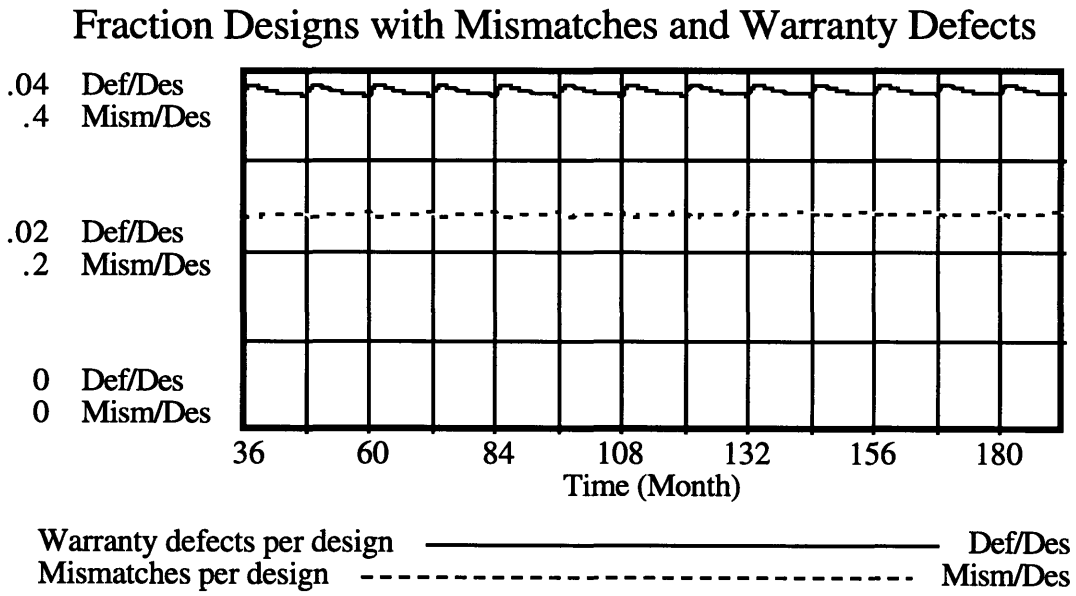



Figure 15 shows the defect fix rate and rework defect introduction rate. The greatest rates of fixing occurs just after Build 1 in month 170 and just after Build 2 in month 175. Because fixing a defect involves making a design change, the periods of greatest fixing are also the periods of greatest rework defect introduction.

Figure 15 shows the relative effects of loops B1 and R1, shown in Figure 7. For example, in month 171 engineers are fixing defects in the process shown in loop B1 at a rate of 63 per month, but the rework design changes are also reintroducing defects via loop R1 at a rate of 29 per month. Fixing defects is a "double-edged sword" -- it both eliminates and reintroduces defects.

Figure 16




While previous figures showed one year of behavior, Figure 16 shows the long-term behavior of two variables, the *Warranty defects per design*, and the fraction of *Mismatches per design*. The figure shows 13 years of behavior, with launch occurring on each grid line (in month 36, 48, 60 etc.). The number of defects are slightly higher at the start of the year, making the line jagged, because defects introduced at launch remain in production for a short time before engineers fix them. Equilibrium requires that the number of defects fixed or retired every year is equal to the number introduced. The number of mismatches per design in production is in equilibrium at 24%, meaning that the number introduced is equal to the number fixed or retired.

Figure 17

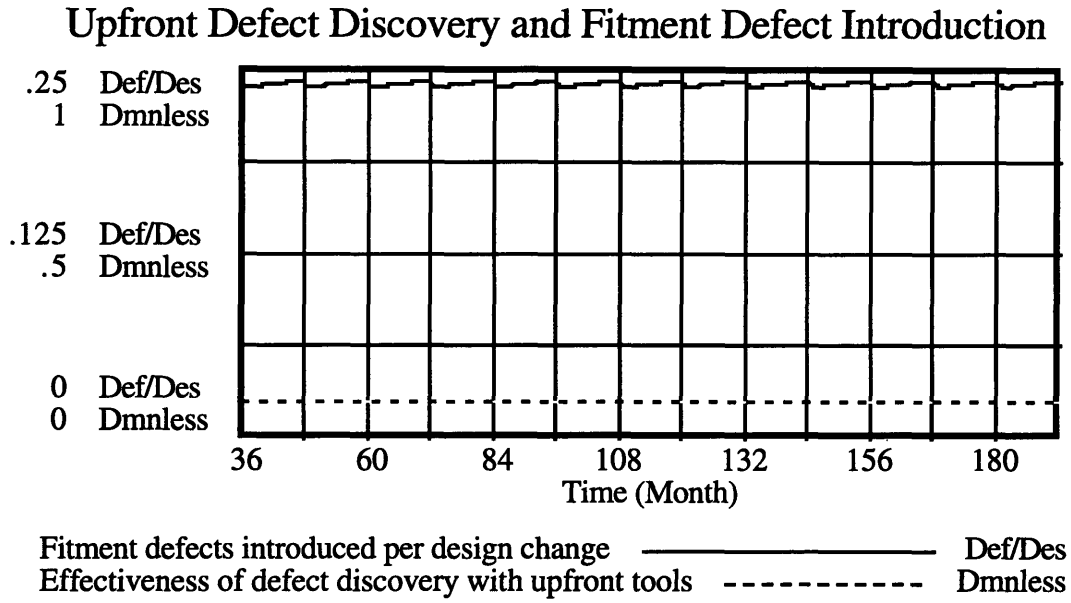



Figure 17 shows two critical variables in the loops R2 and R3 from the causal diagram, figure 11. Fitment defects introduced per design change is in equilibrium around .24, meaning that every time a design engineer makes a design change, he or she has a 24% chance of inadvertently introducing a new defect. Effectiveness of defect discovery with upfront tools is in equilibrium at .11. That is, engineers use tools such as simulation to discover 11% of all defects introduced at the beginning of the product delivery process. Figure 12 (several pages back) shows the effect of the 11% discovery on the creation of open concerns and the reduction in defects in the advanced year.

*Base Case Summary*

In the base case, the organization's product development process operates in a low-quality mode. Engineering introduces a large number of defects several years before launch and discovers a few with upfront simulation tools -- the tools are not yet effective. With minimal discovery, engineers fix a few defects before the start of the current year (the year before launch). People in manufacturing, then, discover and fix almost all the defects in the twelve months before launch, but many fixes reintroduce new defects along the way (as shown in loop R1). The result is that some defects get contained or directly enter production, where they join other defects that already exist in production.

The rates of containing defects, launching defects, and rework design changes combine to create new mismatches between parts and prints, but retiring mismatches offsets the level

of creation. While the number of mismatches is not growing, it has leveled at a high value -- almost a quarter of designs have part and print mismatches. The high percentage of mismatches feeds back to drive engineers to introduce a large number of defects per design change and do little defect discovery early in the product delivery process. The result is a low-quality overall process with a high percentage of defects and part/print mismatches in production.

The base run captured the most important features of the "reference modes" that defined the problem -- high defect introduction and minimal early discovery led to a large fraction of defects per design and mismatches per design. The base case operated in a dynamic equilibrium -- the values of key stocks were constant in the long term, even though inflow and outflow rates were positive. For example, the number of mismatches was not constant in the long run because the same mismatched designs stayed in the system. Many were created and destroyed every year, and yet the number was relatively constant in the long term. In this way, the low-quality mode appears to be a long-term fixture in an organization fitting the characteristics of the model; the problem looks unlikely to solve itself.

## Pulse or "Lobin" Test

One test of the model is to deliver a shock to the system -- in this case a set of designs entering final testing six months before production launch -- and observe the behavior. It is interesting to see, in particular, how the two reinforcing feedback loops respond to the shock.

### *Test in the Real World*

Occasionally, a set of design changes gets introduced into the product delivery cycle after the usual deadline. For example, the marketing department may announce that a certain product is not selling well and would be more successful if a new feature was added to the design, such as parts a different color or a different shape. If the need is pressing, marketing could introduce the change in the year before new product launch. Playing on the way such changes seem to arrive from nowhere, manufacturers called such changes "lobins."

*Test in the Model*

In the "Lobin" test 200 design changes arrive in month 66, after Build 2. All other parameters are set at their base case values.

*Model Runs*

Figure 18

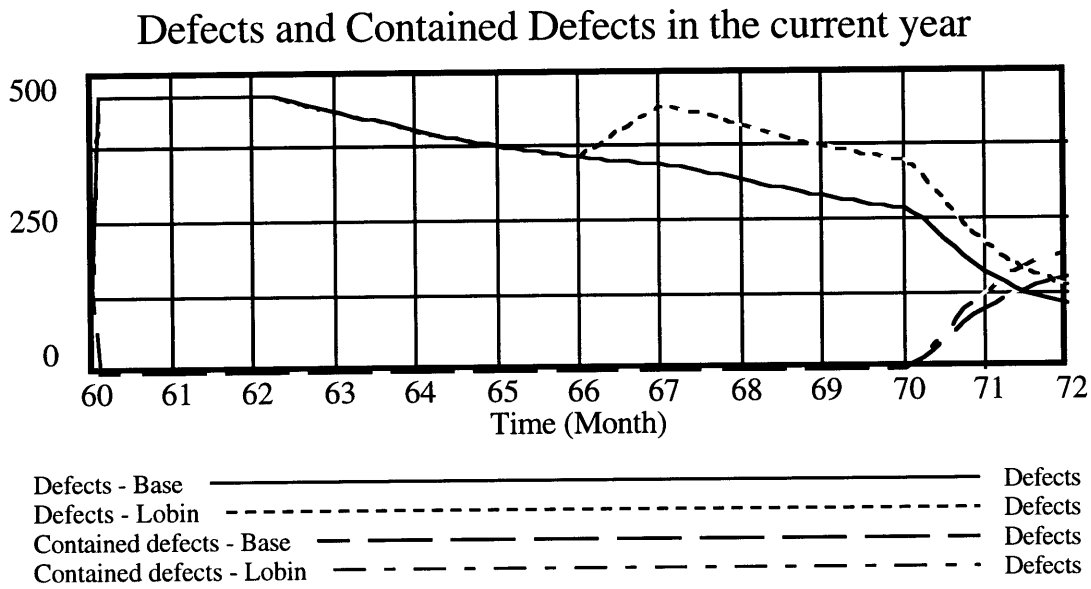



Figure 18 shows the *immediate* effect of the lobbed-in designs. Comparing the "Lobin" run to the "Base" run over a single year, the number of defects rises dramatically in month 66 as some of the new designs bring new defects with them. The number of defects then falls as engineers discover and fix some of the defects. In the "Lobin" run, engineers contain more defects but still pass more defects enter production than they do in the "Base" run.

Figure 19

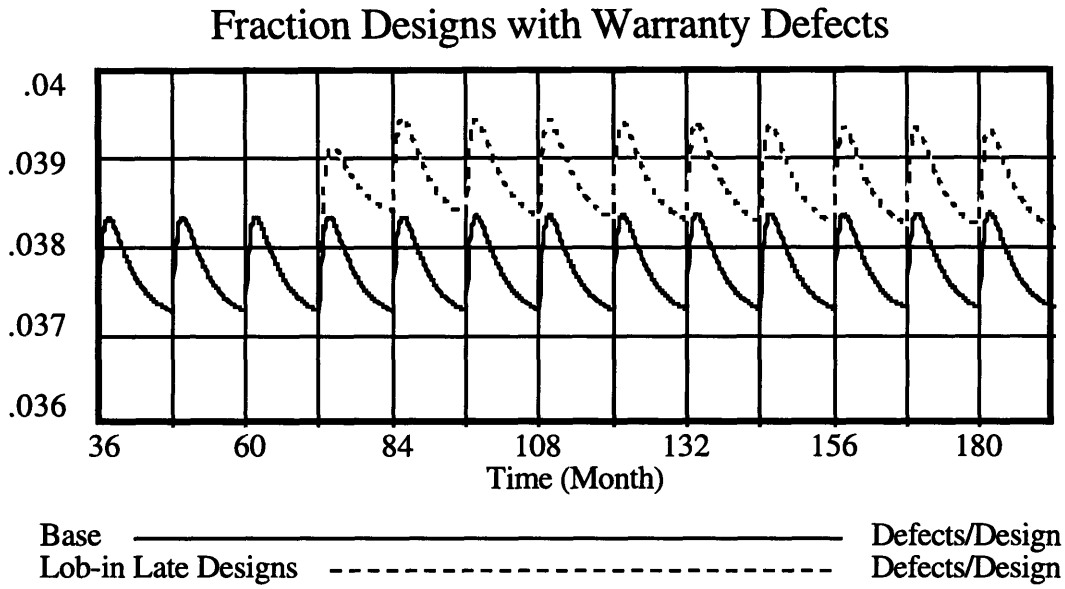



Figure 19 shows the effects on quality. As the previous figure shows, defects arriving in month 66 do not enter production until launch at month 72. In the first year after the pulse, quality is worse; the dotted line jumps up higher than the solid line. One might expect such a short term reaction -- it would be difficult to find and fix all the new defects before new product launch. But the number of defects does not fall back to the base case level for the next ten years. The organization never recovers from the one-time arrival of more defects.

One hypothesis is that defects that get launched may stay in production for a long time. Indeed, *Defects in production* and *Contained defects in production* are both "stocks" that are built up and drained down slowly. In the model presented here, if defects in production are not actively fixed, then they are reduced only by slowly retiring defects. One might expect the variable in Figure 19, *Fraction designs with warranty defects*, to fall slowly after defects are lobbed in, but not to stay constant at the higher level. There must be another process in place that sustains the low-quality behavior.

Figure 20

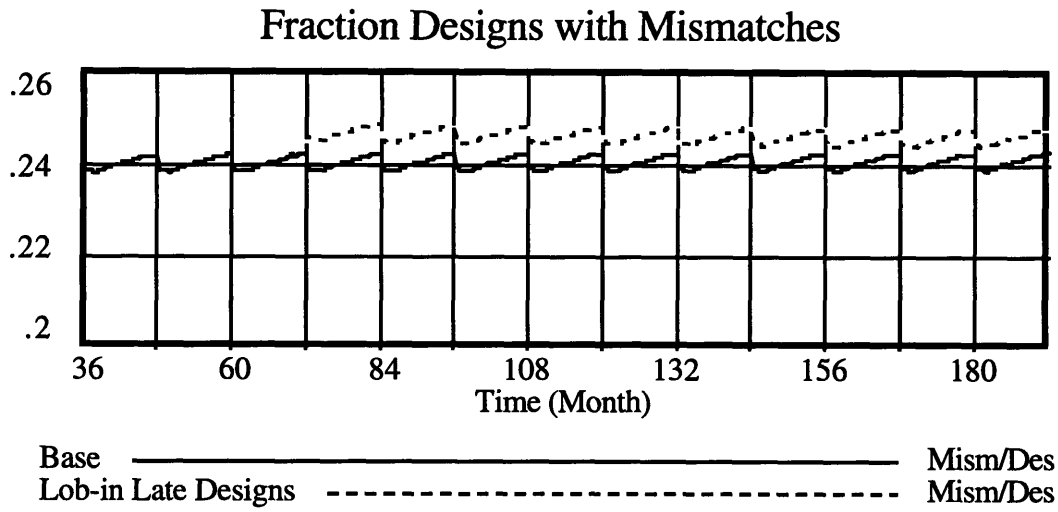



Figure 20 shows an important variable in loops R2 and R3, the fraction of designs with mismatches. In month 72, after the test, the lobbed-in defects create more mismatches, driving the fraction of designs with mismatches higher.

Figure 21

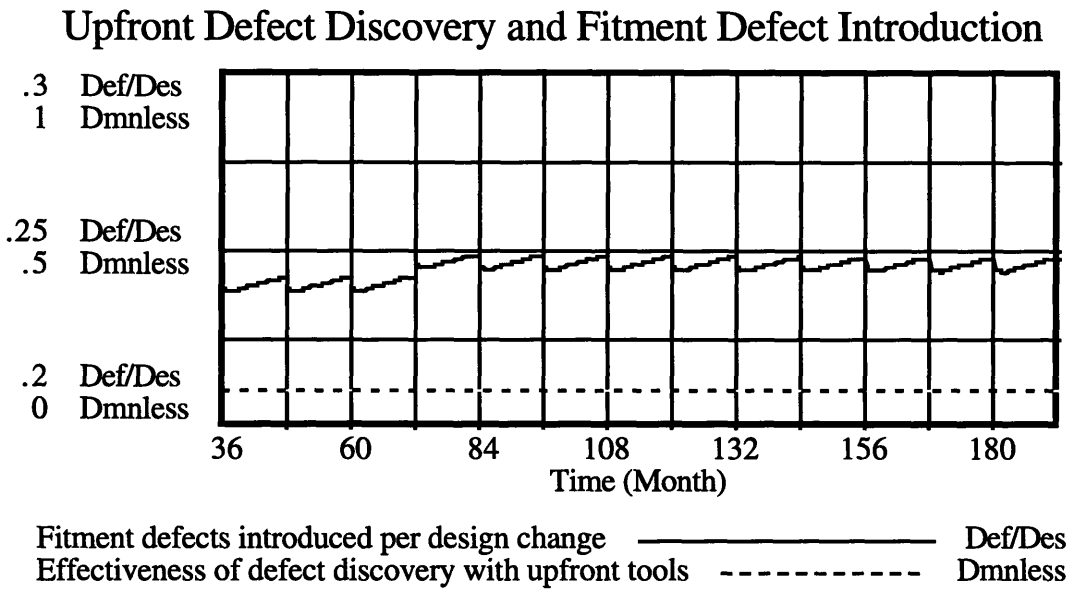



Figure 21 shows the effects of the increased number of part and print mismatches. *Effectiveness of defect discovery with upfront tools*, a critical variable in loop R2 of Figure 11 (the dotted line), remains unchanged, indicating that the loop R2 is likely not driving change in the system. When engineers are discovering defects early in the process, they are not relying on a well-matched database, so a small increase in the number of

mismatches does not adversely effect their ability to discover defects early. The rate of early discovery is already below its desired level, and thus unlikely to worsen.

The figure also shows the behavior of *Fitment defects introduced per design change* (the upper, dotted line). In month 72, because of the increased number of mismatches, slightly more defects are being introduced by design engineers in the advanced year, approximately 18 months before launch and during the current year, when rework design changes introduce new defects. The effect on quality from more defect introduction will not be realized immediately, in month 72. Instead, the effect will be delayed until month 84, with the launch of designs created and reworked in an organization with more mismatches than it had previously.

One significant cause of the sustained poor quality from the lobbed-in defects is that the feedback dynamics explained by loop R3 (the fitment defect introduction loop) is holding the system in a lower-quality mode, with slightly more defect introduction and launching than previously. With loop R3 active, the influx of defects are not draining out of the system as one might expect. If the hypothesis was true, than the system with loop R3 *inactive* would produce different behavior; the new defects would simply drain out over time. The next model run shows the results of this test.

Figure 22

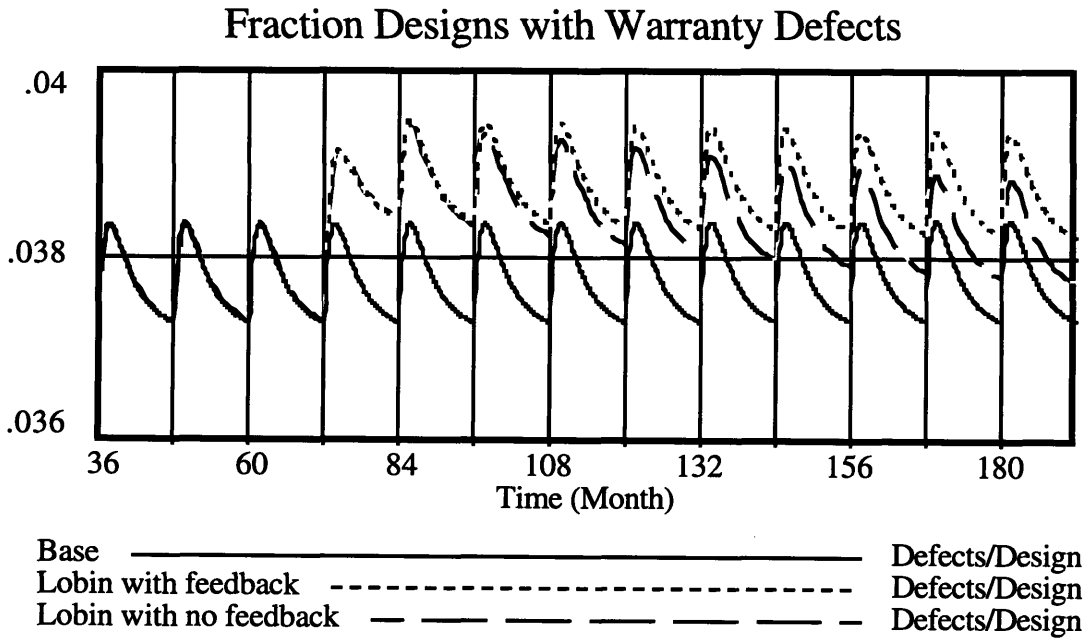



Figure 22 adds an additional run to the graph in Figure 19. "Lobin with no feedback" has

*Fitment defects introduced per design change* fixed at the level in the base case. It does not increase as more mismatches get created. Without loop R3, the fraction of designs with warranty defects slowly returns toward the level shown in the base case. Without the loop active, engineers did not begin introducing more fitment defects, and after several years retirement of designs restores the higher quality. With the loop active, on the other hand, increased defect introduction traps the system in the low-quality regime.

*Summary of Lobin test*

A single, small, one-time introduction of defects permanently decreases the quality of products. Some new defects get launched into production in the first year; even more get launched the second year, when the first year's defects created mismatches that increased the number of fitment problems introduced by design engineering, as explained by loop R3. The perturbation to the system has two effects -- one immediate and certainly part of the understanding of the managers who are working to improve this system, and one more delayed and subtle.

Besides delayed behavior, the system also displays a sustained effect from the one-time pulse of new defects. Principle sources of the effect are twofold: defects in production leave the system gradually; and engineers continue to introduce more defects into the system with every design change because the increased number of mismatches holds the organization in the lower quality regime.

The long-term effects of a pulse of design changes is counter-intuitive. One might have expected that because of the system's stability, balancing loops would have quickly moved the system back into the original mode, like a marble that is pushed up the inside of a bowl quickly rolls back to the bottom. Instead, defects created new mismatches, the mismatches created new defects, and the system found a new, lower-quality equilibrium. Because of the mismatch dynamic, what may seem like a one-time, short-term trauma to the system may actually have longer term implications. Further, if a small pulse of defects or design changes can move the system into a lower-quality regime, then it would seem possible for sustained growth in design changes to drive a sustained decline in quality.

**Pulse out Mismatches test**

The hypothesis tested in this model depends critically on the existence of mismatches between parts and prints. The following model tests how the system would behave with fewer mismatches.

*Test in the Real World*

A step decrease in the number of mismatches is not a realistic test -- it is equivalent to a group of people changing 1,500 mismatched prints overnight . However, it provides insight into the behavior of the system.

*Test in the Model*

In month 60, 1,500 part and print mismatches (63% of the total) are eliminated.

*Model Runs*

Figure 23

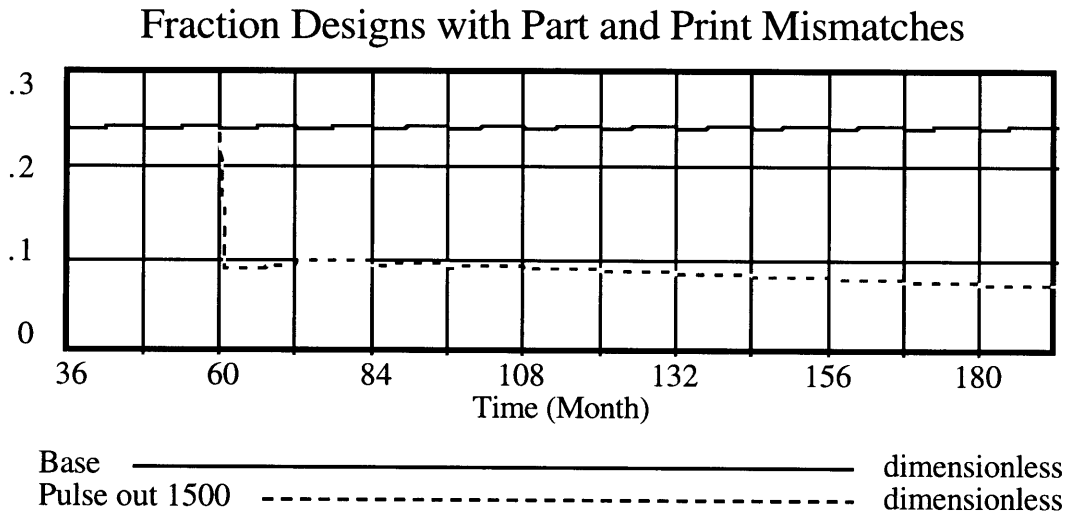



Figure 23 shows the effect of the test on mismatches. The fraction of mismatches immediately drops 63% when the test is implemented. At first the number of mismatches begins to climb slightly as the projects begun and completed before month 60, when quality was lower, are being launched and are creating mismatches. Soon after, however, the projects that are being launched (after month 72) will have benefited from the lower-mismatch, higher-quality product delivery function. Fewer mismatches lead to less defect introduction and more defect discovery, and thus fewer mismatches created. Although the improvement is slight, the slowly falling mismatches shows the continuing effect of loops R2 and R3.

Figure 24

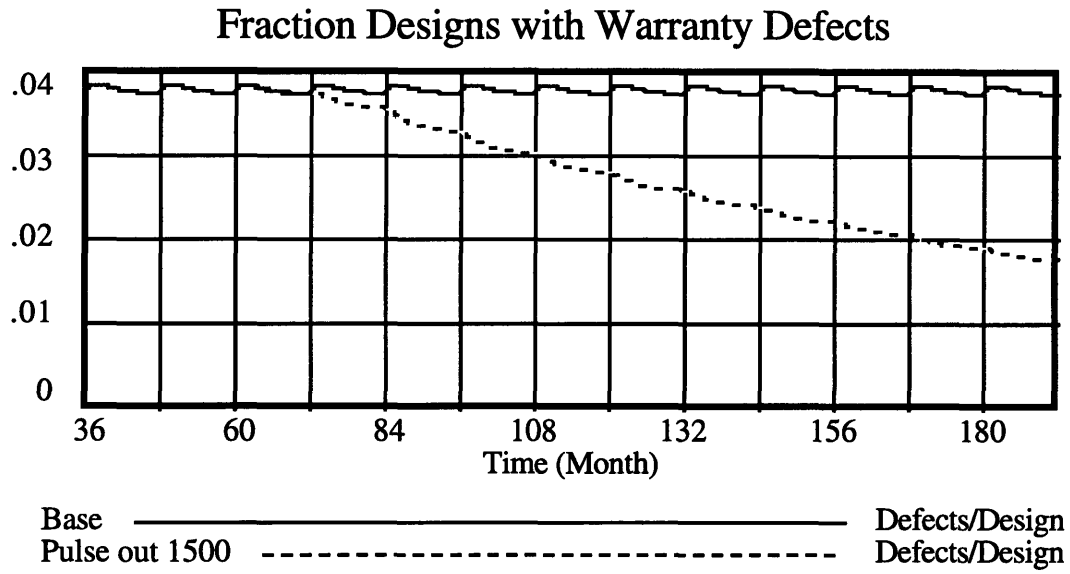



Figure 24 shows the effects on quality. In month 60, when the number of mismatches drops, mismatches are driving engineers to create far fewer mating part fitment problems. Also in month 60, fewer mismatches in the print database are allowing engineers to use more simulation tools for defect discovery. The first effects of the improvement do not effect product quality until month 72, when the designs launched have been reworked under a less defect-ridden product delivery function. The more significant effects (and thus greater rate of improvement) begin in month 84, when the projects for which all the upfront discovery was done in the higher quality environment are launched into production. Over the next ten years, as fewer defects are entering production, and defects continue to be retired out, the fraction of designs with warranty defects falls.

Figure 25

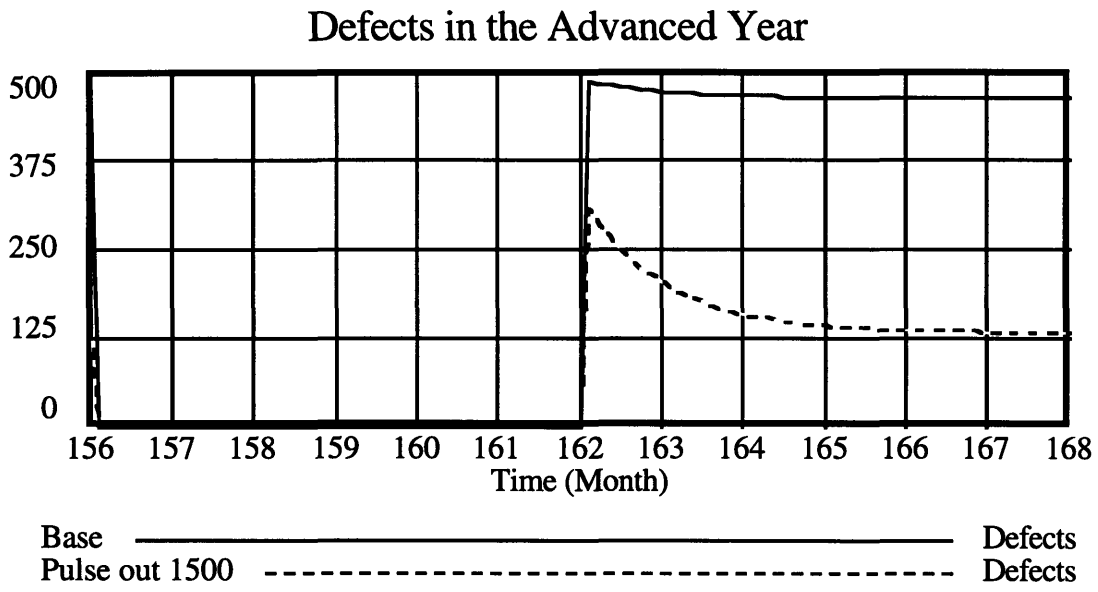



Figure 25 shows *Defects in the advanced year* for the base case (identical to the run in figure 12) and for the pulse test, over one year of time eight years after test implementation. In the base case, a large number of defects are introduced 18 months before launch (in month 162), and engineers discover very few with upfront testing tools like CAD/CAM and other simulation tools. Thus, design engineering is able to fix few defects before the production builds (which start after month 168). In the pulse test run, because there are fewer mismatches, engineers introduce fewer defects in month 162 (the maximum is 306 instead of 487). Further, they discover and fix a greater percentage of the defects -- without all the mismatches, simulation tools are effective. The percentage discovered has leapt from .1 to .62. Because of the early fixing, the number of defects falls steeply in the final six months of the year, and only 132 defects pass on to the next year, as opposed to 463 in the base case.

Figure 26

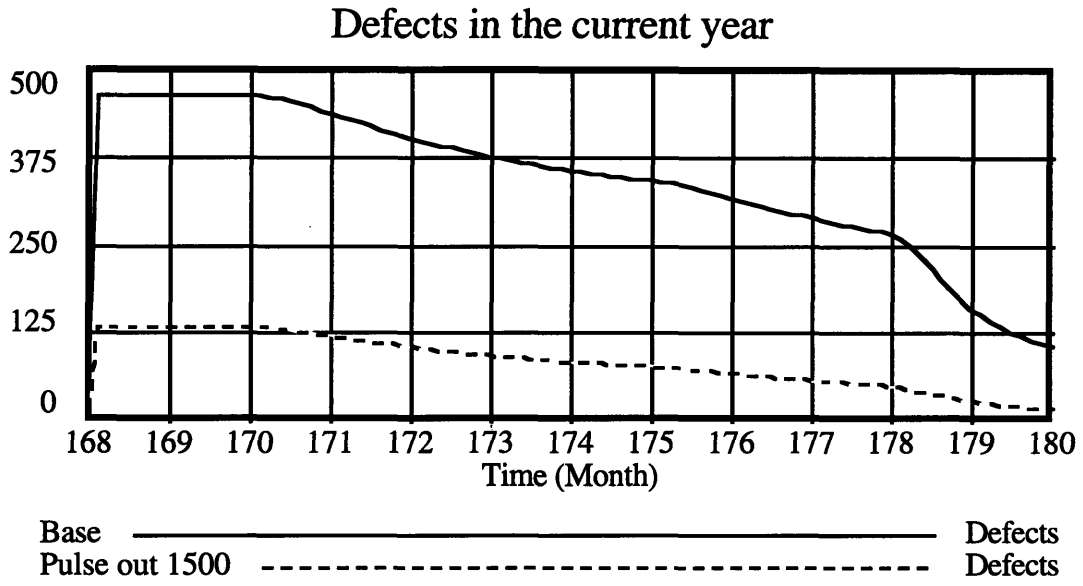



Figure 26 shows the behavior of defects in the next year, the twelve months before launch. In the base run, many defects are fixed, but some are still passed into production. In the test run, engineers discover and fix almost every defect before launch, and very few defects pass into production.

*Test Summary*

The "Pulse out 1500" test outlines a high-quality mode for a product delivery function, as shown in the reference modes in figures 1 and 2. In the organization with few mismatches, engineers introduce fewer defects and discover others earlier with upfront tools, leading to fewer last-minute fixes of defects, and fewer defects in production.

The organization after the pulse test is similar to the organization that much of the literature on improved product development recommends -- engineers prevent defect creation and find defects early. As the introductory sections of this thesis outlines, however, the challenge facing many companies is not describing the new mode for a product delivery function, but implementing policies to get there.

"How to get there" is the challenge of the next section of the thesis. How to get from the low quality base run to the high quality "pulse out 1500" run? Various policies will be tested.

## Policy Test -- More Documentation

The first policy test will address more documentation of potential mismatches. Perhaps by stemming the inflow of mismatches, the mismatches in the system will decline over time, and reinforcing feedback dynamics will also drive improvement. Model runs and analysis will test this theory.

*Policy in the Real World*

The first policy run examines increased documentation of potential mismatches. In the real world, more documentation would mean two improvements:

1. Documenting containment -- engineers in manufacturing would submit a formal design change notice to design engineering every time they have adjusted the production process and changed the design in some way, and design engineering would quickly update the prints to reflect those submitted changes.
2. Improved change order documentation -- fewer changes to prints would introduce mismatches through errors or by accepting the mismatches as a tolerable part of the database.

*Policy in the Model*

In the model, the two improvements involve:

1. Documenting containment -- in the base run, only 10% of the defects that are contained get documented in the manners mentioned above. In the policy run "More Documentation," the percentage increases to 90%.
2. Improved change order documentation -- in the base run, 10% of design changes introduce mismatches. In the policy run, the percentage drops to 0.

*Policy Runs*

Figure 27

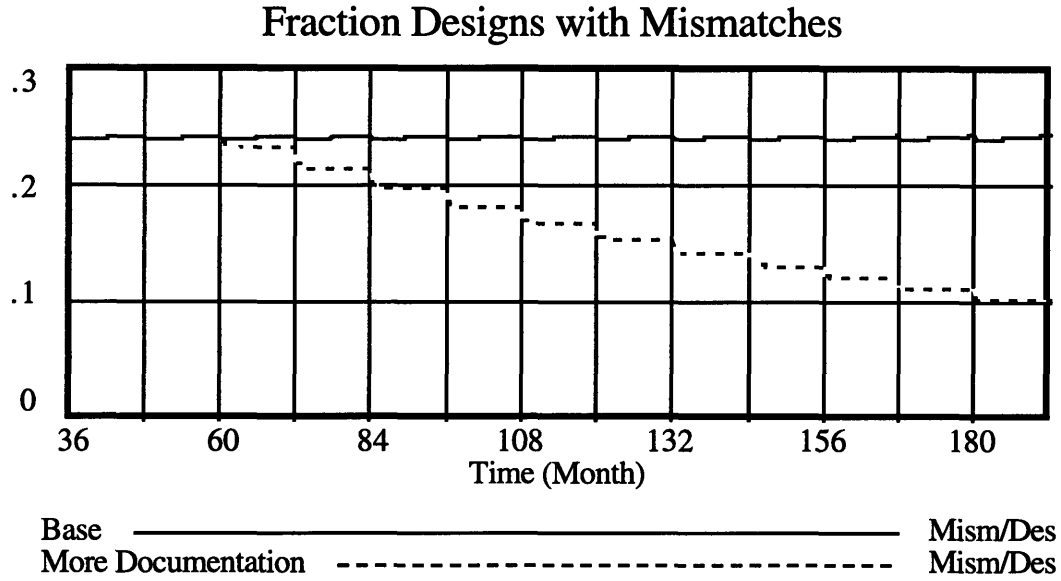



Figure 27 shows the effect of the policy on the fraction of designs with part and print mismatches. In month 60, when the policy is implemented, design engineers create fewer mismatches as they document rework changes and contained defects better, but close to the same number of mismatches get retired as in past years, so the overall number of mismatches falls slightly. By month 72, the effect of the policy is greater, as engineers do even more work with the new policy in place. In five years, the number of mismatches has dropped approximately 30%, a slow but significant improvement. It appears that in the time frame shown, the rate of improvement is decreasing, but it will be necessary to examine a longer time frame to confirm this guess.

Figure 28

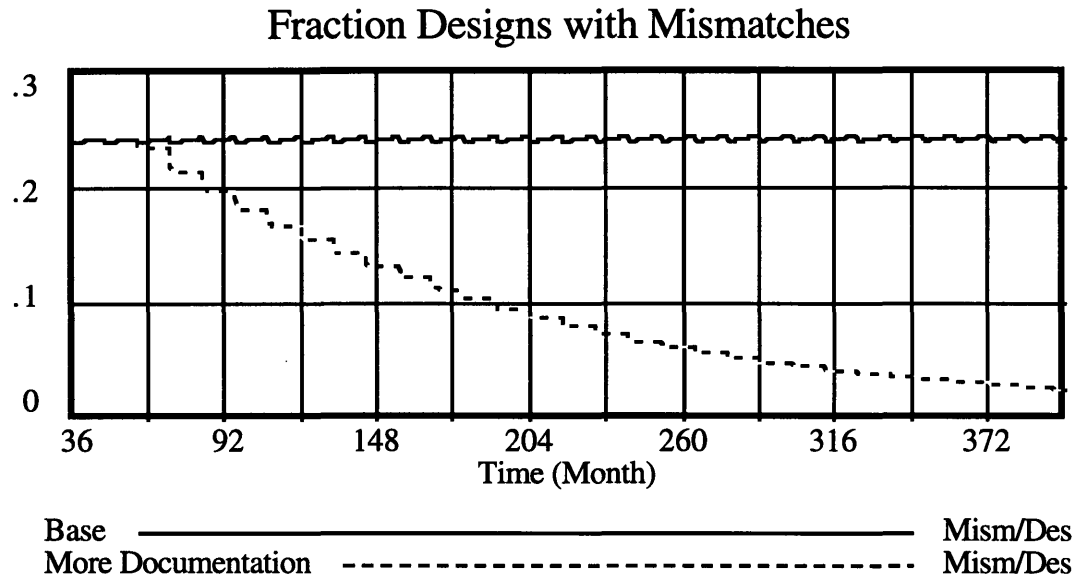
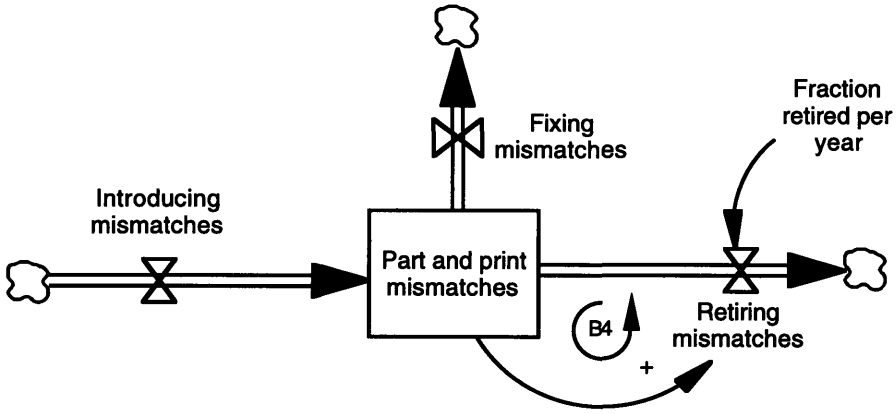



Figure 28 expands the time frame to 30 years. The rate of improvement is decreasing over time. Why?




One dynamic driving the decreasing rate is loop B4, shown in Figure 3. When the stock of mismatches is high, the rate *Retiring mismatches* is high. Fast retirement feeds back to decrease the number of mismatches, thus decreasing the rate of retirement in the next year. In this way, the stock of mismatches behaves like a sink of water with a small inflow and the drain wide open -- the water will flow out quickly at first, and more slowly ever after. Engineers clearing up mismatches by decreasing their creation may observe similar dynamics. Every year there will be fewer to retire, so the mismatches will leave more and more slowly.

Figure 29

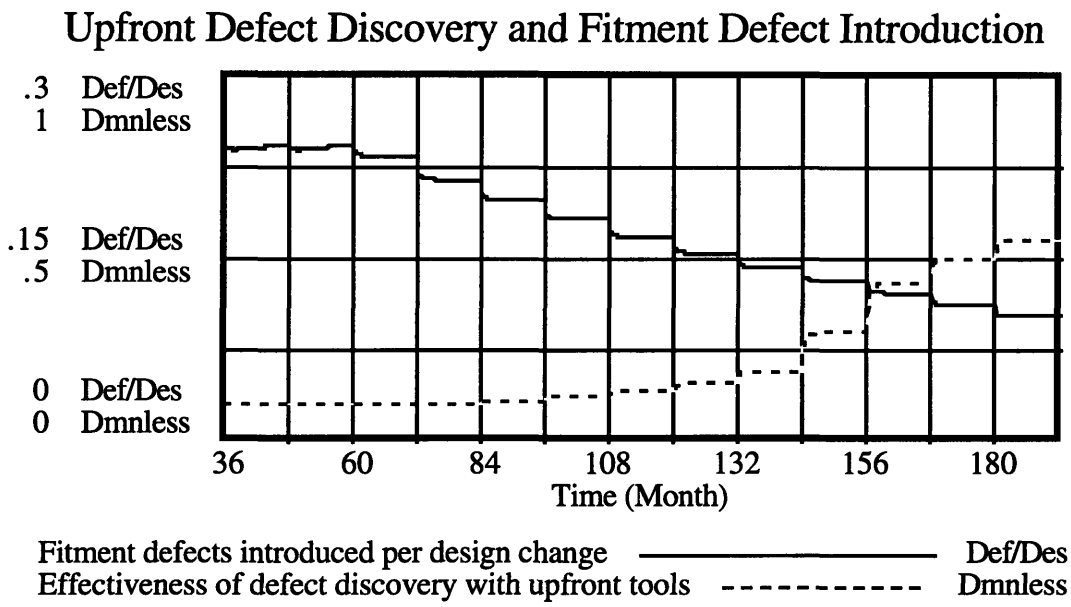



Figure 29 shows the effect of the policy on the several important variables in loops R2 and R3. As the number of mismatches falls, engineers introduce fewer fitment defects with each design change. There are progressively fewer cases where, for example, a design engineer creates a part to mate with a square corner on another part, when in production that other part actually has a round corner. Mimicking the shape of the mismatches curve shown in Figure 27, the rate of improvement is gradually decreasing.

As mismatches decline, the *Effectiveness of defect discovery with upfront tools* is level for a long delay, then increases slowly from month 84 to month 144, and then more steeply. By the end of the run, over 50% of defects are discovered early. During the slow improvement, simulation tools such as CAD/CAM are not yet effective without a database of prints that match parts, even though the database is improving. In time, enough prints match parts that engineers are able to use simulation tools effectively, enabling them to find more defects early.

With the documentation policy, the reinforcing loops R2 and R3 drive improvement; defects are being prevented and more are discovered and fixed early.

Figure 30

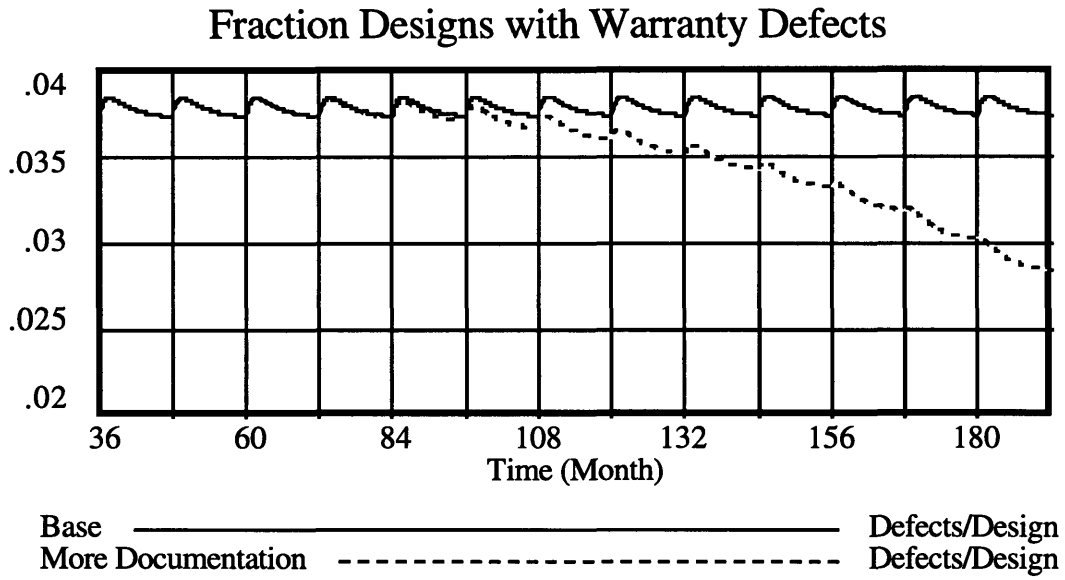



Figure 30 shows the effect of the policy on the fraction of designs with warranty defects. No effect on quality can be seen until approximately three years after month 60, when the policy is implemented. Indeed, the reduced number of mismatches will start helping the first designs in month 66, for a model year that would not be launched until month 84. After a significant delay, the improvement in quality is gradual for the first few years -- engineers introduce fewer and fewer defects, driven by loop R3. Several years later, quality begins to benefit from the increased effectiveness of upfront defect discovery. As indicated by the previous figure, engineers are increasingly able to use simulation technologies to find problems. The rate of improvement in quality thus increases. Even with the breakthroughs in defect prevention and early discovery, the reduction in warranty defects is gradual, as defects leave production through the slow process of retirement of designs. By month 192, it is unclear whether improvement will continue or stop. What is the long term behavior of this variable?

Figure 31

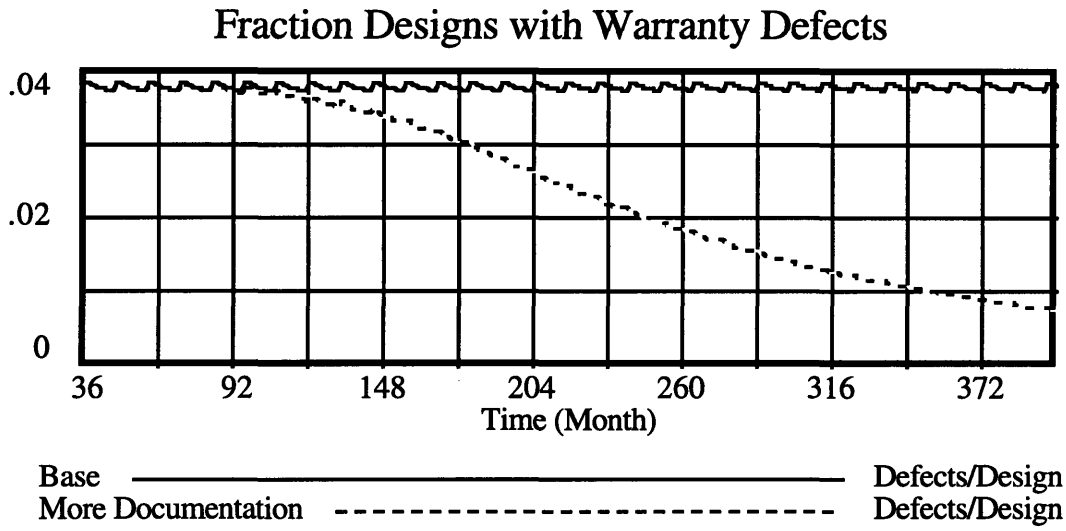



Figure 31 lengthens the time horizon of the graph shown in figure 30. The rate of improvement is initially increasing, but begins decreasing around month 230, and decreases ever after. The shape in the first region suggests that reinforcing loops, likely R2 and R3, are dominating the behavior, accelerating the improvement. Fewer mismatches leads to more defect prevention and early discovery, leading to even fewer mismatches and fewer defects.

The shape in the latter region suggests that balancing loops are dominating. In particular, loop B3, B4 and B5, which drive the retirement of mismatches, defects, and contained defects, may drive the behavior. There are progressively fewer mismatches and defects to eliminate from the system, so fewer and fewer get retired every year.

Figure 32

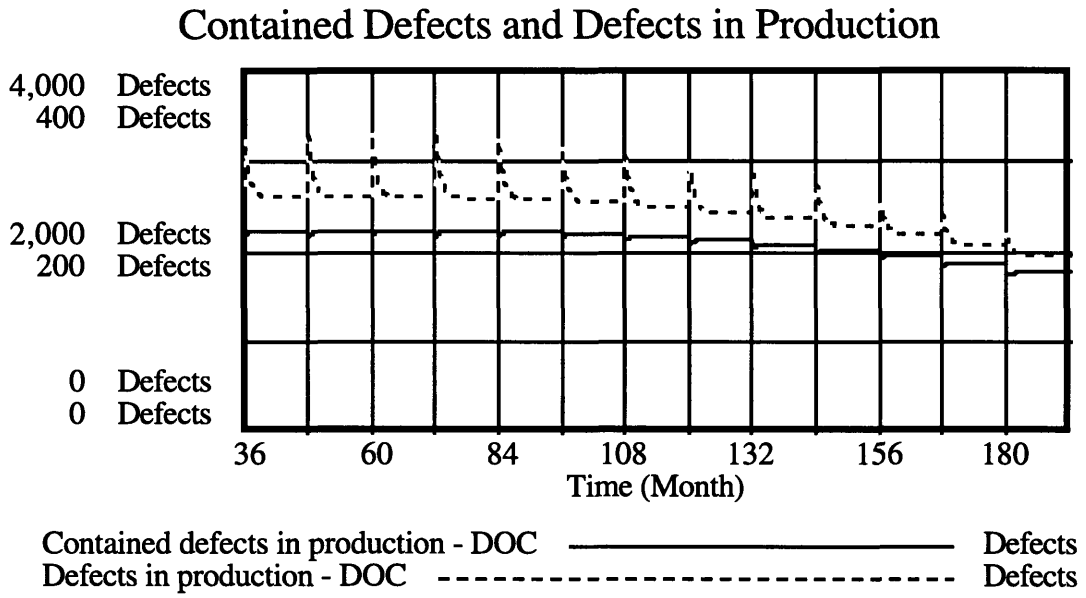



Figure 32 shows the behavior of *Contained Defects in Production* and *Defects in Production*. The second variable has a peak at the start of every year because new defects arrive in production at launch, and engineers take several months to fix the defects they discover. The figure provides more insight into the delayed effect of the documentation policy. The policy does not directly effect contained defects in production and defects in production, the two sources of warranty defects, until multiple years after the beginning of the policy implementation. The reduction occurs as fewer defects get created, fewer get contained, and, with the retirement rate close to its original level, the stock declines slowly. The turnover of the designs is not high enough, however, to drive a significant decline in contained defects and thus an increase in quality.

*Sensitivity to retirement rates*

The "More Documentation" policy reduces the creation of mismatches and defects and thus relies heavily on retirement of designs to rid the organizations of problems created in the past. The selection of a design turnover rate (shown in Figure 4 as *Fraction retired per year*) would then seem quite important in this analysis -- conclusions could be highly sensitive to its value. In particular, how much better would the system perform under the policy if retirement were faster?

Figure 33

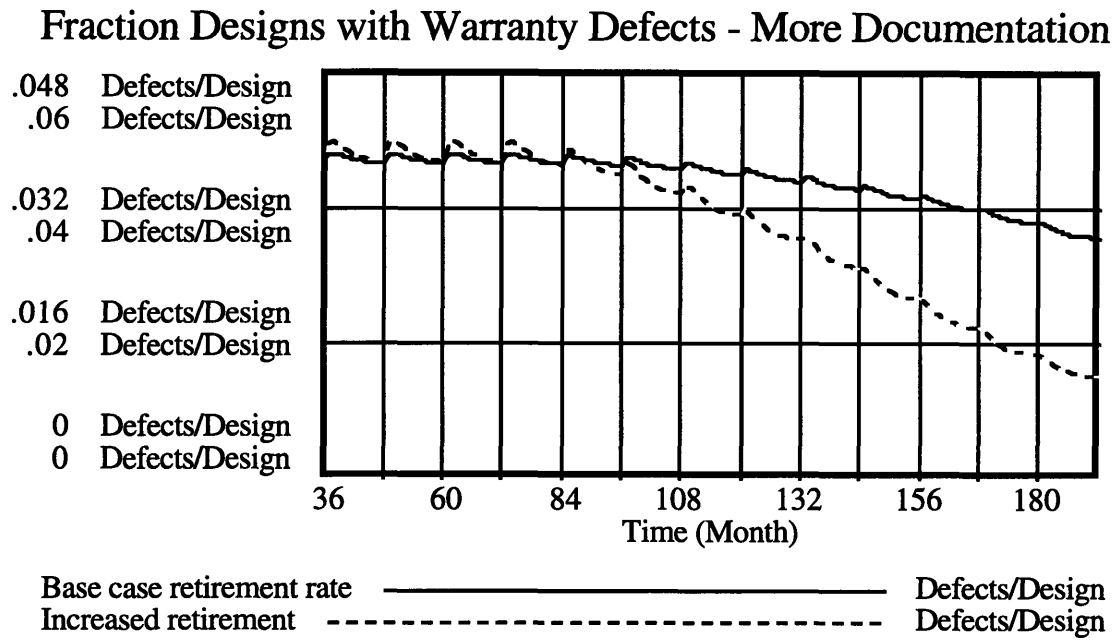



Figure 33 shows the result of a simple sensitivity test. In the run "Increased retirement," 25% of the designs turn over every year, as opposed to 10% every year, in the "Base case retirement run." Both runs have the "More documentation" policy in place. The vertical axis is normalized so that the equilibrium levels (which differ in the two runs) are comparable.

With higher turnover, the mismatches, contained defects, and defects created in past years leave the system much faster, driving improvement in quality more quickly and sooner. By the end of the model run, almost all the designs in production had been created in the organization under the new policy of increased documentation and quality is much better. The rate of improvement, then, is sensitive to the retirement rate.

Unchanged, however, is the almost three-year delay between enacting the policy and realizing its improvements on quality. The delay is insensitive to the retirement rate -- in any case, the improved designs and prints must travel through the 2.5 - 3 year product development pipeline before they can significantly improve quality.

Figure 34

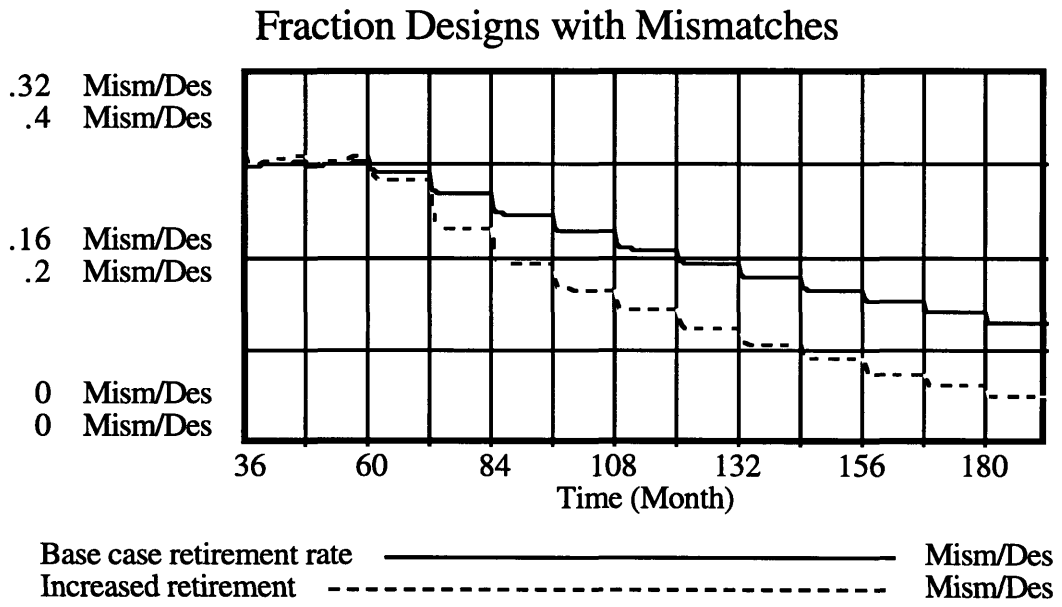



Figure 34 shows the effects of the sensitivity test on mismatches. When one assumes a higher rate of retirement, the documentation policy drives a higher rate of decrease of mismatches. As opposed to a 37% decrease after 5 years, with 25% turnover there is a 60% decrease.

*Loop strength*

So far in this paper, discussion of loops R2 and R3 have made no distinction between the effects of the two. When the number of mismatches falls, previous sections of this thesis have explained that quality has improved by way of both reduced defect creation and increased effectiveness of discovery through upfront testing. But which process is more powerful? At what time in the life cycle of the policy is each process more important? Would the theory hold if one of the feedback dynamics assumed to be in place in the real world is actually not?

To answer the questions, the "More documentation" policy was tested with various feedback loops inactivated. In the first test, for example, the run "Upfront discovery loop R2 inactive" had the variable *Effectiveness of defect discovery with upfront tools* set to its initial value of .1 throughout the model run. As mismatches decreased, the effectiveness of discovery did not increase. In the real world, the test assumes that design engineers are not going to be able to use upfront simulation tools to find defects earlier, even if the database of parts and prints were more accurate.

Figure 35

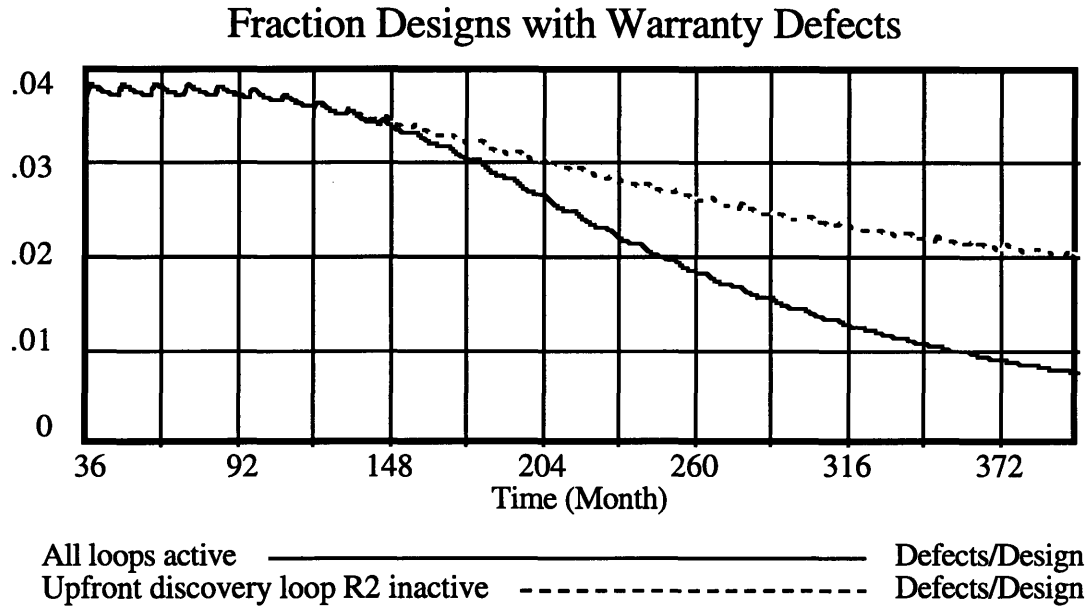



Figure 35 shows the result of the first test on product quality. Without the improvement driven by improved defect discovery, the organization relies (among other dynamics) on the improvement driven by loop R3, which prevents fitment defects from being introduced as mismatches decrease. For the first five or six years after the policy is enacted, the improvement in quality is identical to the standard documentation policy test, marked "All loops active." Engineers documenting better reduces the number of mismatches in system, causing other engineers to introduce fewer fitment defects into the system. After a several year delay, the improvement helps quality, just as in the previous model run. The effect on fitment defects (shown as loop R3 in the diagram in figure 11), then, has a significant role in driving short-run improvement from the policy. Further, short-run success of the documentation policy (albeit meager) does not depend on decreased mismatches improving defect discovery. Assume loop R2 is a fallacy, and the recommendation for short-run effects still holds.

But, turning to consider the long term, reducing mismatches only decreases one type of defects being introduced, fitment defects, and other defects (design defects, tooling defects, and assembly defects) are still being introduced. With defect discovery remaining at its low level, the policy has limited effects in the long term -- defects are still being launched into production, hurting quality, so the level of warranty defects does not fall as far as it does with both loops active.

Figure 36

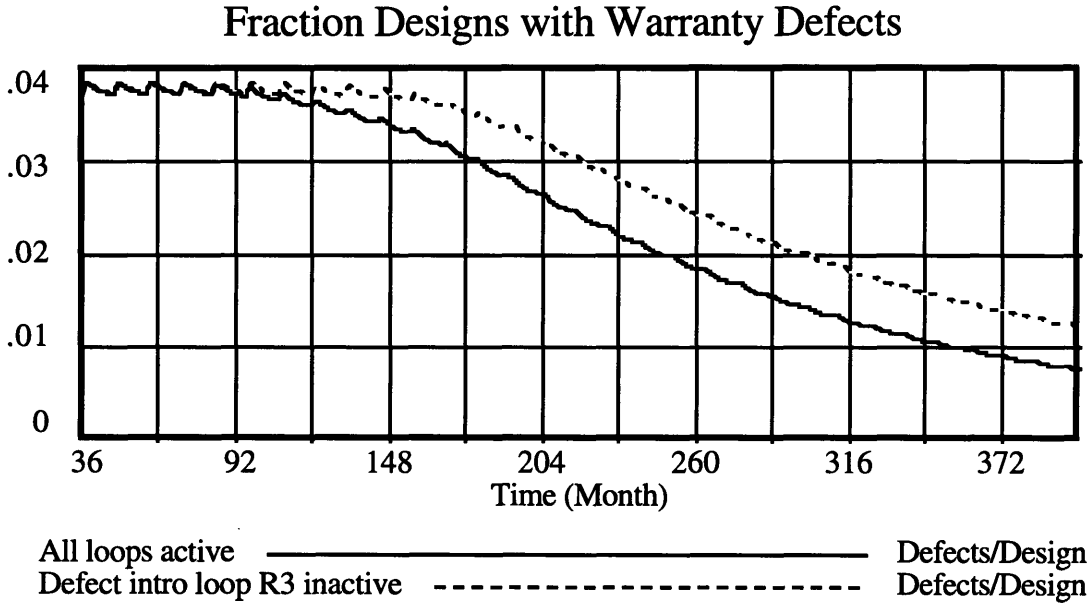



Figure 36 shows the results of a second test of loop strength. The run, "Defect intro loop R3 inactive" shows the behavior of the "More documentation" policy with the variable *Fitment defects introduced per design change* set to its initial value of .24 throughout the model run. As mismatches decreased, engineers introduce just as many fitment defects as they had in the past.

While the effect of the documentation policy is already delayed by approximately three years, without the improvement from the fitment defect reduction, the delay extends another four years. The mismatches must fall significantly before the organization is able to benefit from engineers using simulation tools to find defects early (CAD/CAM, for example, requires a highly accurate database). But with the documentation policy, mismatches fall only gradually in the short term. If policymakers are seeking short term benefits from the "More documentation" policy, then they should not rely on the effects of the policy on defect discovery to drive the improvement.

In the long term, however, the system without the R3 loop active is almost sufficient to drive improvement. Many defects (including a high number of fitment defects) are initially introduced into testing, but as better documentation decreases the number of mismatches, engineers are able to discover more and fix more defects (of all types) earlier.

Figure 37

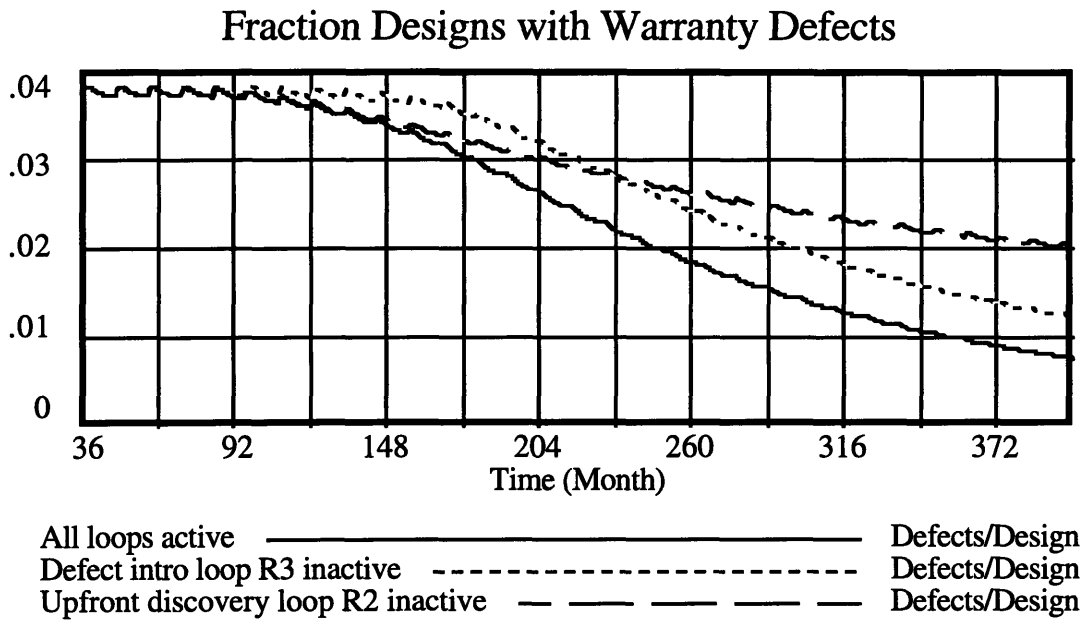



Figure 37 shows the two tests together, allowing comparison. While no strong distinction is appropriate, the behavior of the documentation policy (shown by the run "All loops active") is primarily driven in the short term by reduced defect introduction (shown by the run "Upfront discovery loop R2 inactive") and in the long run by increased defect discovery (shown by the run "Defect intro loop R3 inactive").

*Summary of Documentation Test*

Increasing the documentation of potential mismatches gradually reduces the number of mismatches in the part/print database and increases the quality of designs. It appears to be sufficient in the long term to move the organization from the low quality mode shown in the base run to a high quality mode described by the "Pulse out 1500" run shown in a previous section.

While the effect on mismatches is immediate, the effect on quality is delayed approximately three years, as higher quality designs work their way through the product development cycle and defects in production slowly retire. After the delay, improvement is significant but slow relative to most business decision-making time frames, but this conclusion is sensitive to the rate that designs retire out of production. The initial three year delay is not, however, sensitive to the turnover rate of designs. The improvement that follows the delay is primarily driven in the short term by a reduction in fitment defects introduced (loop R3), and in the longer term by increased defect discovery with upfront tools such as simulation

(loop R2). Testing the two loops showed that the latter effect (early discovery) drives the bulk of the improvement, as well. Thus, the system moves to the high-quality mode sooner and faster when the organization can reap the benefits of fewer mismatches on early defect discovery as quickly and thoroughly as possible. After a long period of increasing rates of improvement, driven by the reinforcing nature of the relationship between defect prevention, defect discovery, and mismatches, (shown in loops R2 and R3) the rate of improvement declines as fewer defects and mismatches remain to be fixed or retired.

### Policy Test -- Fix Mismatches and Contained Defects

There are at least two ways to drain a bathtub: Decrease the inflow by closing the faucet or increase the outflow by opening the drain. The previous policy, "More Documentation," drove improvement principally by decreasing the inflow to mismatches, defects, and contained defects. The second approach, increasing the outflow, deserves consideration as well. The following policy test will help test the second approach, by actively fixing the mismatches that have built up in the system over the years of sequential engineering.

One distinction deserves note. There are two ways to fix a mismatch. One way is to change the print to match the part. For example, assume an engineer learns that operators are belt-sanding the corner off a part but the print does not show such a step as necessary. Thus there is a mismatch. The engineer may decide, perhaps because of lack of time or the high cost of the alternative, to accept that the manufacturing system is producing the part differently than planned, and redraws the print to show the extra step in the process and the rounded corner.

The second way to fix the mismatch is to change the part to match the print. For example, the engineer who discovered the belt-sanded part would decide to submit a design change so that the manufacturing process would produce a part with a rounded corner in the first place. Another scenario is that someone would fix a machine making parts out of tolerance so that it made parts within tolerance, and thus fix a mismatch.

In both cases, the mismatch is repaired. But only in the second case, changing the part to match the print, is a defect or contained defect possibly repaired. The policy examined in this section involves fixing mismatches by, whenever possible, employing the second case -- changing parts to match prints. Certainly, there are many situations where it would be unwise to follow this plan -- reordering an expensive worn-out tool die so the parts it makes match prints, when the part will be retired in the next year, for example. But the

second case is favored because of its potential quality benefit; in many ways changing parts to match prints will "kill two birds with one stone" by both fixing a mismatch and possibly fixing a defect.

Because a policy of actively fixing mismatches is resource-intensive, and programs are often short-lived, the test only lasts three years. Thus, the policy test has a second purpose beyond testing the "fix" policy -- examining the behavior of the system when a policy has a limited duration. One can imagine how a continuing policy like "More Documentation" would drive sustained improvement in quality -- even eight years after enacting, the policy is still in place. But will a three year policy succeed in driving long-term change?

*Policy in the Real World*

In the real world, the policy would entail surveying designs in production, perhaps by checking them at a control point like the mock-up builds. Every time a part is put on a mock-up product and checked with other mating parts, the print for the part would be compared to a part as made in production. When possible, engineers would change the *parts* to match the *prints*, not the other way around.

*Policy in the Model*

The model operationalizes the policy test as a three year program of finding and fixing mismatches, starting in month 60 and ending in month 96. After three years of fixing defects, the policies and parameters in place are returned to the base run levels.

*Policy Runs*

Figure 38

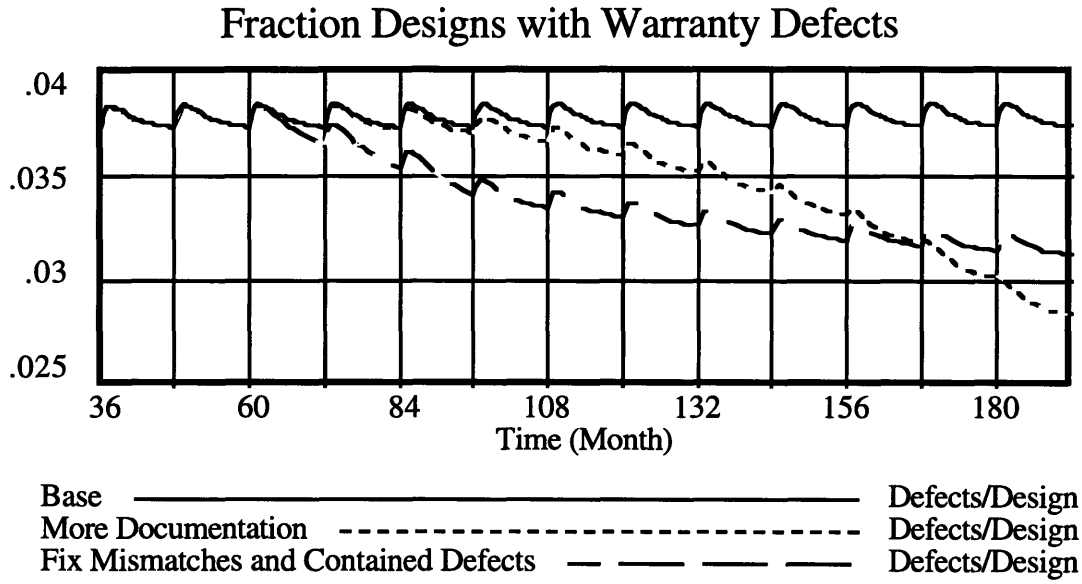



Figure 38 shows a comparison of the documentation policy and the "Fix" policy in the short term. Almost immediately after implementing the fix policy, quality begins to improve; engineers are discovering and fixing contained defects in production. Fewer extra steps in production are causing disruption, fewer "band-aid" fixes are failing to avoid defects, fewer parts are cut to a smaller size to ease assembly, and so on. Also, as engineers fix mismatches discovered at the mock-up builds or using statistical process control tools, they begin increasingly preventing fitment defects and possibly discovering more defects early (other model runs may help explain which of the two dynamics are present). In month 96, the fix policy ends, but quality continues to improve, because designs are coming through product development pipeline with higher quality. When these better designs arrive in production several years after their beginning, quality continues to improve slightly.

Figure 39

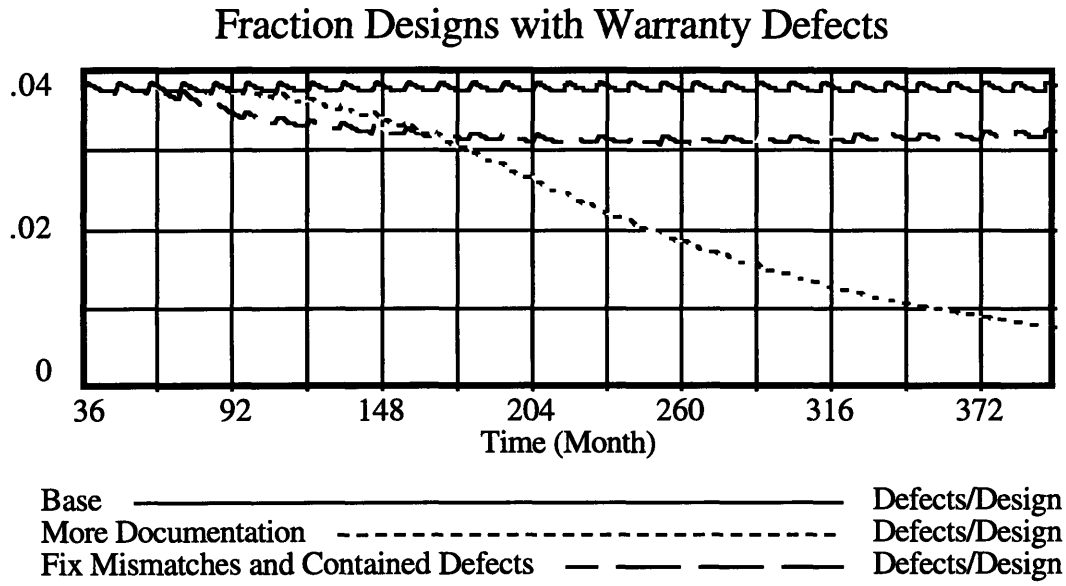



Figure 39 shows the long term behavior of quality. The documentation policy, as described in earlier sections, drives long-term improvement. But the fix policy does not continue to drive improvement in the long term.

Figure 40

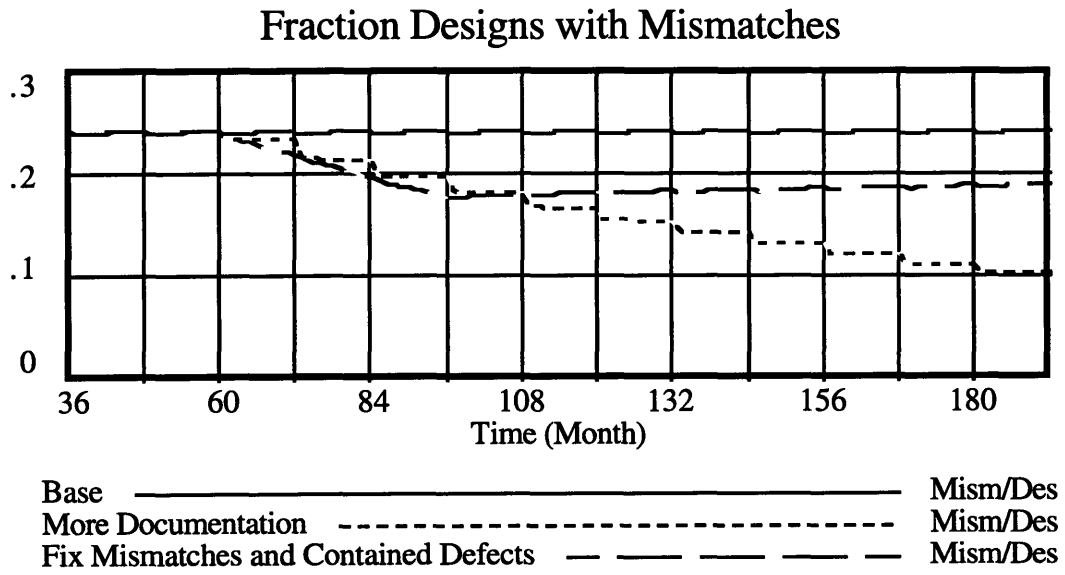



Figure 40 shows that as mismatches are being fixed by engineers, the number of mismatches decreases steadily during the years of the fix policy. In this way, lowering the mismatches in the bathtub by increasing the outflow with the fix policy appears equivalent to reducing the inflow with the documentation policy.

But the benefits of the fix policy do not extend beyond month 96, in contrast to the documentation policy.

Figure 41

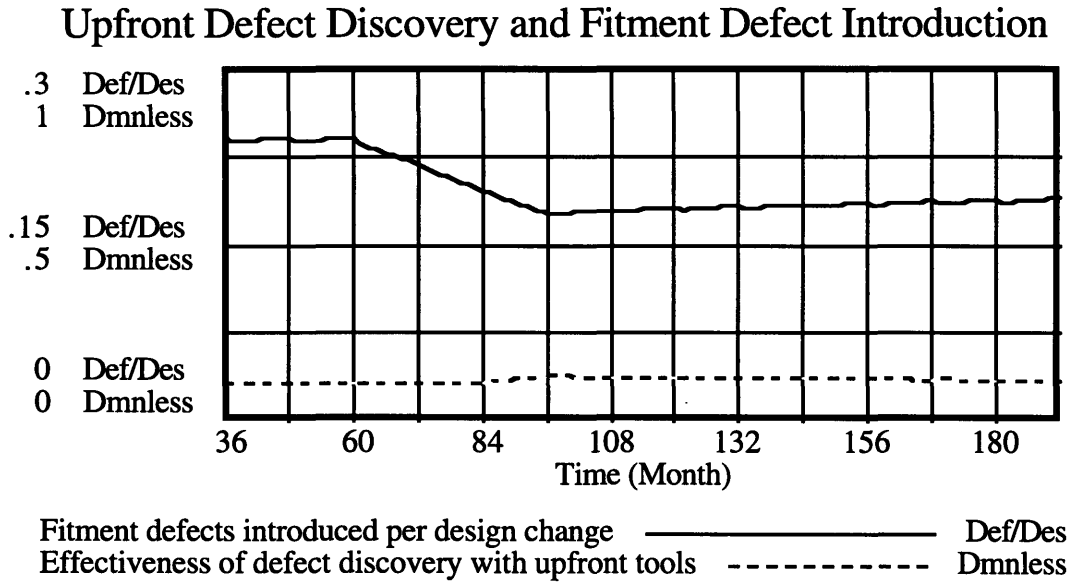



Figure 41 confirms the lack of continuation of improvement. Fixing mismatches reduced the rate of fitment defect introduction, but no longer than the duration of the policy. The bottom checked line shows that the improvement in the part and print database did not spark improvement in the ability of the engineers to discovery defects earlier; the dynamics described by loop R2 did not drive improvement.

The top, solid line shows the results of improvement on fitment defect introduction. Fixing mismatches reduces defect introduction for the duration of the fix policy. After the end of the policy, defect introduction does not continue to improve, confirming that the dynamics illustrated by loop R3 did not drive improvement in the long term.

*Sensitivity to quality effect of contained defects*

One conclusion from the fix test is that the policy has greater immediate benefits than the documentation policy because fixing mismatches by changing parts to match prints will eliminate some warranty-defect-causing contained defects in production. How sensitive is the conclusion to the possibility of a different quality effect from contained defect? What if extra steps to belt-sand parts or force parts on in assembly or polish out marks on parts

never effects quality? Does the conclusion still hold if contained defects do not create actual defects?

In the base runs, it takes twenty contained defects in production to have the effect of one defects (.05 defects/contained defect). The following sensitivity test examines an increase to .1 defects/contained defect and a decrease to 0.

Figure 42

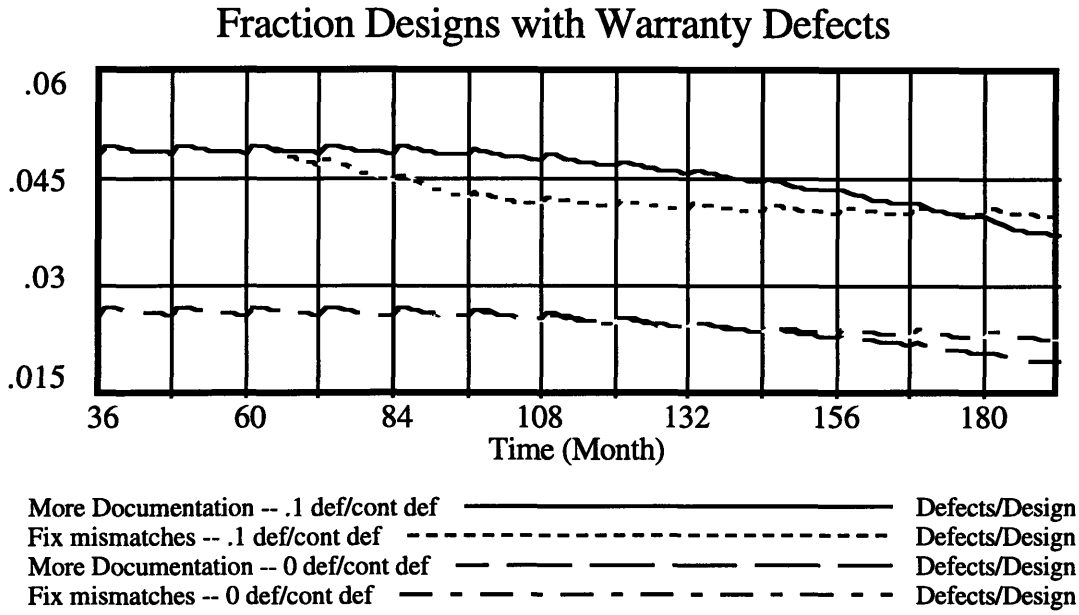



Figure 42 shows the results of the sensitivity test. The top pair of runs has an increased effect on quality; the bottom, no effect on quality from contained defects. The equilibrium conditions are different for the two runs, so the initial values are different.

With an increased quality effect, the immediate benefit of the fix policy relative to the documentation policy is more pronounced and lengthy; the lines cross around month 170, where in the base case they cross around 150 (see figure 38 for comparison). Fixing mismatches clears up contained defects, bringing greater returns in avoiding warranty problems in the short term.

With no quality effect, the fix policy has zero immediate benefit relative to the documentation policy. The short term superiority of the fix policy appears to diminish as the parameter decreases. That said, a policy of fixing mismatches as outlined earlier appears

to be more effective than documenting more in the short term, as long as the effect of contained defects on quality is non-zero.

Eventually, the documentation policy begins to do better, and the gap between the two policies with widen in the long term (the fix policy runs below the documentation policy). But the important conclusion from this test is that the short term benefit of the fix policies is highly sensitive to large reductions in the effect of contained defects on quality.

*Summary of Fix Mismatches and Contained Defects Test*

Fixing mismatches primarily by changing parts to match prints is effective in immediately improving quality. By fixing defects in production that were created over years of a sequential process, warranty defects drop without the three year delay of the documentation policy. This effect could have enormous advantages in policy implementation. Short term feedback of results from the policy to the policymakers could ensure commitment to the policy in future years. The fix policy may benefit from this dynamic in a way that the documentation policy, with its delay, may not. The short-term advantage of the fix policy, however, is highly dependent on a greater-than-zero effect of contained defects on quality.

Turning to examine the effects on mismatches, the fix policy is equivalent to the documentation policy during the first three years. But when the fix policy ends, improvement continues only briefly -- three years of steady mismatch reduction did not spark sustained improvement.

## Policy Test -- Combined policy

The documentation policy brought delayed, slow, sustainable improvement. The fix policy brought immediate, limited improvement. As of yet, we tested the one policy against the other, in order to recognize that both policies cost money and time and with the likely resource constraints in an organization, it may be necessary to choose between the two policies. But now model runs will present the effects of a combination of the two policies.

*Policy in the Real World*

The two policies combined here are not mutually exclusive. One policy involves a survey of parts in production; the second, a better documentation policy.

*Policy in the Model*

The policy as tested is simply the identical policies together.

*Policy Runs*

Figure 43

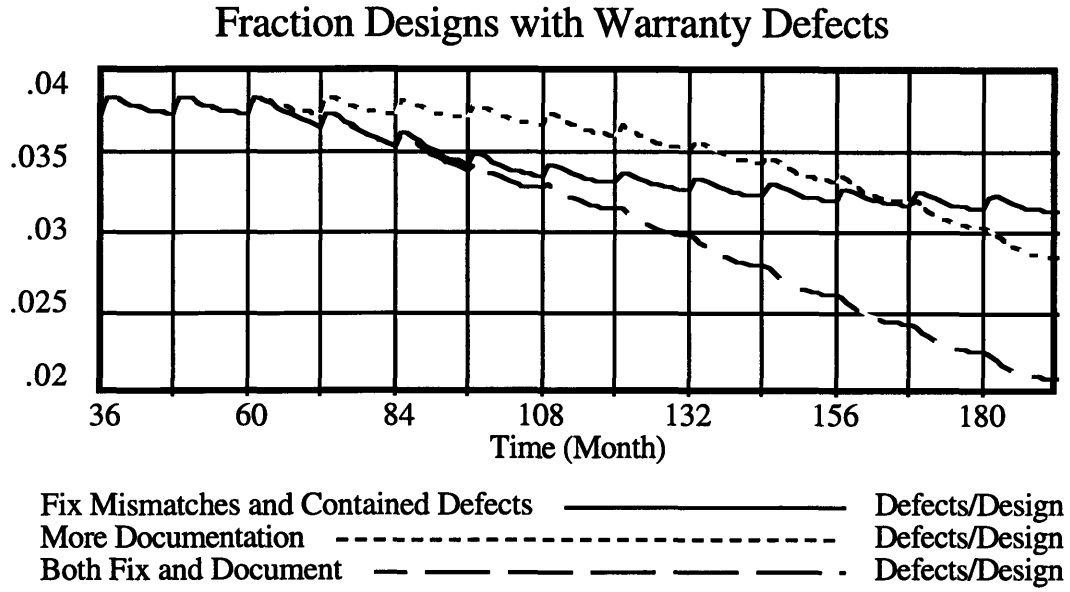



Figure 43 shows the effects of the two policies separately, and then together, on quality. The top two graph lines have been presented previously in Figure 38. Under the combined policy (the lowest, long-dashed line), quality immediately improves for the first few years after the policy begins in month 60 as fixing mismatches also fixes some defects in production. Engineers are documenting potential mismatches better, but the results are not visible in the *Fraction designs with warranty defects*. By month 96, the combined policy is clearly better than the fix policy alone, as more and more designs with fewer fitment defects come into production -- these latest designs have been created in an organization that had fewer mismatches. Improvement continues as the second feedback loop (loop R2) activates more strongly; with a more accurate database, engineers are able to find more defects earlier. After ten years of the policy in place, the combined policy has brought a significant reduction in warranty defects.

Figure 44

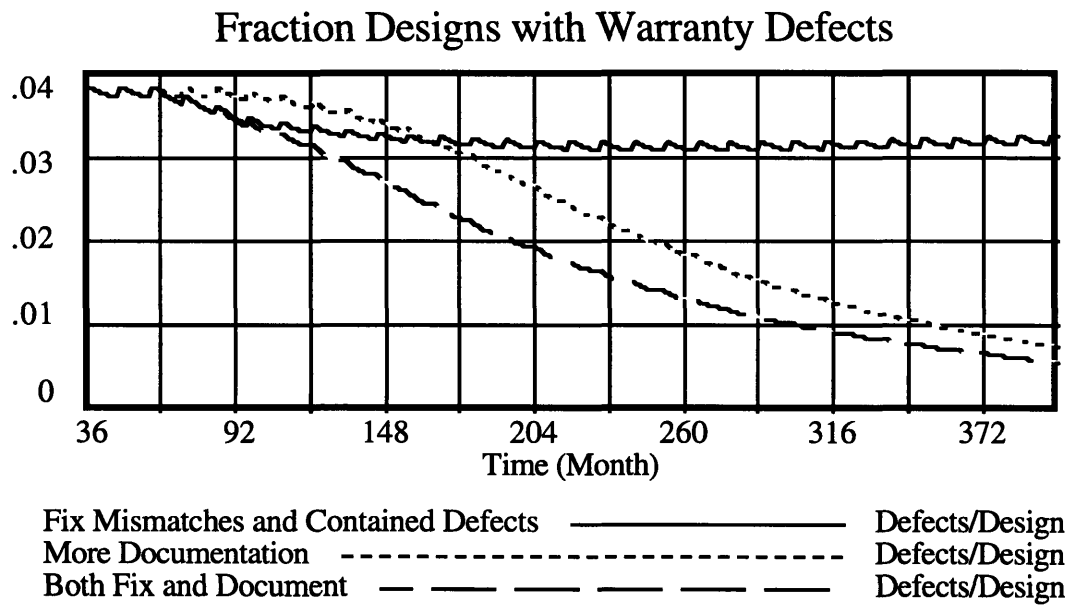



Figure 44 shows a longer term version of the previous figure. Much like the documentation policy, the combined policy drives decreasing rates of improvement in the long term. There are not as many mismatches, defects, and contained defects left to fix as time goes on, so engineers fix them and the system retires them more slowly. In the longer term, the combined policy fares little better than the documentation policy alone.

Figure 45

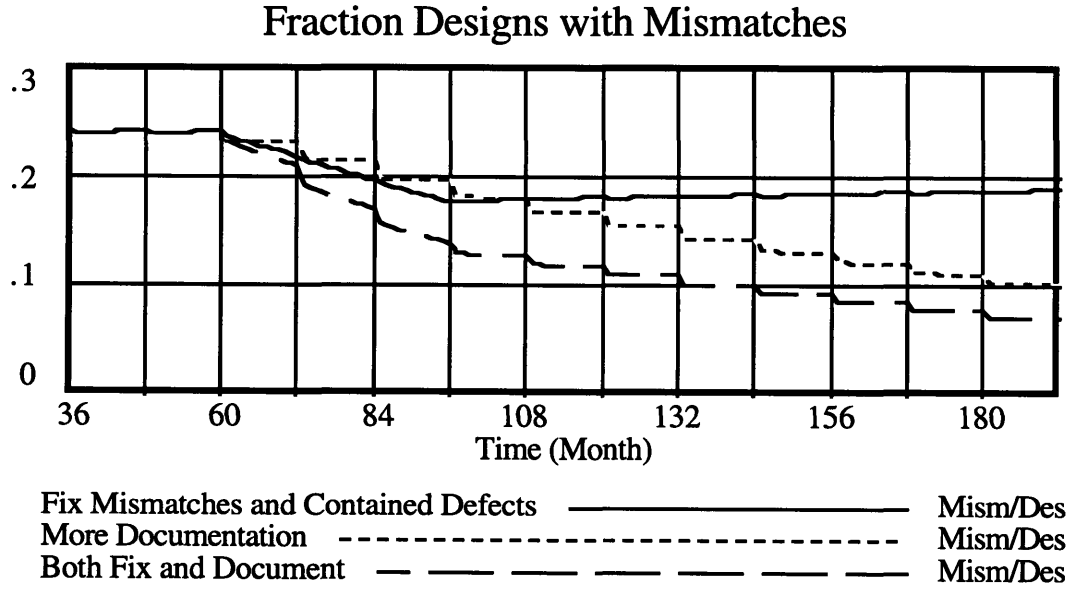


Figure 45 shows the effect of the combined policy on mismatches. For the first three years, the fix policy and the documentation policy are approximately equivalent. Together, however, the number of mismatches falls more steeply as engineers' documentation means that fewer mismatches are being created, plus engineers' efforts means that more are being fixed. After month 96, the end of the fix policy, improvement in the number of mismatches continues under the combined policy as increased defect prevention and increased discovery by engineers continue to reduce mismatch creation. The rate of improvement falls in the latter years as most of the mismatches have been fixed.

Figure 46

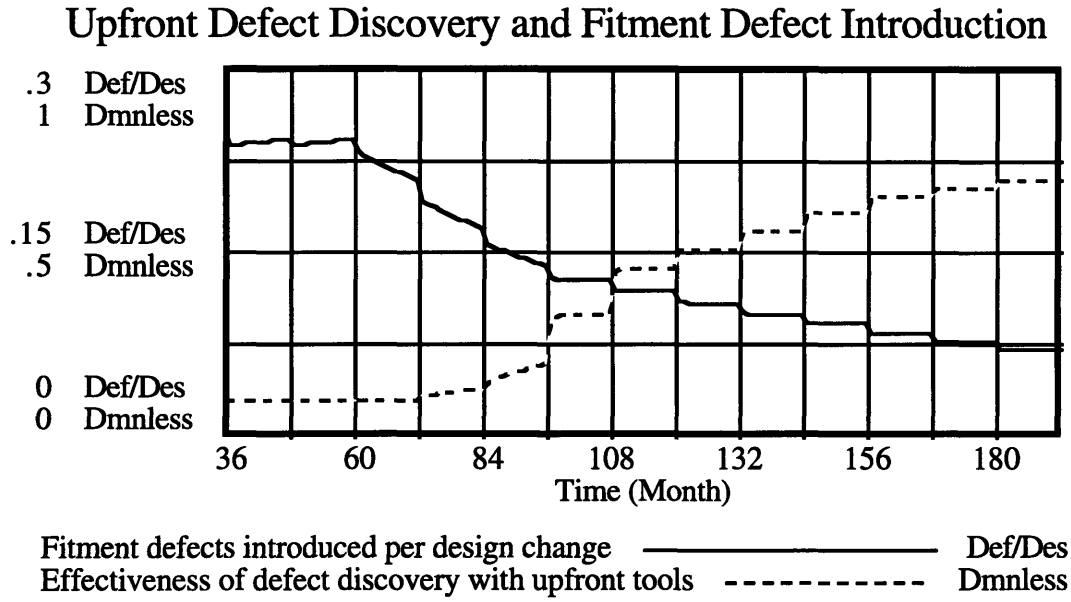



Figure 46 shows the behavior of two key variables under the combined policy.

*Fitment defects introduced per design change* falls steeply in the short term -- engineers are fixing mismatches aggressively, so other engineers introduce fewer defects. Improvement continues after month 96 as mismatches continue to fall, as explained in the previous paragraph.

*Effectiveness of defect discovery with upfront tools* improves minimally for three years, but by month 96 the number of mismatches has fallen to the point where engineers are able to benefit from the use of simulation tools and rely less on the later builds to find defects. Between month 96 and 120, three and six years after the policy is enacted, they leap from finding 20% of defects up front to finding 48% of defects early. Over the following years,

that percentage rises to 72%. In time, engineers reach their goal of defect prevention and early discovery.

*Summary of Combined Test*

Fixing mismatches and contained defects and increasing documentation of potential mismatches has a significant positive effect on quality. In the short term, fixing the legacy of old defects and contained defects helps quality immediately and reduces the number of mismatches. In the longer term, the documentation policy continues to drive the reduction of mismatches, which sparks more defect prevention and early defect discovery, helping increase quality and decrease mismatches even more.

Two reinforcing feedback loops contribute to the long term improvement and drive the organization from the low quality "base case" mode to a higher quality mode. Loop R3, the defect introduction loop, drives improvement immediately, and most strongly for the first three years after policy implementation, as engineers prevent defect creation. Loop R2, the upfront discovery loop, drives significant improvement after a three year delay, as the part and print database become coordinated enough that engineers are able to effectively use upfront simulation tools to discover and fix defects early in the product delivery process.

## Policy Implementation

This section presents insights into policy implementation based on ideas from managers at LaunchTech, observation of LaunchTech operations, and the modeling work.

### More Documentation

To implement the "More Documentation" policy at LaunchTech, the company could assign high-level engineers the task of checking that workers change the blueprints to reflect changes made to the production line. Setting such a "control point," coupled with educating workers about the importance of the policy, is likely to increase the amount of documentation.

Secondly, the company could invest resources in design engineering to increase the rate of processing print revisions. Possible resources are either new people hired or new people assigned to the task.

### Fix Mismatches

To implement the "Fix Mismatches" policy, the company could set a point in the product delivery process when engineers would check that parts match prints. A likely point is the "mock-up" process in design engineering. When engineers make parts to check how well the new parts fit with existing parts, they could exclusively use parts produced in the actual manufacturing facility, not the prototype parts specially made for testing. If the production parts received from Manufacturing do not match the prints on file, the engineers could correct the mismatch at this point.

### ISO 9000

Particularly to enable sales in overseas markets, LaunchTech is seeking ISO-9000 certification. This certification requires rigorous labeling and standardization of processes, and would shift the information storage system away from informal modes of documentation.

At LaunchTech, ISO certification may help drive the implementation of the policies described above. Certification does not require that a company fix past problems, it requires only that it put processes in place to improve into the future. Thus, ISO will likely drive implementation of the "More Documentation" policy more than the "Fix Mismatches

and Contained Defects" policy, and its effect on quality via decreasing mismatches will be subject to the long time delays described earlier.

The high-leverage policies considered here are not radical or excessively complex. Common sense could drive someone to suggest them. Thus, it may seem surprising that such policies had not been successfully implemented in the past. Why haven't managers thought of these policies before?

Because, in most cases, they did think of the policies before, but ran into significant barriers to implementation. At LaunchTech, for example, there are many disincentives to fixing or documenting a mismatch.

### Disincentives to fixing or documenting a mismatch

Various people have identified several reasons why a mismatch may not be corrected. They include: liability issues, increased analytical work, the "snowball effect" of corrections, retrofitability, and economic judgment.

One disincentive regards liability issues. In the case of a part produced by an external company, manufacturing the part differently from the supplier's specifications may legally implicate the company with responsibility for any problems with the part in the future. An engineer explained,

> You buy a bearing and manufacture the mating shaft. The bearing supplier publishes specifications for the size of the shaft, but the shop makes the shaft a little undersized to ease assembly. Now the shaft as made is not to print, and when you discover that later, you ask manufacturing to change the part. Manufacturing says 'We've always done it this way.' So you ask design engineering to change the print away from the supplier's recommendations, but you can't do that for liability reasons. The result is that you just don't inspect the parts nor document too closely. In this way, what you don't know apparently can't hurt you.

A second reason that someone might not choose to change the print to match the part is that, as one person said, "If you change the design to match the part in production, then you ought to do all the engineering analysis all over again to make sure that the part as made actually works." Design engineers do not have the time to do such work, so they perceive themselves to be better off with the part not matching exactly.

A third disincentive is what one person called "the snowball effect of changing prints to match parts." If an engineer changes the print of a large, integral part that has many mating

connections to match the part as made in production, it may lead to changing multiple other prints, too. The design engineering organization may not have the time or capacity to fix such a large number of prints in a short period of time.

A fourth disincentive is that correcting mismatches may require LaunchTech to begin manufacturing a new part for current production, but also require that they continue manufacturing the old part to sell to customers repairing their products. Parts would not be so interchangeable. An engineer explained,

> When two parts mate and you find that one or both parts mismatch the print, you can't change just one. You change both and they mate fine with one another. But you don't have interchangeability. You can't use old part X and new part Y. You then have a retrofit problem. Say a guy in the field breaks part X and wants to replace it. But now new part X doesn't fit with old part Y.

The most powerful disincentive is that making a design change can be expensive, so often times, it is not a wise economic and institutional decision to correct a mismatch. Some problems are better just coped with, so people often make an active decision to not make a change.

## Conclusion

I have developed a theory that explains how a firm with advanced CAD/CAM technologies and a well-developed concurrent methodology can operate in a low-quality mode. Results show how the tools prescribed for high-quality product development are insufficient to drive improvement without a unified database of blueprints and manufactured parts.

This research further shows the difficulties of managing the improvement of product development. As the case study outlines, intelligent and rational behavior in a changing environment created the low-quality mode of operation at LaunchTech. Additionally, the policy implementation section of the thesis describes powerful institutional disincentives to improvement.

However, the more compelling reason that the system is difficult to manage is because its behavior is counter-intuitive. Because of non-linear relationships that span diverse parts of the organization, one can not accurately predict the results of decisions. For example, introducing design changes into testing late in the process does not just temporarily hurt product quality, it also helps create mismatches between parts and prints that produces more defects in the next year and damages product quality ever after. A competent, well-trained manager would likely find it difficult to integrate the complex dynamics of this system into her decision to approve or not approve a proposed late design change.

The limits to policy-making in complex systems are not, of course, limited to people managing product development. Research in dynamic decision-making has revealed how our mental models of complex environments are generally poor. In particular, people do not account well for feedback loops, time delays, accumulations, and nonlinearities (Sterman 1994).

Leaders in process improvement, and in all policy-making areas, would benefit from increasing their ability to understand the long-term, system-wide effects of their actions within the complex systems they manage. The challenge for future research will be to create ways to help managers build such skills and learn to apply them to their everyday decisions.

## Model Documentation

The purpose of this section is to provide supporting documentation for the system dynamics model presented in this paper.

The model contains seven sectors: Designs, Defects, Open Concerns, Effectiveness of Early Discovery, Mismatches, Totals, and Simulation Control. Each sector has a brief introduction, but most of the model documentation is in the "comment" field within each equation listing, which is in the line below the units for the variable listed. Most variables can be seen in the small diagrams within each sector. When no diagram is shown for an equation, the equation is similar to structures diagrammed elsewhere in the documentation. In general, the formulations draw upon established system dynamics models of the firm (Forrester 1961) and models more specific to quality improvement (Sterman, Repenning and Kofman 1994).

The code for the model is written using Vensim modeling software, version 1.62-8, available from Ventana Systems. The stock and flow diagrams and causal loop diagrams used throughout this paper were created with the Vensim software.

The equations are reproduced exactly as used in the simulation model. An example is shown below:

```
New defects intro rate = Design intro rate*Total defects per design introduced
    ~ Defects/Month
    ~ Defect introduction is driven by the introduction of new designs every year.
```

The first line shows the actual equation from the model. The new defects introduction rate is equal to the design introduction rate times the total defects per design introduced. The second line shows the units of measure for the initial variable. The third line provides comments that may supplement the equation with more intuition about the idea behind the equation, or may offer a source for the equation.

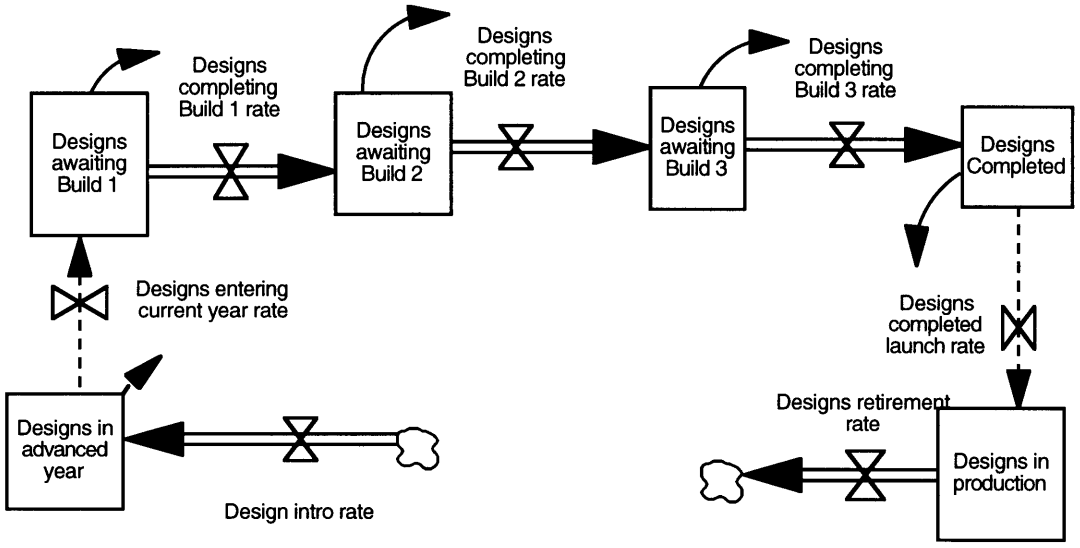
### Designs

The purpose of this sector is to drive the movement of design tasks through various stages over the three years of advanced work, current work, and production. Entry into and exit from the current year (shown as dotted lines in Figure 47) as well as the retirement of

designs are discrete "flushes," in which the entire contents of the stock exits the stock in one time step. Each build is simplified as a first order drain.

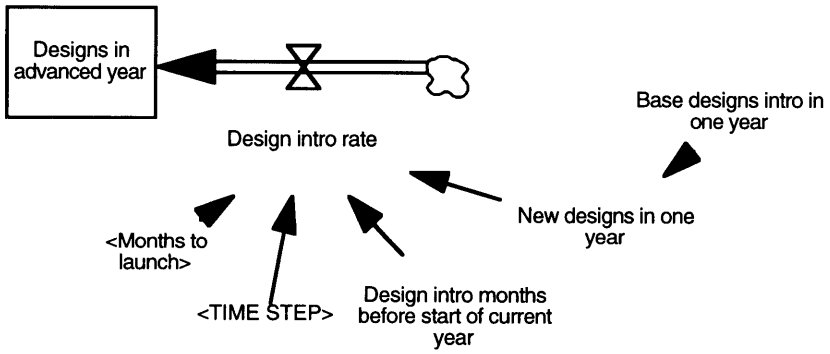
One simplification is that designs enter the product development pipeline only at the beginning of the process. Further, no designs ever get canceled. The model does not account for allocation of resources, so the timing or speed of the builds is not dependent on available engineers' time, nor on access to the production line.

Figure 47




The purpose of the Figure 47 is to show the broad lay-out of the sector. The equations below roughly follow the chronology of a design through the system, clockwise through the figure, starting at the bottom left of the figure.

Figure 48



All *New designs in one year* are assumed to enter the advanced year testing 18 months before launch, in a single time step. Design introduction is assumed to be steady every year; under the standard runs the number of designs introduced do not vary.

Designs in advanced year = INTEG(Design intro rate-Designs entering current year rate,0)
~ Designs
~ The number of designs in advanced year work. Advanced year is 12 to 24 months before launch.

Design intro rate = if then else(Months to launch=
Design intro months before start of current year, New designs in one year/TIME STEP,0)
~ Designs/Month
~ Designs are either new or changes to previous designs. New designs are flushed in every year.

New designs in one year = Base designs intro in one year
~ Designs
~ This intermediate variable allows for other designs to potentially be added.

Base designs intro in one year =1000
~ Designs
~ Based on an estimate of 800 - 1000 per year.

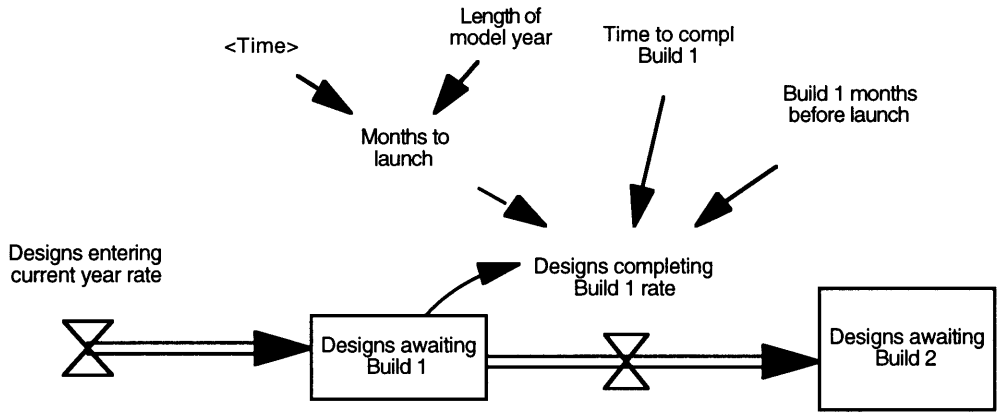
Base designs intro in one year f ([(0,0)-(1000,1000)],(0,0)
,(240,1000),(1000,1000) )
~ Designs
~ Table not used at present. Could be used to test a variable number of designs.

Design intro months before start of current year = 6
~ Months
~ Assumes defect discovery and correction begins 18 months before launch.

Designs entering current year rate = if then else(Months to launch=Months before launch that model year changes
,Designs in advanced year/
TIME STEP,0)
~ Designs/Month
~ Designs go from the advanced year to the current year in a discrete flush. All designs travel in one dt. Operationally unrealistic, but necessary to capture the changing of the model year.

Months before launch that model year changes =12
~ Months
~ Model year changes every year

Figure 49




The outflow rate is a first order drain. Its rate is the designs in the stock divided by a time to complete the build. All the designs pass through the flow starting at the month indicated by *Build 1 months before launch.* Once a design passes through the rate, it arrives in a stock of designs awaiting the next build.

Each of the outflows in this sector has an identical structure, as shown in Figure 49. Figure 49 can support the next four structures listed in this sector.

Designs awaiting Build 1 = INTEG(Designs entering current year rate
-Designs completing Build 1 rate,0)
~ Designs
~ Number of designs awaiting Build 1. Build 1 corresponds to the "Manufacturing Try-Out" or "Design-Intent" builds at LaunchTech.

Designs completing Build 1 rate = if then else(Months to launch<Build 1 months before launch:AND:Months to launch>5,Designs awaiting Build 1/Time to compl Build 1,0)
~ Designs/Month
~ The "and" structure is necessary so that build 1 happens only at its assigned time during the year.

Time to compl Build 1 = 0.5
~ Month
~ Assumes the bulk of the builds happen over a two week span.

Build 1 months before launch = 10
~ Months
~ May. This date has varied in the past years from 7 to 11.

Length of model year = 12
~ Months
~ The twelve months run from July to the end of June.

Months to launch = Max(Length of model year-MODULO(Time,Length of model year),0)
~ Months
~ Structure starts at 12 in July and declines linearly over time.

The equations for the next stock are similar in structure for the equations for the previous stock, as shown in Figure 49. Thus, a diagram is not included.

Designs awaiting Build 2 = INTEG(Designs completing Build 1 rate
-Designs completing Build 2 rate,0)
~ Designs
~ Build 2 corresponds to the Pilot build at LaunchTech.

Designs completing Build 2 rate = if then else(Months to launch<Build 2 months before launch,
Designs awaiting Build 2/Time to compl Build 2,0)
~ Designs/Month
~ The build starts at the month indicated in Build 2 months before launch, and ends when all the designs have left the stock.

Time to compl Build 2 = 0.5
~ Month
~ Same length at build 1.

Build 2 months before launch = 5
~ Months
~ February, given launch in July.

For a diagram, consult Figure 49.

Designs awaiting Build 3 = INTEG(Designs completing Build 2 rate-Designs completing Build 3 rate,0)
~ Designs
~ Build 3 corresponds to the Pre-Production build at LaunchTech.

Designs completing Build 3 rate =if then else(Months to launch<Build 3 months before launch,
Designs awaiting Build 3/Time to compl Build 3,0)
~ Designs/Month
~ The build starts at the month indicated in Build 3 months before launch, and
ends when all the designs have left the stock.

Time to compl Build 3 = 0.25
~ Month
~ Pre-prod builds are more compacted in time than the others, so the time is shorter.

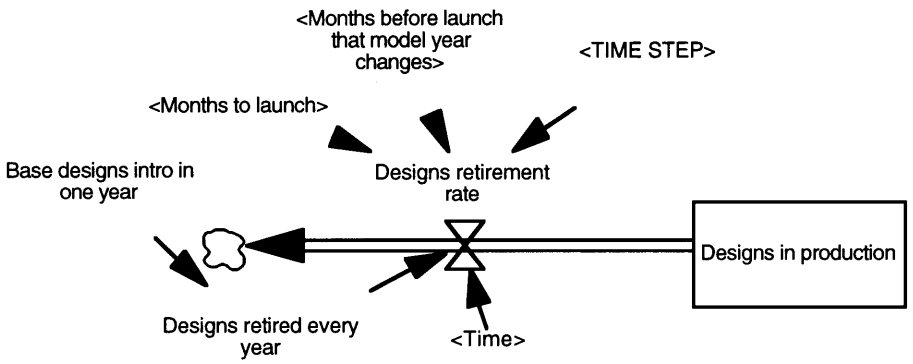
Build 3 months before launch = 2
~ Months
~ May.

For a diagram, consult Figure 49.

Designs Completed = INTEG(Designs completing Build 3 rate
-Designs completed launch rate,0)
~ Designs
~ Designs that have completed the builds and are awaiting launch.

Designs completed launch rate =if then else(Months to launch=
Months before launch that model year changes,
Designs Completed/TIME STEP,0)
~ Designs/Month
~ At launch, all designs are "flushed" into production.

Figure 50




Unlike the other flows in this sector, the rate of retiring designs is not formulated as a draining function. Instead, when the model year changes, the number of designs that were introduced into the system get retired out of the system, over a single time step. The model assumes that the number of designs leaving production is equal to the number entering it.

Designs in production = INTEG(Designs completed launch rate-Designs retirement rate
,Initial designs in production)
~ Designs
~ Includes all the active designs in production.

Initial designs in production = 10000
~ Designs
~ Based on an estimate by an engineer. This value changes to test the sensitivity to the retirement rate.

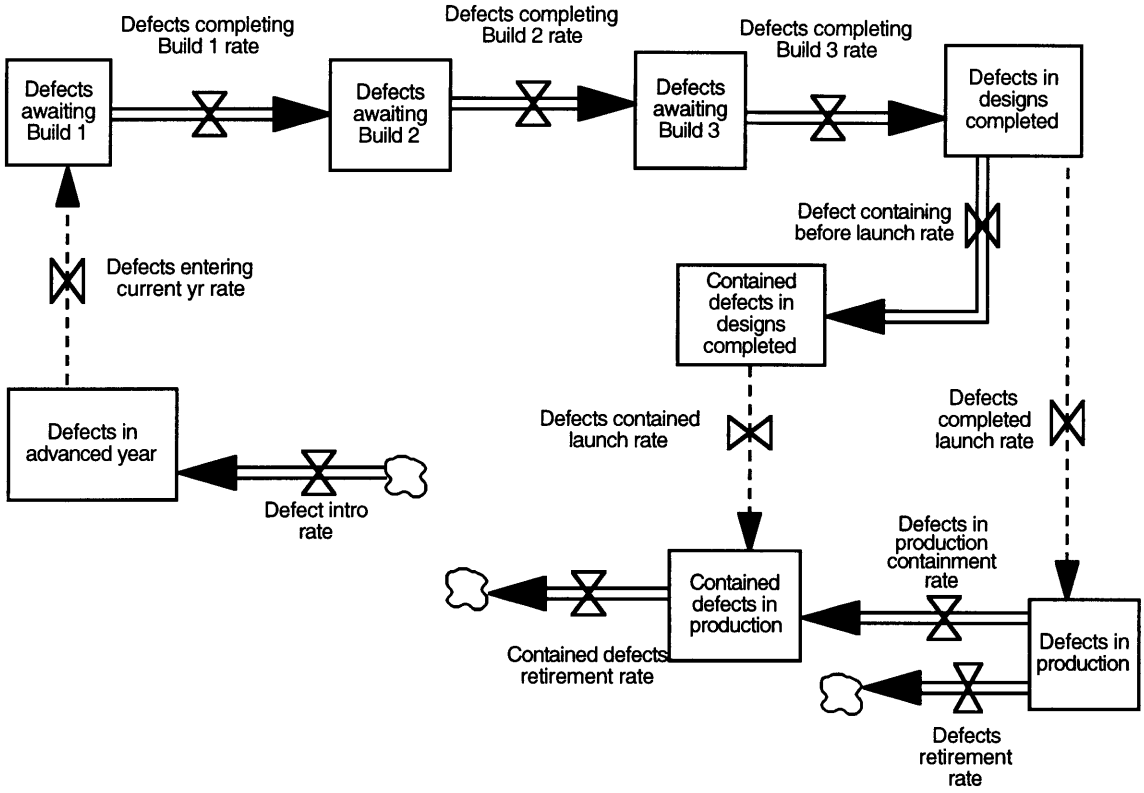
Designs retirement rate =if then else(Months to launch=
Months before launch that model year changes:AND:Time>18,
(Designs retired every year)/TIME STEP,0)
~ Designs/Month
~ Designs leave the stock of designs in production as they obsolesce or make way for new incoming designs. The Time>18 structure ensures that designs are not retired until the first production year, which happens in month 12.

Designs retired every year = Base designs intro in one year
~ Designs
~ This ensures a steady state of designs.

## Defects

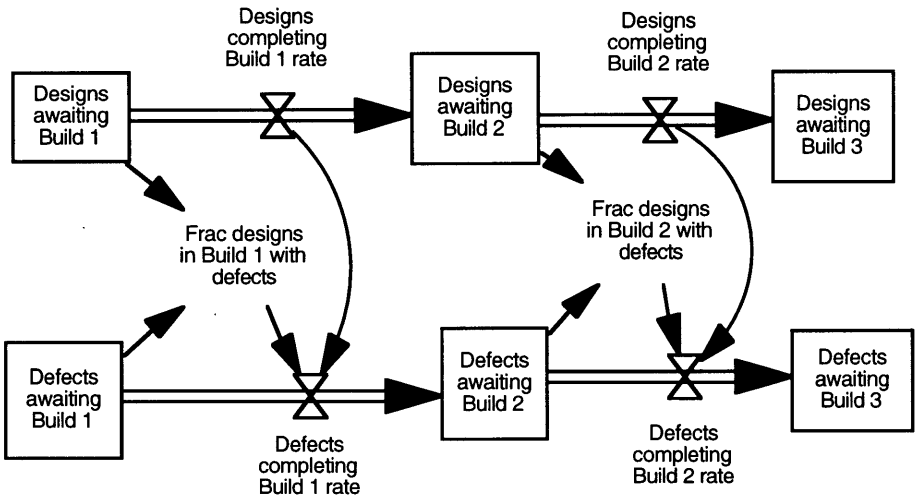
This sector tracks the number of defects in the various stages of design testing and completion. Defects include three types:design/fitment, assembly and manufacturing/tooling.

Figure 51



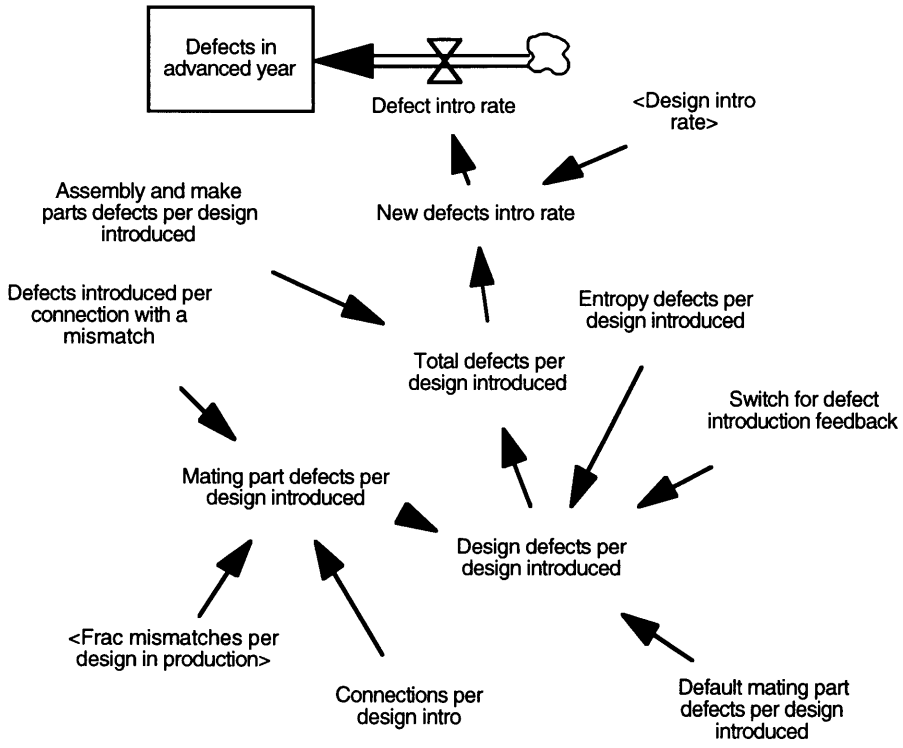

The purpose of the diagram shown above is to provide a broad overview of the sector. Parallel in structure to the design sector, defects move in one direction from stock to stock. Defects remain in the advanced year testing until the current year, when they enter the builds. First build 1, then build 2, then build 3, then they are completed and enter production. Defects leaving *Defects in designs completed* that do not enter production can be contained and thus enter an intermediary stock, *Contained defects in designs completed*, before entering production. Multiple outflows and inflows are not shown in Figure 51, thus defects are not conserved.

Figure 51.5




All transfers of defects from stock to stock is driven by the movement of designs from stock to stock, using a co-flow structure. One small piece of the coflow structure is shown above, as an example.

Figure 52




Introduction of defects is modeled as the number of designs introduced times the number of defects for every design. Defects come from multiple sources: design defects (including entropy and mating part), assembly defects and make parts (or tooling) defects. The total

number of defects per design is the sum of the three types. This structure includes a switch that can set mating part defects either to a default setting, or to a dynamic setting. The dynamic number of mating part defects is formulated as the number of connections between mating parts that include a mismatch times the chance that a connection with a mismatch actually introduces a defect. For more information on the formulation, see the section titled *Effects of mismatches -- Fitment defect introduction* in the Dynamic Hypothesis section.

The formulations assume that mismatches do not effect the creation of assembly and make part defects; nor do they increase the number of random, or entropy, defects. The default or initial values for the number of defects per design are rough estimates.

Defects in advanced year = INTEG(Defect intro rate-
Defects entering current yr rate+Defects intro during adv year rate-
Defects during adv year fix rate,0)
~ Defects
~ Number of defects in the designs in the advanced year.

Defect intro rate = New defects intro rate
~ Defects/Month
~ Includes defects introduced for the new model year changes.

New defects intro rate = Design intro rate*Total defects per design introduced
~ Defects/Month
~ Defect introduction is driven by the introduction of new designs every year.

Total defects per design introduced =Design defects per design introduced+
Assembly and make parts defects per design introduced
~ Defects/Design
~ Three sources of defects

Assembly and make parts defects per design introduced = 0.15
~ Defects/Design
~ This includes two types of defects. Rough estimate.

Design defects per design introduced = ((1-Switch for defect introduction feedback)*
Default mating part defects per design introduced)+(Switch for defect introduction
feedback*Mating part defects per design introduced) +Entropy defects per design introduced
~ Defects/Design
~ Design defects include mating part, or fitment defects and entropy defects. Inclusion of a switch allows for testing of loop strength.

Default mating part defects per design introduced = 0.24
~ Defects/Design
~ Rough estimate.

Mating part defects per design introduced =
Connections per design intro*Defects introduced per connection with a mismatch*
Frac mismatches per design in production
~ Defects/Design
~ Same as fitment defects.

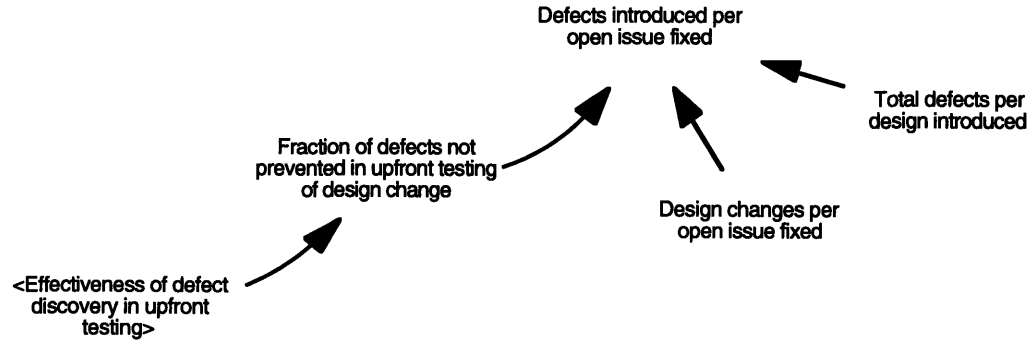
Defects introduced per connection with a mismatch = 0.25
~ Defects/Connection
~ If there is a connection between two parts, and one of the parts is mismatched, this is the probability that a defect will be introduced. Rough estimate from an engineer.

Connections per design intro = 4
~ Connections/Design
~ Rough estimate from an engineer.

Entropy defects per design introduced = 0.1
~ Defects/Design
~ Includes design mistakes not directly related to mating part or fitment issues.

Switch for defect introduction feedback = 1
~ dimensionless
~ 1 means on, 0 means off.

Figure 53



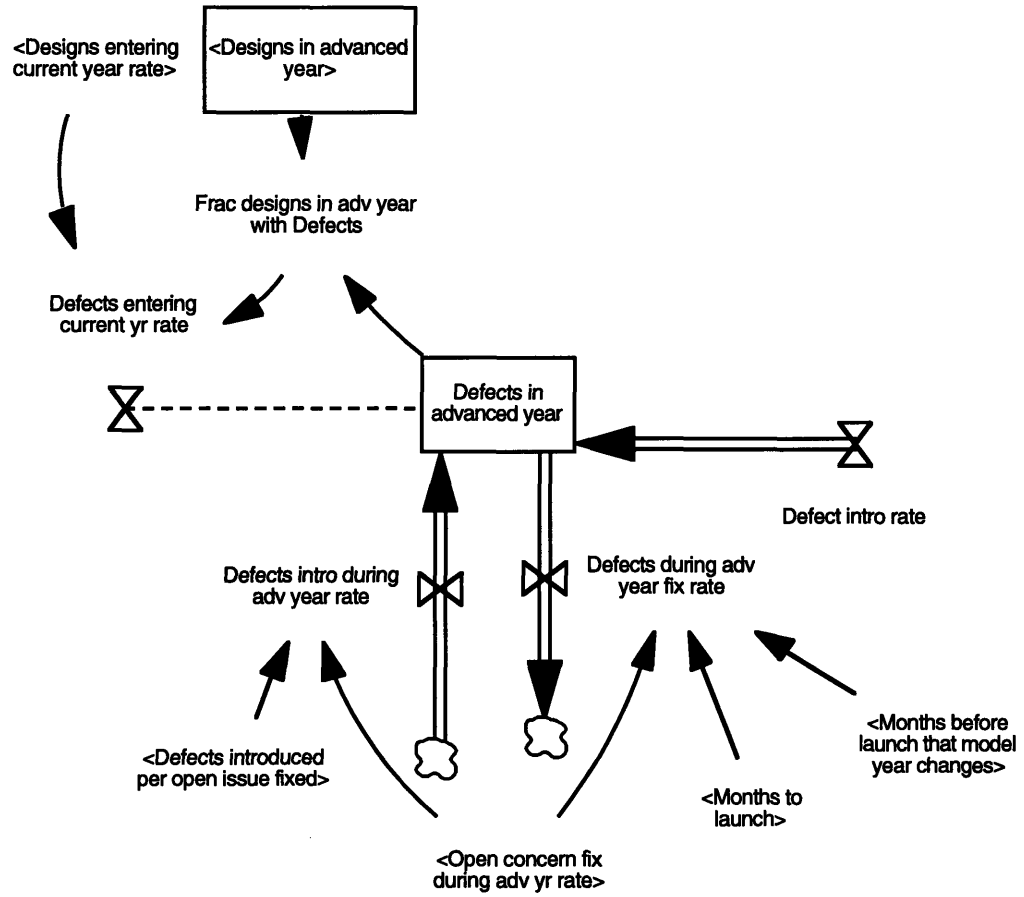

Defects introduced with every fix of an open issue is formulated as a standard multiplication of fractions, with an additional factor to account for the defects discovered and fixed by upfront testing in design engineering. Engineers will catch a certain fraction of defects, so those defects will not be introduced.

Defects introduced per open issue fixed = Total defects per design introduced*
Design changes per open issue fixed*
Fraction of defects not prevented in upfront testing of design change
~ dimensionless
~ Includes defects introduced for rework changes which fixed other open issues. Often called systems engineering effects. Some defects are prevented in upfront testing, although that dynamic is not explicitly modeled with defect discovery, creation of an open concern, and a fix like the other defect dynamics are modeled.

Design changes per open issue fixed = 1
~ Design/Defect
~ Every fix requires a design change submitted.

Fraction of defects not prevented in upfront testing of design change =
1-Effectiveness of defect discovery in upfront testing
~ dimensionless
~ Undiscovered defects.

Figure 54




The structure shown in Figure 54 regulates the number of defects in the advanced year. As shown in earlier formulations, the horizontal introduction inflow and the outflow to the current year are both driven with a coflow structure by the flow of designs through the various stages. The structure in the top left of Figure 54 is similar in structure to the formulations in Figure 51.5. The outflow rate of defects is equal to the outflow rate of designs times the fraction of defects per design.

The vertical outflow rate is driven by the rate of fixing open concerns. Every fix of an open concern fixes a defect. During the model year change, the fix rate is set to zero because the discrete flush of defects out of the stock will empty the entire stock in one time step. Any additional outflow such as through fixing would send the stock negative.

The vertical introduction rate is driven by the rate of fixing open concerns, as well. As outlined by loop R1 in Figure 7, fixing defects introduces new ones. The rate is formulated as the open concern fix rate times the number of defects introduced for every open concern fixed.

Defects intro during adv year rate =Open concern fix during adv yr rate*
Defects introduced per open issue fixed
~ Defects/Month
~ Defects that get created as a result of another fix.

Defects during adv year fix rate =if then else(Months to launch=
Months before launch that model year changes,0,
Open concern fix during adv yr rate)
~ Defects/Month
~ During the model year change, the fix rate is set to zero because the discrete flush of defects out of the stock will empty the entire stock in one time step. Any additional outflow such as through fixing would send the stock negative.

Defects entering current yr rate = Designs entering current year rate*
Frac designs in adv year with Defects
~ Defects/Month
~ Part of a coflow.

Frac designs in adv year with Defects = zidz(Defects in advanced year,
Designs in advanced year)
~ Defects/Design
~ Part of a coflow. zidz means zero if division by zero, and is necessary because designs in advanced year can reach zero.

Figure 55

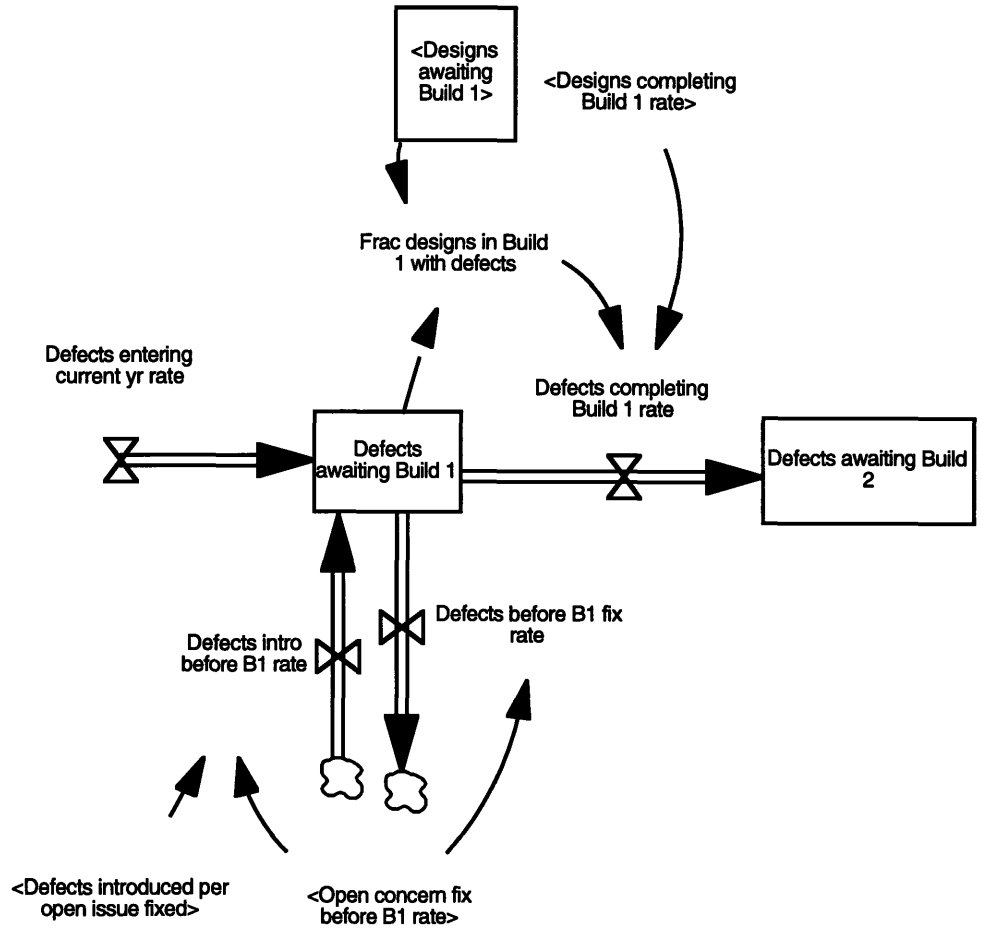



Figure 55 shows a structure that is analogous to the structure shown in Figure 54. For a description of the formulations, consult the text accompanying the previous figure.

Defects awaiting Build 1 = INTEG(Defects entering current yr rate-
Defects before B1 fix rate-Defects completing Build 1 rate+
Defects intro before B1 rate,0)
~ Defects
~ Number of defects in the designs that are awaiting the first build.

Defects intro before B1 rate = Open concern fix before B1 rate*
Defects introduced per open issue fixed
~ Defects/Month
~ Introduction of defects through "systems engineering effects" whereby fixing one defect introduced another one.

Defects before B1 fix rate =Open concern fix before B1 rate
~ Defects/Month
~ Fixing an open concern will cause this flow to reduce the number of defects in the stock.

Defects completing Build 1 rate =Frac designs in Build 1 with defects*

Designs completing Build 1 rate
~ Defects/Month
~ Defects in designs pass through the build. Part of a co-flow.

Frac designs in Build 1 with defects =zidz(Defects awaiting Build 1,
Designs awaiting Build 1)
~ Defects/Design
~ Part of the co-flow structure by which the movement of designs drives the movement of defects. Part of a coflow. Zidz means zero if division by zero, and is necessary because designs can reach zero.

The following structure has no diagram, because it is analogous to the previous structures shown in Figures 54 and 55.

Defects awaiting Build 2 = INTEG(Defects completing Build 1 rate
-Defects completing Build 2 rate-Defects before B2 fix rate+
Defects intro before B2 rate,0)
~ Defects
~ Awaiting the pilot build.

Figure 56

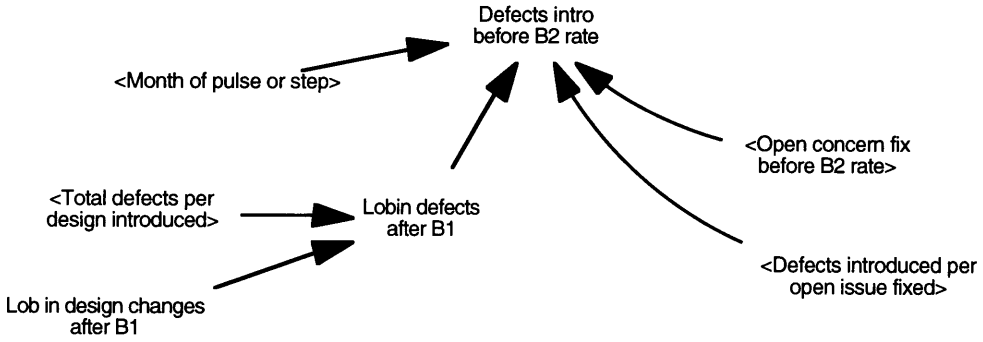



Figure 56 illustrates one change to the defect introduction rate for defects awaiting build 2. To incorporate the "lobin" test, the formulation includes *Lobin defects after B1*, which is determined by the number of design changes times the fraction of defects per design introduced. The variable, Month of pulse or step, determines when the test occurs.

Designs added through the test are not added to the total number of designs accounted for in the designs sector. Instead, only the defects that accompany the designs are included.

Defects intro before B2 rate = (Defects introduced per open issue fixed*
Open concern fix before B2 rate)+(Lobin defects after B1*pulse(Month of pulse or step,1))
~ Defects/Month
~ Includes a "lobin" test.

Lobin defects after B1 = Lob in design changes after B1*
Total defects per design introduced
~ Defects/Month
~ Part of a policy test.

Month of pulse or step = 10000
~ Months
~ Used in the lobin test formulation.

Lob in design changes after B1 = 0
~ Designs/Month
~ Default value set to zero. Suggested test value is 200.

Defects before B2 fix rate = Open concern fix before B2 rate
~ Defects/Month
~ Fixing an open concern fixes a defect.

Defects completing Build 2 rate = Frac designs in Build 2 with defects*
Designs completing Build 2 rate
~ Defects/Month
~ Part of a coflow.

Frac designs in Build 2 with defects =zidz(Defects awaiting Build 2,
Designs awaiting Build 2)
~ Defects/Design
~ Identical structure to the other builds. Zidz means zero if division by zero, and is necessary because designs can reach zero.

The following structure has no diagram, because it is mostly analogous to the structure shown in Figure 55.

Defects awaiting Build 3 = INTEG(+Defects completing Build 2 rate
-Defects before B3 fix rate-Defects completing Build 3 rate+
Defects intro before B3 rate,0)
~ Defects
~ Awaiting the pre-production build.

Defects intro before B3 rate = Open concern fix before B3 rate*
Defects introduced per open issue fixed
~ Defects/Month
~ Identical structure to the other builds.

Defects before B3 fix rate =Open concern fix before B3 rate
~ Defects/Month
~ Fixing an open concern fixes a defect.

Defects completing Build 3 rate = Designs completing Build 3 rate*
Frac designs in Build 3 with defects
~ Defects/Month
~ Part of a coflow, driven by the movement of the designs.

Frac designs in Build 3 with defects = zidz(Defects awaiting Build 3,
Designs awaiting Build 3)
~ Defects/Design
~ Identical structure to the other builds. Zidz means zero if division by zero, and is necessary because designs can reach zero.

The following structure has no diagram, because it is analogous to the structure shown in Figure 55. For explanation and a diagram explaining the additional outflow from Defects in designs completed, see Figure 57.5 and the accompanying text.

Defects in designs completed = INTEG(Defects completing Build 3 rate-
Defects completed launch rate-Defects compl fix rate+
Defects intro before launch rate-Defect containing before launch rate,0)
~ Defects
~ Defects in designs awaiting launch into production.

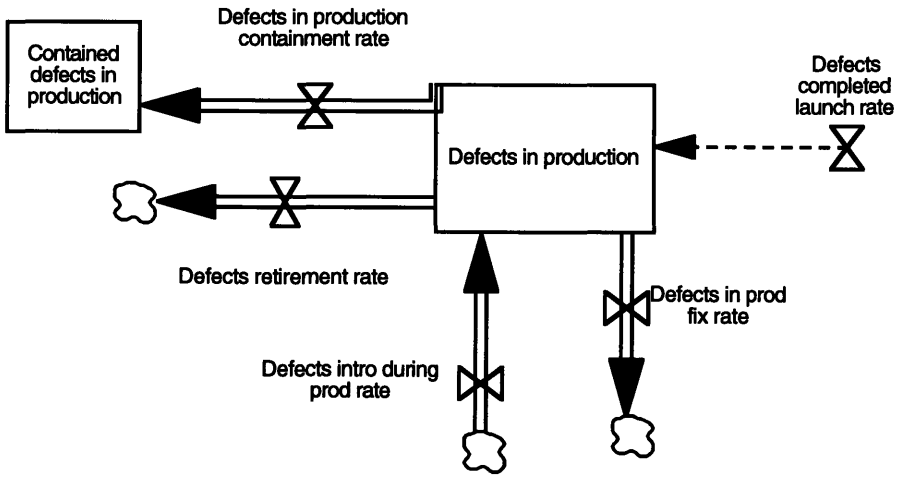
Defects intro before launch rate = Open concern fix before launch rate*
Defects introduced per open issue fixed
~ Defects/Month
~ Defects created while designs are awaiting launch.

Defects compl fix rate = if then else(Months to launch=
Months before launch that model year changes,0,
Open concern fix before launch rate)
~ Defects/Month
~ The rate is set to zero when the discrete "flush" of designs from the current year to the production year happens, so the stock does not go negative.

Defects completed launch rate =Frac designs completed with defects*
Designs completed launch rate
~ Defects/Month
~ Part of a coflow.

Frac designs completed with defects = zidz(Defects in designs completed,
Designs Completed)
~ Defects/Design
~ Identical structure to the other builds. Zidz means zero if division by zero, and is necessary because designs can reach zero.

Figure 56.5




Defects in production increase in two ways. Defects enter production, and rework changes introduce new defects. They decrease in three ways. Engineers fix them, engineers contain them, and they retire out of the system. The initial value corresponds to the equilibrium condition when 25% of parts and prints are mismatched.

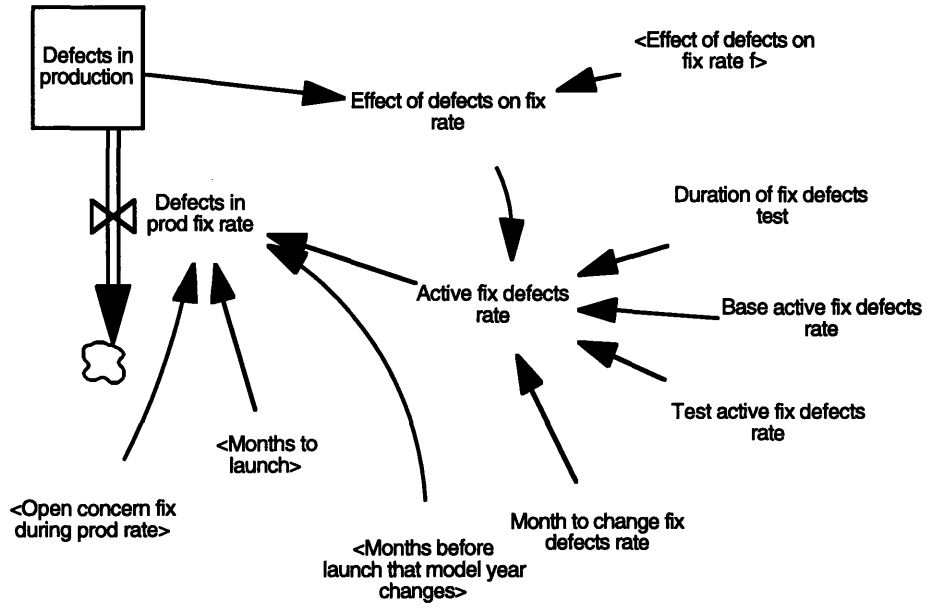
Defects in production = INTEG(Defects completed launch rate-
Defects in prod fix rate-Defects retirement rate-Defects in production containment rate+

Defects intro during prod rate,Initial defects in production)
~ Defects
~ This stock does not include contained defects. Includes defects built up over years of operation.

Initial defects in production = 260
~ Defects
~ Equilibrium condition based on 25% of designs mismatched.

Defects retirement rate =Frac designs in prod with defects*
Designs retirement rate
~ Defects/Month
~ Part of a coflow. Defects leaving the system as designs leave.

Figure 57




The rate of fixing defects in production includes the structure shown in previous, analogous, equations -- fixing open concern fixes defects. The rate additionally includes active fixing of defects, as part of a policy test. For the test, the fix rate is changed from the base fix rate to a test rate at a certain month. The test lasts the number of months indicated by the duration of the test. The rate of fixing is limited by the number of defects in the stock, by way of the *Effect of defects on fix rate*. Such an effect is necessary for two reasons. First, fixing gets more difficult the fewer defects there are. Second, if there are no defects, fixing is impossible.

The model assumes no constraint on resources, so time spent by engineers running builds, discovering defects, or fixing mismatches will not effect time spent fixing defects.

Defects in prod fix rate = if then else(Months to launch=

Months before launch that model year changes,0,
Open concern fix during prod rate)+Active fix defects rate
~ Defects/Month
~ Fixing an open concern fixes a defect. The rate is set to zero when the discrete "flush" of designs from the production year to be retired happens, so the stock does not go negative.

Active fix defects rate = Base active fix defects rate +
(pulse(Month to change fix defects rate, Duration of fix defects test)*
Test active fix defects rate*
Effect of defects on fix rate)
~ Defects/Month
~ Active fix rate is for testing.

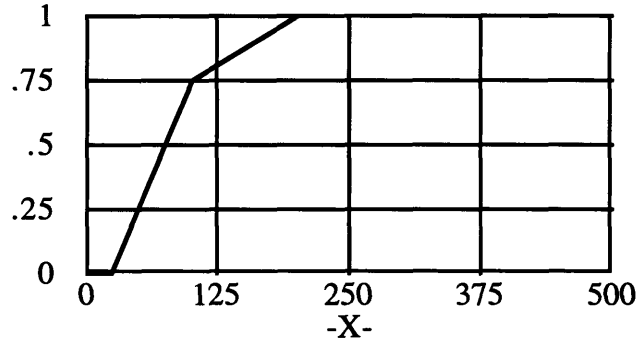
Base active fix defects rate = 0
~ Defects/Month
~ No active effort to fix defects.

Test active fix defects rate = 0
~ Defects/Month
~ Not changed for model runs in the paper.

Effect of defects on fix rate = Effect of defects on fix rate f(Defects in production)
~ dimensionless
~ First order control.

BASE

Effect of defects on fix rate f




The effect of defects on the rate of fixing defects is operationalized as the function shown above. The domain of the function is the interval between 0 and 500, and represents the number of mismatches. As the number of defects falls below 250 (the initial value is 260), fixing defects has less and less effectiveness. When there are only 25 defects left, they are so difficult to find and fix that the effect is 0.

Effect of defects on fix rate f ([(0,0)-(500,1)],(0,0),(25,0)
,(100,0.75),(200,1),(500,1) )
~ dimensionless
~ Fixing gets more difficult the fewer defects there are. Eventually, if there aren't defects, you can't fix them.

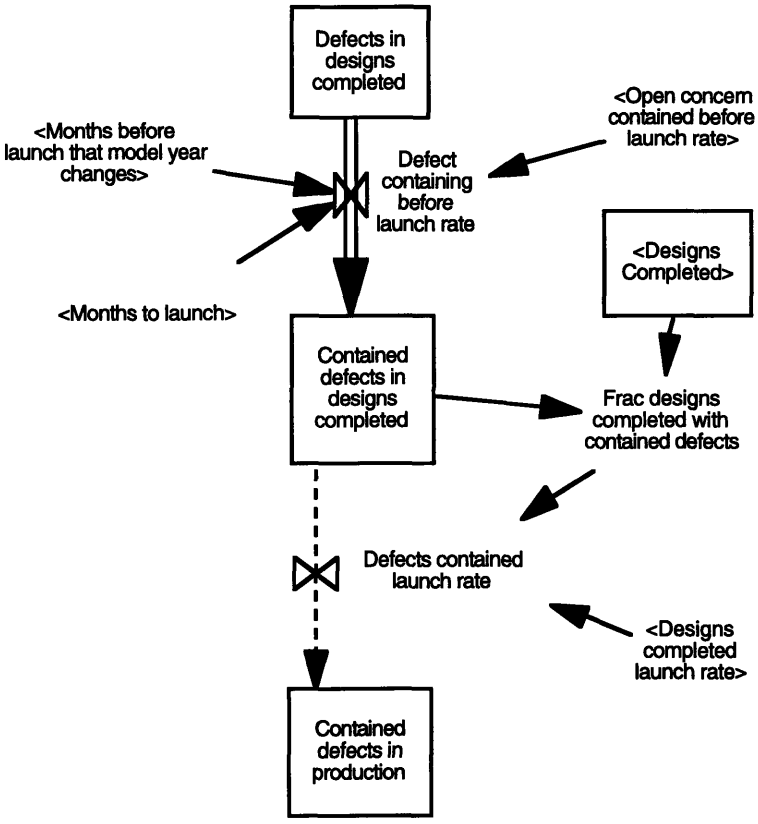
Duration of fix defects test = 0
~ Months
~ Value to be changed with testing.

Month to change fix defects rate = 10000
~ Months
~ Set so high so that the test doesn't happen in base model runs.

Defects intro during prod rate = Defects introduced per open issue fixed*
Open concern fix during prod rate
~ Defects/Month
~ Defects introduced because of rework design changes.

Frac designs in prod with defects = zidz(Defects in production,
Designs in production)
~ Defects/Design
~ Part of a coflow. Zidz means zero if division by zero, and is necessary because designs can reach zero.

Figure 57.5




Containing defects before launch increases the number of contained defects in designs completed. Containing defects is driven by the rate of containing open concerns. The rate is set to zero when the defects flush into production, so the stock does not go negative. The launch rate of contained defects is determined by the rate of launching designs, in a co-flow structure explained previously.

Contained defects in designs completed = INTEG(Defect containing before launch rate
-Defects contained launch rate,0)

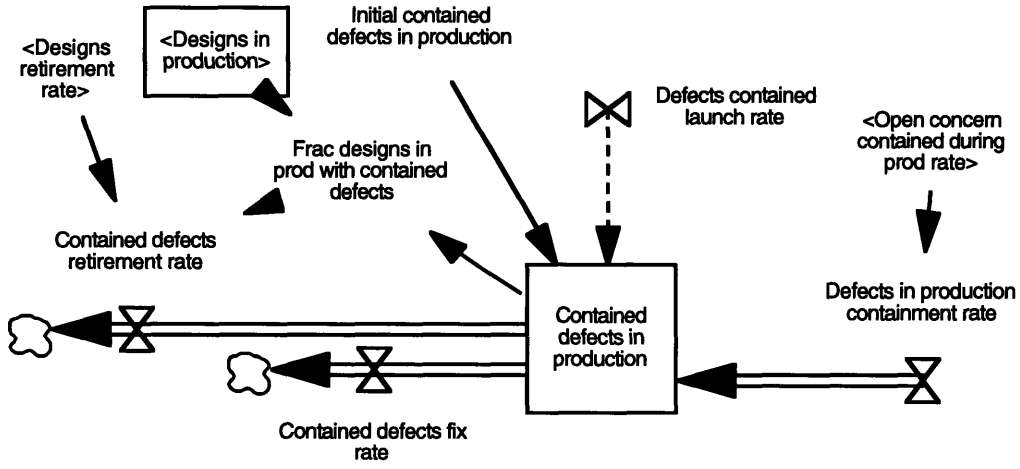
~ Defects
~ Contained defects awaiting launch

Defect containing before launch rate = if then else(Months to launch=
Months before launch that model year changes,0,
Open concern contained before launch rate)
~ Defects/Month

Defects contained launch rate = Designs completed launch rate*
Frac designs completed with contained defects
~ Defects/Month
~ Part of a coflow.

Frac designs completed with contained defects = zidz(Contained defects in designs completed,
Designs Completed)
~ Defects/Design
~ Part of a coflow. Zidz means zero if division by zero, and is necessary because designs can reach zero.

Figure 57.7




Contained defects in production increase in two ways. Defects that were already contained enter production (driven by the containing of open concerns), or defects that had already entered production get contained. The stock decreases in two ways. People fix the contained defects (as described more extensively in the text accompanying Figure 58), or they retire. The retirement rate is determined by the retirement of designs, in a co-flow structure explained previously. The initial number of contained defects is based on the equilibrium conditions corresponding to 25% mismatches.

Contained defects in production = INTEG(Defects contained launch rate
+Defects in production containment rate-Contained defects retirement rate-
Contained defects fix rate,Initial contained defects in production)
~ Defects
~ Contained defects built up over years of production.

Initial contained defects in production = 2235

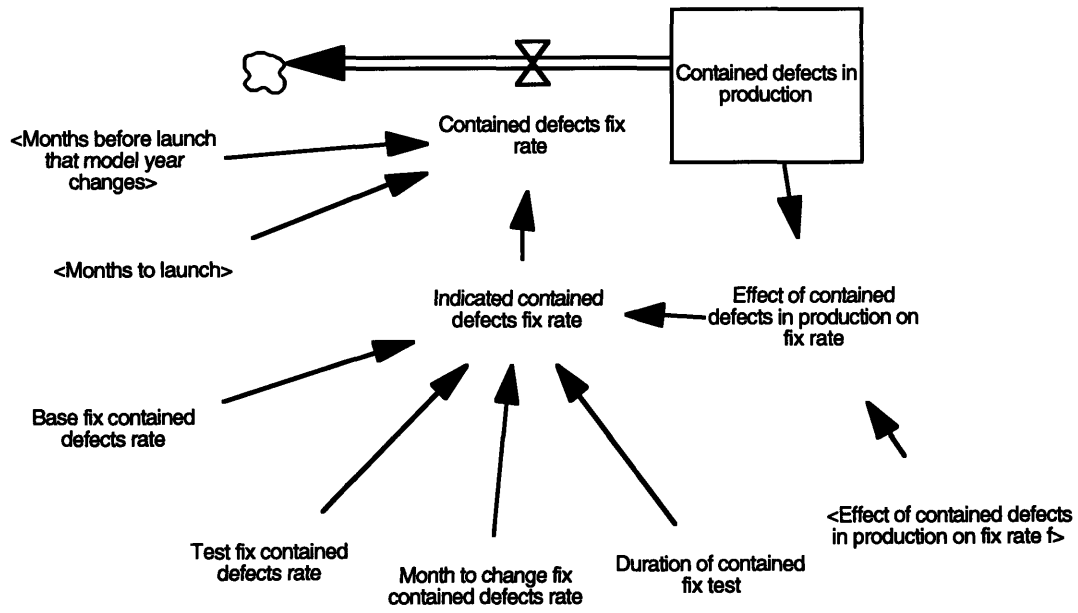
~ Defects
~ Based on the equilibrium conditions corresponding to 25% mismatches.

Defects in production containment rate = if then else(Months to launch=
Months before launch that model year changes,0,
Open concern contained during prod rate)
~ Defects/Month
~ The rate is set to zero when the discrete "flush" of designs from the production year to be retired happens, so the stock does not go negative.

Contained defects retirement rate = Designs retirement rate*
Frac designs in prod with contained defects
~ Defects/Month
~ As designs are retired, contained defects are retired too, at a rate proportional to their fraction within the designs.

Frac designs in prod with contained defects = zidz(Contained defects in production,
Designs in production)
~ Defects/Design
~ Part of a coflow. Zidz means zero if division by zero. The structure is necessary because designs can reach zero.

Figure 58



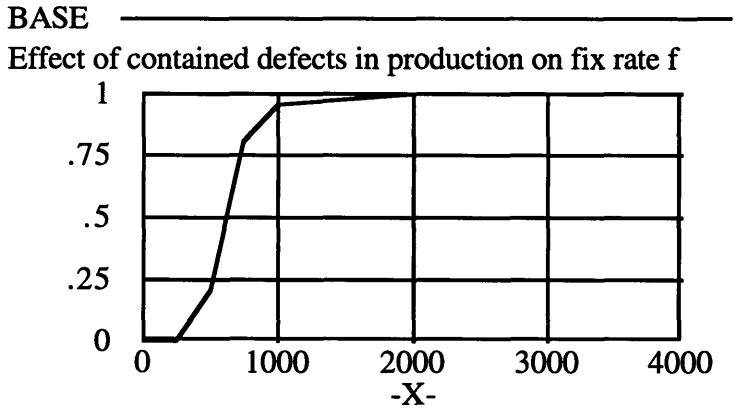

The outflow rate of contained defects includes active fixing of defects, as part of a policy test. For the test, the fix rate is changed from the base fix rate to a test rate at a certain month. The test lasts the number of months indicated by the duration of the test. The rate of fixing is limited by the number of defects in the stock, by way of the *Effect of contained defects in production on fix rate*. Such an effect is necessary for two reasons. First, fixing gets more difficult the fewer defects there are. Second, if there are no defects, fixing is impossible.

The model assumes no constraint on resources, so time spent by engineers running builds, discovering defects, or fixing mismatches will not effect time spent fixing contained defects.

Contained defects fix rate = if then else(Months to launch=
Months before launch that model year changes,0,
Indicated contained defects fix rate)
~ Defects/Month
~ The rate is set to zero when the discrete "flush" of designs from the production year to be retired happens, so the stock does not go negative.

Indicated contained defects fix rate = Base fix contained defects rate +
(pulse(Month to change fix contained defects rate, Duration of contained fix test)*
Test fix contained defects rate*Effect of contained defects in production on fix rate)
~ Defects/Month
~ Pulse structure allows for the Fix Contained Defects policy test.

Effect of contained defects in production on fix rate =
Effect of contained defects in production on fix rate f(Contained defects in production)
~ dimensionless
~ This structure ensures that the stock does not go negative. That is, there will be no fixing if there are no contained defects.




The effect of contained defects on the rate of fixing contained defects is operationalized as the function shown above. The domain of the function is the interval between 0 and 4000, and represents the number of contained defects. As the number of contained defects falls below 2000 (the initial value is 2235), fixing defects has less and less effectiveness. When there are only 250 contained defects left, they are so difficult to find and fix that the effect is 0.

Effect of contained defects in production on fix rate f ([(0,0)-
(4000,1)],(0,0),(250,0),(500,0.2),(750,0.8),(1000,0.95)
,(2000,1),(4000,1) )
~ dimensionless
~ Fixing gets more difficult as there are fewer defects to fix, because they will be more diffuse.

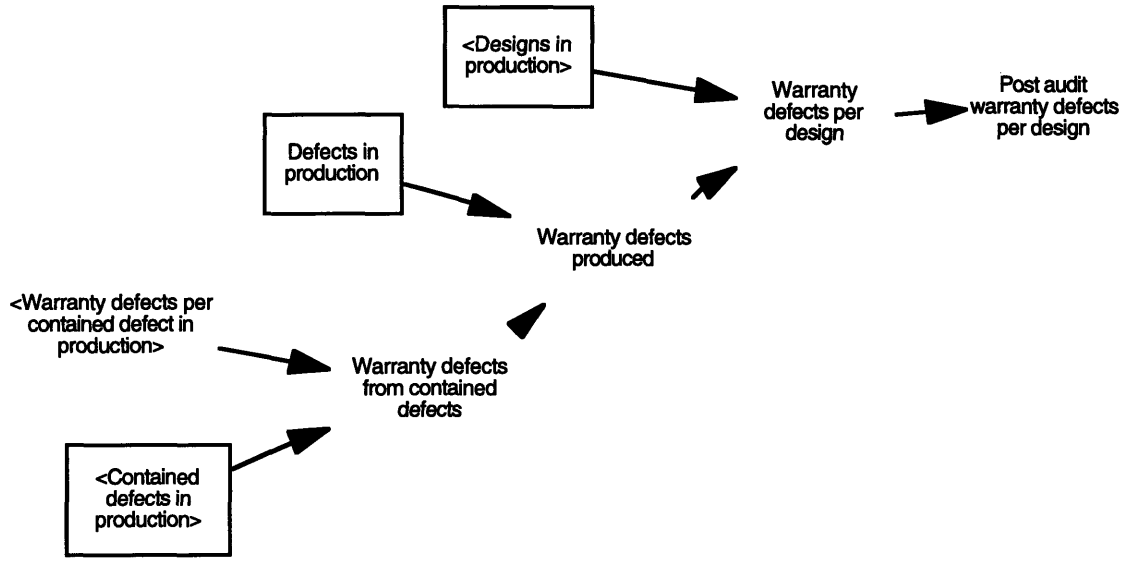
Duration of contained fix test = 36
~ Months
~ Value to be changed in the test runs.

Base fix contained defects rate = 0
~ Defects/Month
~ No active fixing in the base run.

Test fix contained defects rate = 20
~ Defects/Month
~ Rough estimate. Value to be changed in the test runs.

Month to change fix contained defects rate = 10000
~ Months
~ Value to be changed in the test runs.

Figure 59




The purpose of this structure is to determine the quality metric, *Post audit warranty defects per design.* The number of warranty defects produced is the sum of defects from contained defects and defects from normal defects. The first type of defects is determined by the number of contained defects in production and the number of warranty defects introduced per contained defect. The second type is more straightforward -- there is implicitly one warranty defect for every defect in production. Dividing the defects by the number of designs gives a fraction of warranty defects per design. The model assumes that some of the defects (particularly those around launch) will be discovered and repaired before shipment, so the number of defects are "smoothed," removing the initial spike.

Post audit warranty defects per design = SMOOTH(Warranty defects per design,4)
~ Defects/Design

~ The quality metric. The smooth eliminates the extreme spikiness of quality around launch and represents the fixing of many defects in the first weeks of the production year.

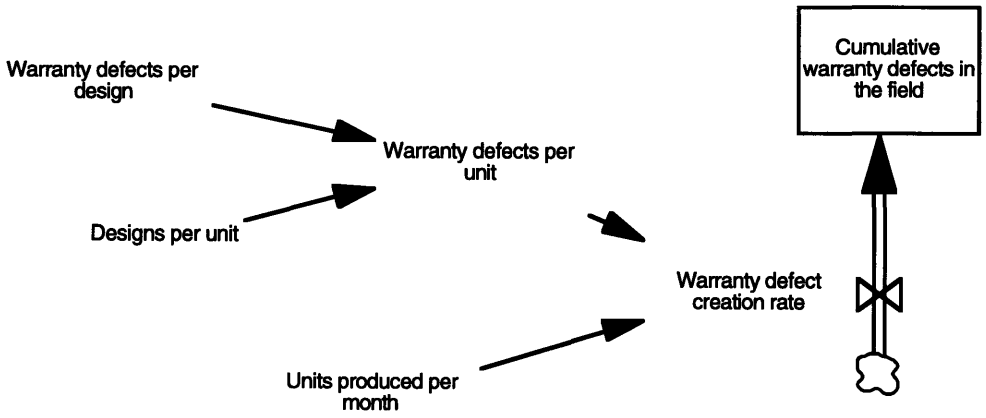
Warranty defects produced = Warranty defects from contained defects+
Defects in production
~ Defects
~ Two sources of warranty defects

Warranty defects per design = zidz(Warranty defects produced,
Designs in production)
~ Defects/Design
~ Zidz means zero if division by zero. The structure is necessary because designs can reach zero.

Warranty defects from contained defects =Contained defects in production*
Warranty defects per contained defect in production
~ Defects

Warranty defects per contained defect in production = 0.05
~ dimensionless
~ Twenty contained defects have the same quality effect at 1 defect. This parameter is varied in sensitivity tests.

Figure 60




The purpose of this structure is determine the cumulative warranty defects in the field. The creation rate is determined by the number of units produced per month times warranty defects per unit. Warranty defects per unit is determined by the number of warranty defects per design times the number of designs per unit. The model assumes no retirement of products in the field.

Cumulative warranty defects in the field = INTEG(Warranty defect creation rate,0)
~ Defects
~ Measurement of total defects in shipped products.

Warranty defect creation rate = Warranty defects per unit*Units produced per month
~ Defects/Month

Designs per unit = 100
~ Designs/Unit
~ Rough estimate.

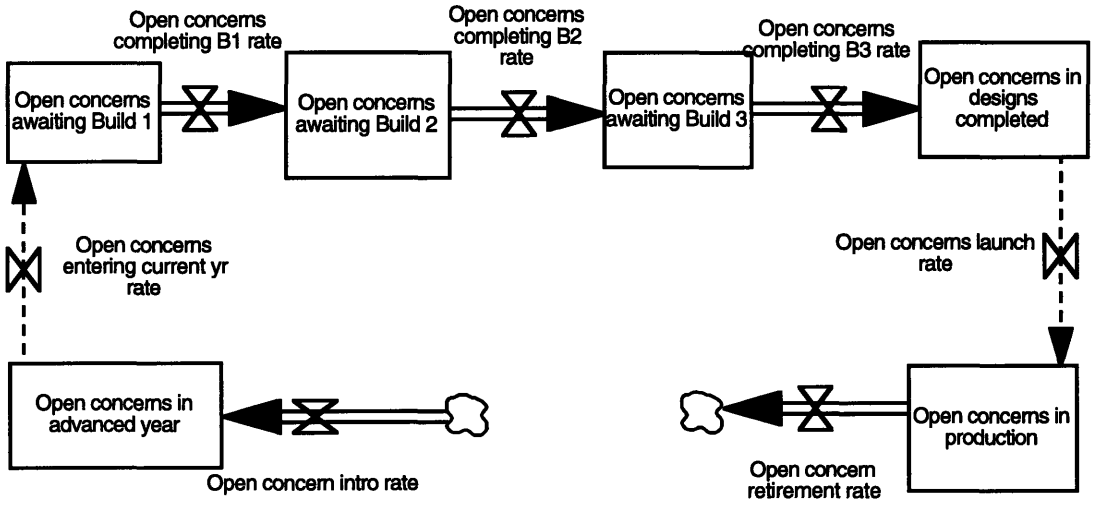
Units produced per month = 10000
~ Designs/Unit
~ Rough estimate.

Warranty defects per unit = Warranty defects per design*Designs per unit
~ Defects/Unit

## Open Concerns

The purpose of this sector is to track open concerns (or "discovered defects") through the various stages in the three year cycle. These open concerns are specific to ones that could cause warranty problems or customer satisfaction problems down the line.

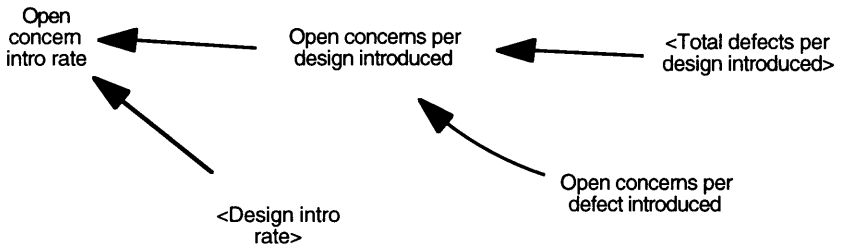
Figure 61




The purpose of Figure 61 shown above is to provide a broad overview of the sector. Similar to designs and defects, open concerns are introduced, move from the advanced year to the current year, pass through each of the builds, enter production, and retire away. Multiple other inflows and outflows are not shown in this diagram.

Like defects, this chain of stocks is driven by the flow of designs, using a coflow structure. For an example, see Figure 51.5.

Figure 62



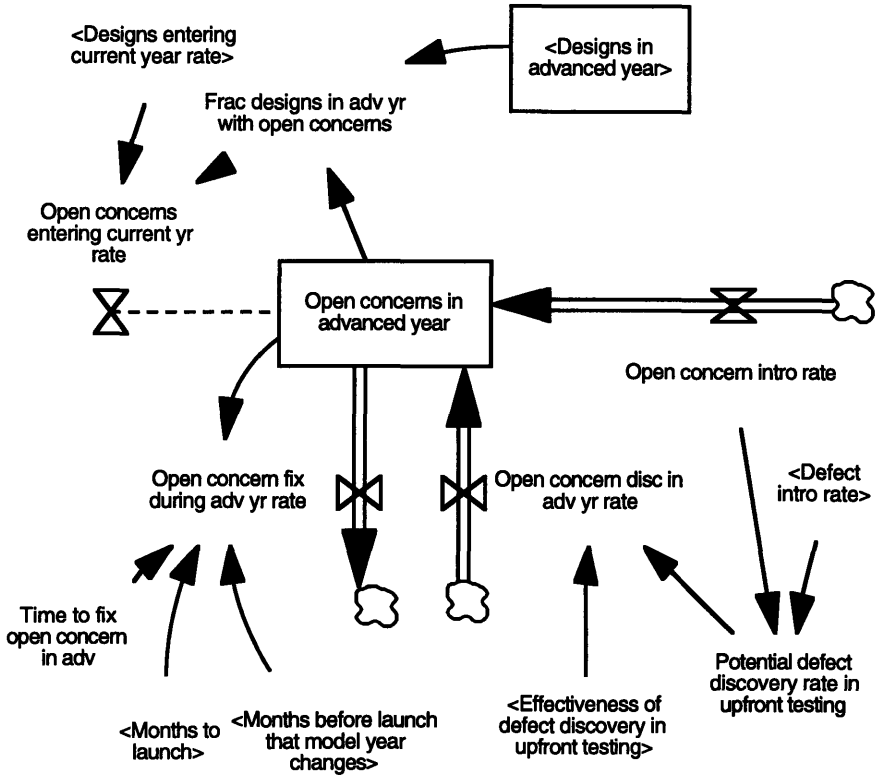

The open concern intro rate is determined by the design introduction rate and the number of open concerns for every design introduced. The number of open concerns per design is determined by the defects per design introduced and the open concerns per defect introduced. The latter variable is greater than zero only if design engineering knows of some defects before any testing occurs.

Open concern intro rate = Design intro rate*Open concerns per design introduced
- ~ Defects/Month
- ~ Introduction is driven by introduction of designs.

Open concerns per design introduced = Total defects per design introduced*
Open concerns per defect introduced
- ~ Defects/Design

Open concerns per defect introduced = 0
- ~ dimensionless
- ~ Open concerns are not created until someone discovers a defect.

Figure 63




The structure shown in Figure 63 regulates the number of open concerns in the advanced year. As shown in earlier formulations, the horizontal introduction inflow and the outflow to the current year are both driven with a coflow structure by the flow of designs through the various stages. The structure in the top left of Figure 63 governing the outflow, *Open concerns entering current year rate,* is similar in structure to the formulations in Figure 51.5. The outflow rate of open concerns is equal to the outflow rate of designs times the fraction of open concerns per design.

The vertical outflow rate is driven by the rate of fixing open concerns. Fixing open concerns is formulated as a simple first order drain. The rate is the number of open concerns in the stock divided by the average time to fix an open concern. The model makes a strong assumption about resources that impacts this formulation. The model does not account for resources, so other engineering activities such as running builds and discovering defects do not impact the rate of fixing open concerns. Nor does time fixing open concerns impact other activities.

During the model year change, the fix rate is set to zero because the discrete flush of defects out of the stock will empty the entire stock in one time step. Any additional outflow such as through fixing would send the stock negative.

The vertical introduction rate is driven by discovery of defects in the builds. The defects that could be potentially discovered in any build is the number that are present during a build minus the number that engineers already know about. The number actually discovered is determined by the potential discovery rate times how effective the organization is at discovering defects in the build.

The structure explained here is analogous for the subsequent stocks. Thus, diagrams will not be shown. Because defect discovery in the advanced year is formulated similar to the other builds, discovery happens in the short period of time when defects are entering the advanced year. Discovery does not happen gradually over the first few months, as one might see in the real world.

Open concerns in advanced year = INTEG(Open concern intro rate
-Open concerns entering current yr rate+Open concern disc in adv yr rate-
Open concern fix during adv yr rate,0)
~ Defects
~ This stock represents all the defects that are known during the time between
24 months before launch and 12 months before launch.

Open concern disc in adv yr rate =Potential defect discovery rate in upfront testing*
Effectiveness of defect discovery in upfront testing
~ Defects/Month
~ The discovery rate is a function of how many defects could be discovered and
the effectiveness of discovery.

Potential defect discovery rate in upfront testing = Defect intro rate-Open concern intro rate
~ Defects/Month
~ One cannot discover defects that have already been discovered.

Open concern fix during adv yr rate =if then else (Months to launch=
Months before launch that model year changes,0,Open concerns in advanced year/
Time to fix open concern in adv)
~ Defects/Month
~ Similar to other structures, the rate is set to zero during the "flush."

Time to fix open concern in adv = 1
~ Months
~ Time to fix is shorter in the advanced year because most of the discovery happens in
design engineering and does not require communication between areas like the fixing that
happens later in the year.

Open concerns entering current yr rate = Designs entering current year rate*
Frac designs in adv yr with open concerns
~ Defects/Month

~ Part of a coflow.

Frac designs in adv yr with open concerns = zidz(Open concerns in advanced year,
Designs in advanced year)
~ Defects/Design
~ Part of a coflow. Zidz means zero if division by zero. The structure is necessary because designs can reach zero.

The following structure has a set of equations that are analogous to the ones depicted in the Figure 63. Thus, the diagram has not been replicated here.

Open concerns awaiting Build 1 = INTEG(Open concern disc before B1 rate+
Open concerns entering current yr rate-Open concern fix before B1 rate-
Open concerns completing B1 rate,0)
~ Defects
~ Open concerns that have not yet been fixed, that are awaiting the first build.

Open concern disc before B1 rate = Potential defect discovery rate in current year arrival*
Effectiveness of defect discovery in current year arrival
~ Defects/Month
~ Rate of discovery.

Effectiveness of defect discovery in current year arrival = 0
~ dimensionless
~ Some defects could be discovered without testing, when the designs arrive in manufacturing. Set to zero in the base run.

Potential defect discovery rate in current year arrival = Defects entering current yr rate-
Open concerns entering current yr rate
~ Defects/Month
~ One cannot discover defects that have already been discovered.

Open concern fix before B1 rate = Open concerns awaiting Build 1/
Time to fix open concern aw B1
~ Defects/Month
~ Formulated as a draining function.

Time to fix open concern aw B1 = 3
~ Months
~ Time includes documentation of the change, delivering the change to design engineering, assigning someone to complete the change, completing the change, upfront mockup testing of the change, documenting the change and delivery of the results back to manufacturing.

Open concerns completing B1 rate = Frac designs aw B1 with open concerns*
Designs completing Build 1 rate
~ Defects/Month
~ Part of a coflow.

Frac designs aw B1 with open concerns =zidz(Open concerns awaiting Build 1,
Designs awaiting Build 1)
~ Defects/Design
~ Part of the coflow structure. Zidz means zero if division by zero. The structure is necessary because designs can reach zero.

The following structure has a set of equations that are analogous to the ones depicted in the Figure 63. Thus, the diagram has not been replicated here.

Open concerns awaiting Build 2 = INTEG(Open concern disc in B1 rate
+Open concerns completing B1 rate-Open concern fix before B2 rate
-Open concerns completing B2 rate,0)
~ Defects
~ Open concerns that have not yet been fixed, that are awaiting the second build.

Open concern disc in B1 rate = Effectiveness of defect discovery in Build 1*
Potential defect discovery rate in Build 1
~ Defects/Month
~ They discover the number of undiscovered defects times the fraction they discover in the build.

Potential defect discovery rate in Build 1 =
Defects completing Build 1 rate-
Open concerns completing B1 rate
~ Defects/Month
~ One cannot discover defects that have already been discovered.

Effectiveness of defect discovery in Build 1 = 0.6
~ dimensionless
~ Rough estimate.

Open concern fix before B2 rate = Open concerns awaiting Build 2/
Time to fix open concern aw B2
~ Defects/Month
~ Formulated as a draining function.

Time to fix open concern aw B2 =3
~ Months
~ Time includes documentation of the change, delivering the change to design engineering, assigning someone to complete the change, completing the change, upfront mockup testing of the change, documenting the change and delivery of the results back to manufacturing.

Frac designs aw B2 with open concerns = zidz(Open concerns awaiting Build 2,
Designs awaiting Build 2)
~ Defects/Design
~ Zidz means zero if division by zero. The structure is necessary because designs can reach zero.

Potential defect discovery rate in B2 = Defects completing Build 2 rate-
Open concerns completing B2 rate
~ Defects/Month
~ One cannot discover defects that have already been discovered.

Effectiveness of defect discovery in Build 2 =
Default effectiveness of defect discovery in Build 2
~ dimensionless
~ Intermediary variable allows for the changing of the effectiveness.

Default effectiveness of defect discovery in Build 2 = 0.6
~ dimensionless
~ Rough estimate. Same as other builds.

Open concerns completing B2 rate = Designs completing Build 2 rate*
Frac designs aw B2 with open concerns
~ Defects/Month

~ Part of a coflow

The following structure has a set of equations that are analogous to the ones depicted in the Figure 63. Thus, the diagram has not been replicated here.

Open concerns awaiting Build 3 = INTEG(Open concern disc in B2 rate
+Open concerns completing B2 rate-Open concern fix before B3 rate
-Open concerns completing B3 rate,0)
~ Defects
~ Open concerns that have not yet been fixed, that are awaiting the third build.

Open concern disc in B2 rate =Potential defect discovery rate in B2*
Effectiveness of defect discovery in Build 2
~ Defects/Month
~ They discover the number of undiscovered defects times the fraction they discover in the build.

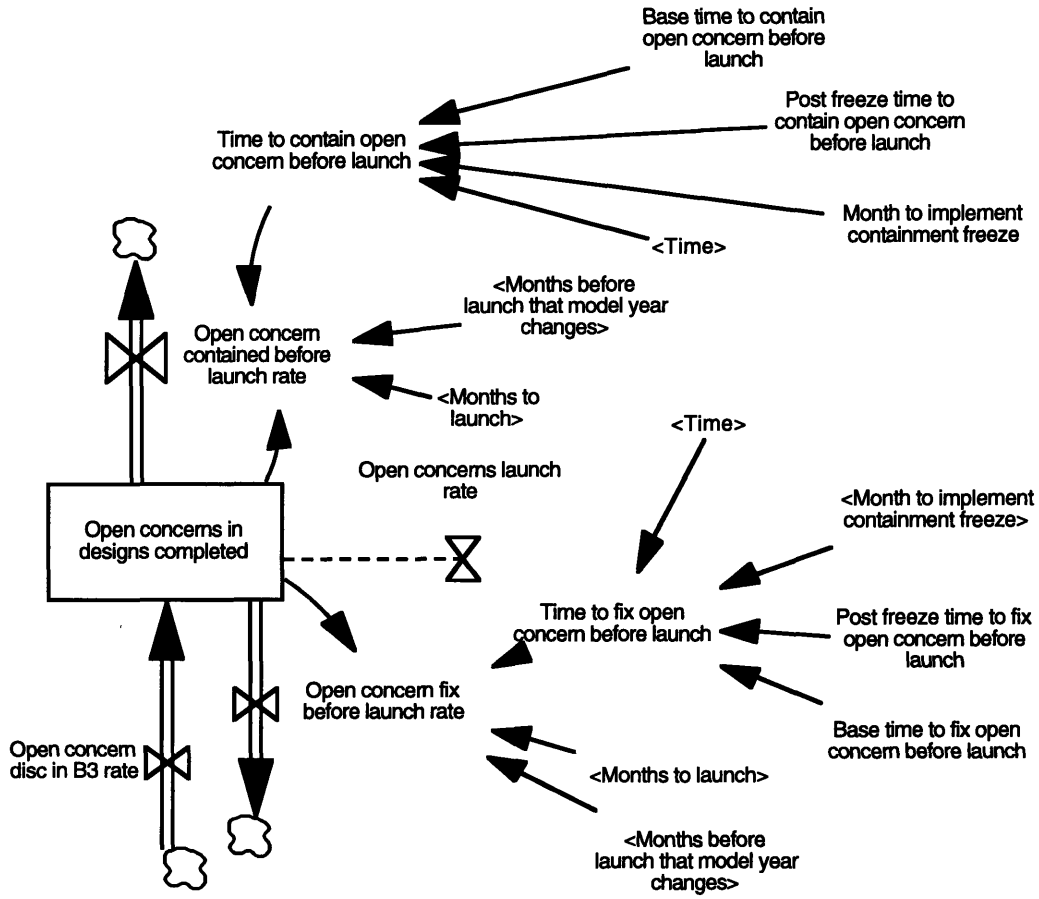
Open concern fix before B3 rate = Open concerns awaiting Build 3/
Time to fix open concern aw B3
~ Defects/Month
~ Formulated as a draining function.

Time to fix open concern aw B3 = 3
~ Months
~ Time includes documentation of the change, delivering the change to design engineering, assigning someone to complete the change, completing the change, upfront mockup testing of the change, documenting the change and delivery of the results back to manufacturing.

Open concerns completing B3 rate = Designs completing Build 3 rate*
Frac designs aw B3 with open concerns
~ Defects/Month
~ Part of a coflow.

Frac designs aw B3 with open concerns = zidz(Open concerns awaiting Build 3,
Designs awaiting Build 3)
~ Defects/Design
~ Zidz means zero if division by zero. The structure is necessary because designs can reach zero.

Figure 64




Discovering open concerns in build 3 increases the number of open concerns in designs completed. Discovery is formulated similarly to the discovery rate depicted in Figure 63. Three rates decrease the stock. The first is the horizontal outflow, Open concerns launch rate, and is formulated as part of a coflow structure as described earlier.

The top vertical outflow represents containing open concerns, and is a first order drain. The rate is the number of open concerns divided by the time to contain a concern. During the model year change, the rate is set to zero because the discrete flush of defects out of the stock will empty the entire stock in one time step. Any additional outflow such as through fixing would send the stock negative. The time to contain an open concern can be one of two values, for testing. Usually set to the base time, during a test the value will change to the post-freeze time at the specified implementation month.

The bottom vertical outflow represents fixing open concerns, and is analogous in structure to containing open concerns, including the policy test structure.

The times to fix and contain defects determine the emphasis placed on fixing vs. containing. For example, in the base conditions, time to fix is longer than time to contain, so more open concerns will be contained than fixed.

Open concerns in designs completed = INTEG(Open concern disc in B3 rate
+Open concerns completing B3 rate-Open concern fix before launch rate-
Open concern contained before launch rate
-Open concerns launch rate,0)
~ Defects
~ Open concerns awaiting launch.

Open concern contained before launch rate =if then else(Months to launch=
Months before launch that model year changes,0, Open concerns in designs completed/
Time to contain open concern before launch)
~ Defects/Month
~ The rate is set to zero when the discrete "flush" of designs from the
current year to the production year happens, so the stock does not go negative.

Time to contain open concern before launch = if then else (Time>
Month to implement containment freeze,Post freeze time to contain open concern before launch,
Base time to contain open concern before launch)
~ Months
~ Structure allows for the Less Containing policy test.

Base time to contain open concern before launch = 1
~ Months
~ Containing is faster than fixing, so this time is shorter than the fix time.

Post freeze time to contain open concern before launch = 1e+009
~ Months
~ This value will be changed in the policy tests.

Open concern fix before launch rate = if then else(Months to launch=
Months before launch that model year changes,0,
Open concerns in designs completed/
Time to fix open concern before launch)
~ Defects/Month
~ The rate is set to zero when the discrete "flush" of designs from the
current year to the production year happens, so the stock does not go negative.

Time to fix open concern before launch =if then else (Time>
Month to implement containment freeze,Post freeze time to fix open concern before launch,
Base time to fix open concern before launch)
~ Months
~ Structure allows for the Less Containing policy test.

Base time to fix open concern before launch = 10
~ Months
~ Between build 3 and launch, more emphasis is placed on containing defects than on fixing,
so the time delay is long.

Post freeze time to fix open concern before launch = 1e+009
~ Months
~ Set to a high number so the structure is inactive unless used for a test.

Open concern disc in B3 rate = Effectiveness of defect discovery in Build 3*
Potential defect discovery rate in B3
~ Defects/Month
~ They discover the number of undiscovered defects times the fraction they discover in the build.

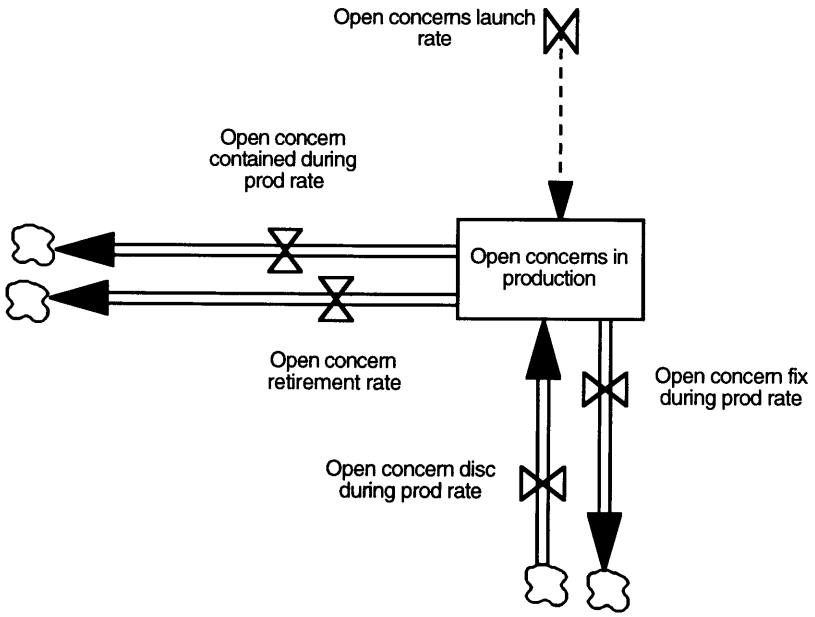
Potential defect discovery rate in B3 = Defects completing Build 3 rate-
Open concerns completing B3 rate
~ Defects/Month
~ One cannot discover defects that have already been discovered.

Effectiveness of defect discovery in Build 3 = 0.6
~ dimensionless
~ Default setting is same as the other builds.

Open concerns launch rate = Frac designs completed with open concerns*
Designs completed launch rate
~ Defects/Month
~ Part of a coflow.

Frac designs completed with open concerns = zidz(Open concerns in designs completed, Designs Completed)
~ Defects/Design
~ Part of a coflow. Zidz means zero if division by zero. The structure is necessary because designs can reach zero.

Figure 65




Open concerns in production increases as open concerns enter production. The launch rate is determined by a co-flow structure described earlier. Discovering open concerns also increases the stock. The formulation for discovery is described and shown in Figure 63

and its supporting text. Three rates decrease the stock. Open concerns retire, again determined by a co-flow structure. Fixing open concerns and containing open concerns also decrease the stock. These two rates are first order drains with different draining times. The structure is analogous to the structure depicted in Figure 64.

Open concerns in production = INTEG(Open concern disc during prod rate
+Open concerns launch rate-Open concern fix during prod rate-
Open concern retirement rate-Open concern contained during prod rate
,0)
~ Defects
~ Discovered defects in production.

Open concern disc during prod rate =Potential defects to discover in production*
Effectiveness of defect discovery in production
~ Defects/Month
~ They discover the number of undiscovered defects times the fraction they discover in the build.

Potential defects to discover in production = Defects completed launch rate-
Open concerns launch rate
~ Defects
~ They cannot discover defects that have already been discovered.

Open concern fix during prod rate = if then else(Months to launch=
Months before launch that model year changes,0,Open concerns in production/
Time to fix open concern during prod)
~ Defects/Month
~ The rate is set to zero when the discrete "flush" of designs from the current year to the production year happens, so the stock does not go negative.

Time to fix open concern during prod = if then else (Time>
Month to implement containment freeze,Post freeze time to fix open concern during prod,
Base time to fix open concern during prod)
~ Months
~ Structure allows for the Less Containing policy test.

Base time to fix open concern during prod = 10
~ Months
~ There is less fixing and more containing during production.

Post freeze time to fix open concern during prod = 1e+009
~ Months
~ Part of the Less Containing policy test.

Time to discover defect in production = 6
~ Months
~ Formulated differently than the other defect discovery structures. Once in production, defects will be discovered slowly.

Effectiveness of defect discovery in production = 0.7
~ dimensionless
~ A greater percentage (but not all) of defects will be discovered in production than in the builds.

Open concern contained during prod rate = if then else(Months to launch=
Months before launch that model year changes,0,

Open concerns in production/Time to contain open concern during prod)
~ Defects/Month
~ The rate is set to zero when the discrete "flush" of designs from the current year to the production year happens, so the stock does not go negative.

Time to contain open concern during prod = if then else (Time> Month to implement containment freeze,Post freeze time to contain open concern during prod, Base time to contain open concern during prod)
~ Month
~ Structure allows for the Less Containing policy test.

Base time to contain open concern during prod = 1
~ Months
~ Discovered defects in production are quickly contained.

Post freeze time to contain open concern during prod = 1e+009
~ Months
~ A high number so the flow is inactive

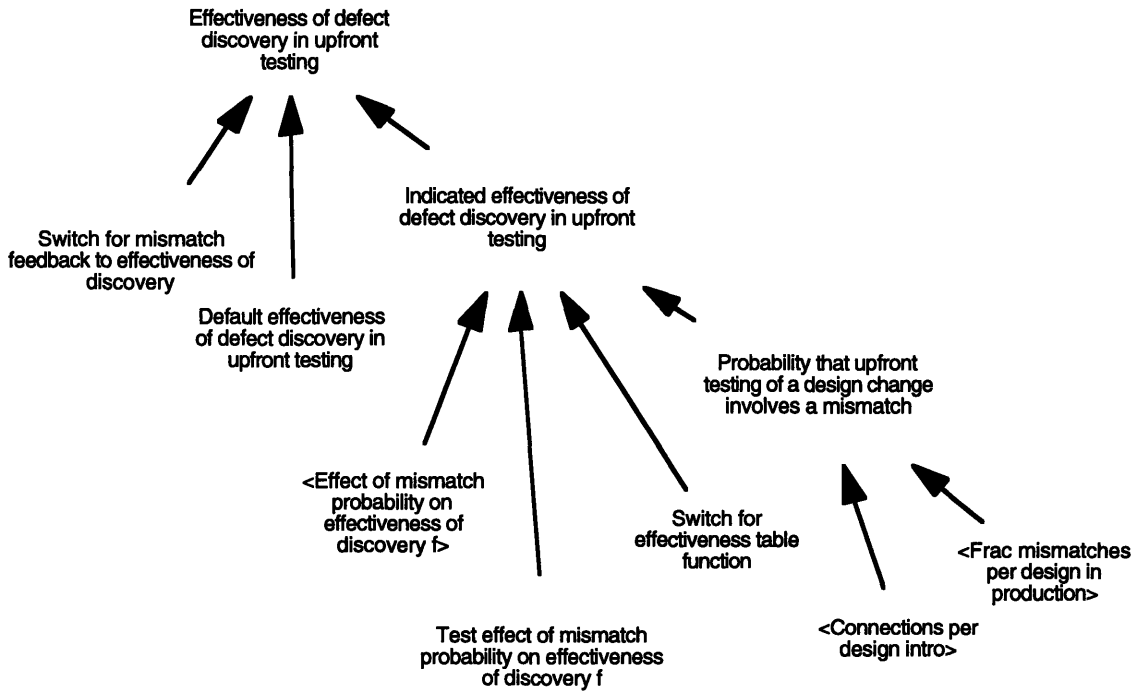
Open concern retirement rate = Frac designs in prod with open concerns* Designs retirement rate
~ Defects/Month
~ Part of a coflow.

Frac designs in prod with open concerns = zidz(Open concerns in production, Designs in production)
~ Defects/Design
~ Part of a coflow. Zidz means zero if division by zero. The structure is necessary because designs can reach zero.

Month to implement containment freeze = 10000
~ Months
~ Set so high so it never happens unless the test is run.

## Effectiveness of Early Discovery

Figure 66




The purpose of this sector is to calculate the effectiveness of defect discovery in upfront testing. A switch determines whether the value is set to a default value (10% of defects discovered, a rough estimate), or to a dynamic variable, called the *Indicated effectiveness*. The dynamic effectiveness is determined by a table function shown below. There are two table functions, the normal one for base runs, and a test table function. A switch determines which one is active. The input to the table function is the probability that upfront testing of a design change involves a mismatch, which is determined by the fraction of mismatches and the number of mating part connections for every design change.

Effectiveness of defect discovery in upfront testing =
(1-Switch for mismatch feedback to effectiveness of discovery)*
Default effectiveness of defect discovery in upfront testing+
(Switch for mismatch feedback to effectiveness of discovery*
Indicated effectiveness of defect discovery in upfront testing)
~ dimensionless
~ Switch structure allows for the turning off loop R2.

Default effectiveness of defect discovery in upfront testing = 0.1
~ dimensionless
~ Rough estimate.

Probability that upfront testing of a design change involves a mismatch = min(0.9,
1 - ((1-Frac mismatches per design in production)^Connections per design intro))

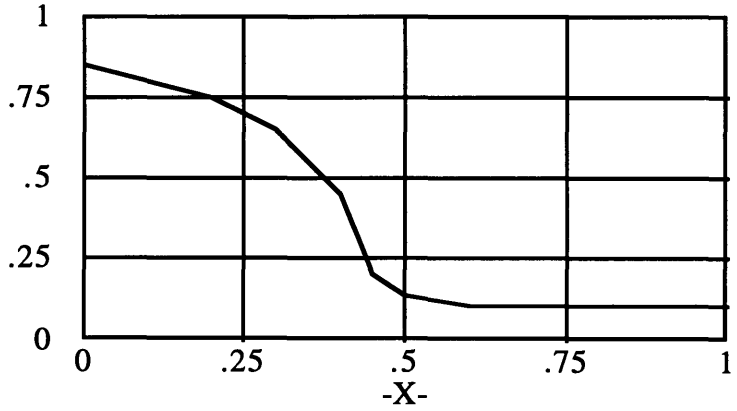
~ dimensionless
~ Min structure ensures that the probability does not fail extreme conditions testing. The structure comes from a basic probability equation for chance an event happens given multiple opportunities.

Indicated effectiveness of defect discovery in upfront testing =
(Switch for effectiveness table function*
Effect of mismatch probability on effectiveness of discovery f(Probability that upfront testing of a design change involves a mismatch)) +
((1-Switch for effectiveness table function)*
Test effect of mismatch probability on effectiveness of discovery f(Probability that upfront testing of a design change involves a mismatch))
~ dimensionless
~ Switch structure allows for the testing of two different table functions.

Switch for mismatch feedback to effectiveness of discovery = 1
~ dimensionless
~ Turns feedback on and off, for testing of loop strength. 1 is the base case.

BASE

Effect of mismatch probability on effectiveness of discovery f




The effect of mismatch probability on the effectiveness of defect discovery with upfront tools is operationalized as the function shown above. The domain of the function is the interval between 0 and 1, and represents the probability that upfront testing of a design change involves a mismatch . Between a 60% chance and a 100% chance, the effectiveness of discovery is 10%. That is, engineers will find only 10% of the defects. The value is based on a rough estimate. As the probability falls below 50%, effectiveness increases sharply and then less so. Even with no chance of a mismatch, effectiveness does not pass 85%.

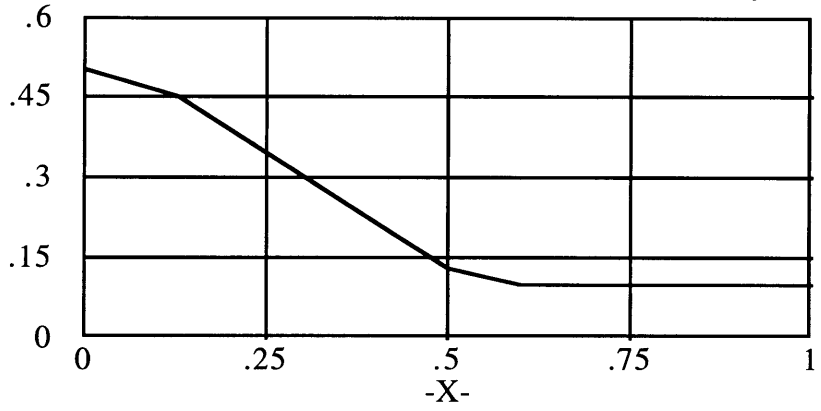
Effect of mismatch probability on effectiveness of discovery f(
[(0,0)-(1,1)],(0,0.85),(0.2,0.75),(0.3,0.65),(0.4,0.45),(0.45,0.2)
,(0.5,0.14),(0.6,0.1),(0.75,0.1),(1,0.1) )
~ dimensionless
~ Table function shows a non-linear improvement in effectiveness once the probability of encountering a mismatch while doing a design change reaches a certain level.

Switch for effectiveness table function = 1

~ dimensionless
~ Default condition uses the standard table function.

BASE

Test effect of mismatch probability on effectiveness of discovery f



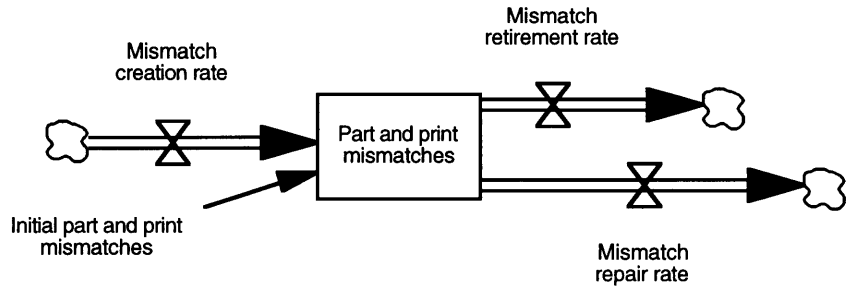

In the test table function, effectiveness increases less suddenly, and reaches a lower maximum than in the base case.

Test effect of mismatch probability on effectiveness of discovery f
([(0,0)-(1,1)],(0,0.5),(0.13,0.45),(0.5,0.13),(0.6,0.1),(0.75,0.1)
,(1,0.1) )
~ dimensionless
~ This is the test version of the table function. Effectiveness increases much more linearly, and not as much even with few mismatches.

## Mismatches

This sector keeps track of the number of part/print mismatches in the system.

Figure 67

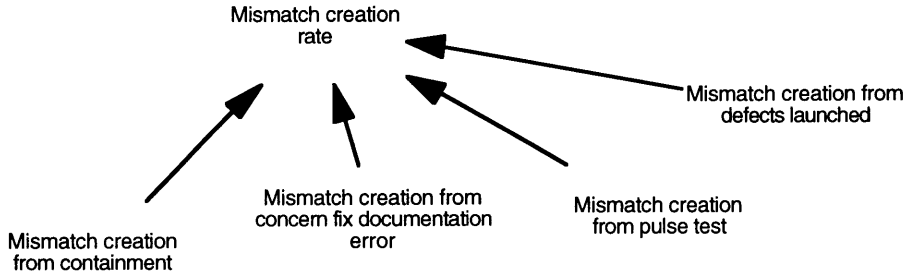



Mismatches increase with creation and decrease with retirement and repair. The initial value corresponds to a LaunchTech estimate of 20 - 30%.

Part and print mismatches = INTEG(Mismatch creation rate-Mismatch retirement rate
-Mismatch repair rate,Initial part and print mismatches)
~ Designs
~ Number of designs in production and testing for which the parts as produced do not match the prints on file.

Initial part and print mismatches = 2380
~ Designs
~ Close to 25% of designs mismatched initially.

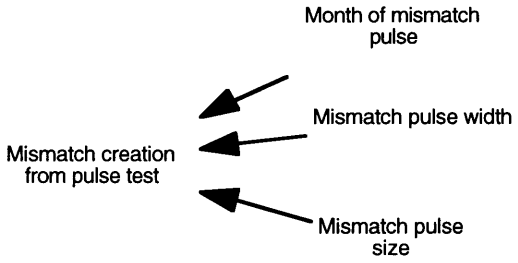
Figure 68




Creation of mismatches has four sources, outlined above.

Mismatch creation rate = Mismatch creation from containment+
Mismatch creation from concern fix documentation error+
Mismatch creation from pulse test+Mismatch creation from defects launched
~ Designs/Month
~ Four sources of mismatches.

Figure 69




The first sources of mismatch creation is a pulse test. The structure shown above pulses in or pulses out a certain number of mismatches, indicated by the pulse "size," in the month of the test. Although the variable is called "creation," the rate can be negative.

Mismatch creation from pulse test = Mismatch pulse size*
pulse(Month of mismatch pulse,Mismatch pulse width)
~ Designs/Month
~ Structure allows for a base case test, in which mismatches are pulsed in or out.

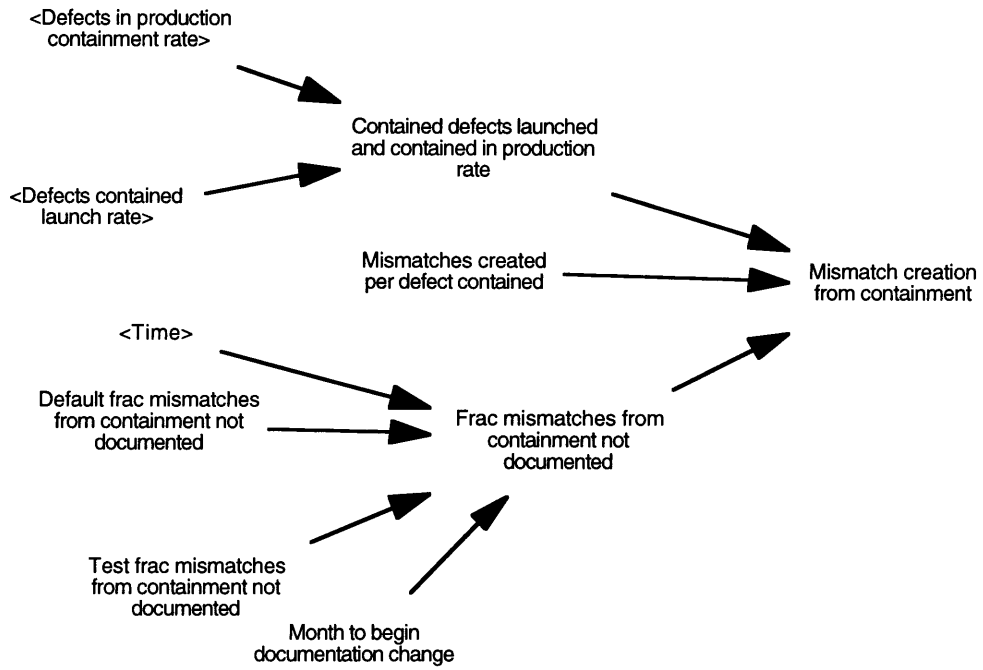
Mismatch pulse size = 0
~ Designs
~ Number is increased for testing.

Mismatch pulse width = 1
~ Month
~ Mismatches pulsed in over one month.

Month of mismatch pulse = 10000

~ Months
~ Set high so the test doesn't happen in the base case run.

Figure 70




The second source of creation is containing defects. Mismatches created is determined by the number of mismatches created for every defect contained times the rate of containment. For containing a defect to create a mismatch, the contained defect must not be documented. The fraction that are documented varies between two values: one, a default value, and two, a test value which is part of the "More Documentation" policy test. Both values are rough estimates based on interviews. The change between the values happens in a specific month.

The model assumes that a mismatch in testing is not yet a mismatch to be counted. Only when the mismatch is launched into production or created while in production does it register in the model.

Mismatch creation from containment = Contained defects launched and contained in production rate*Frac mismatches from containment not documented*Mismatches created per defect contained
~ Designs/Month
~ For a mismatch to be created, the containing of a defect needs to both create a mismatch and not be documented.

Contained defects launched and contained in production rate = Defects contained launch rate+Defects in production containment rate
~ Defects/Month

Frac mismatches from containment not documented = if then else(Time> Month to begin documentation change,
Test frac mismatches from containment not documented,
Default frac mismatches from containment not documented)
~ dimensionless
~ Structure allows for the More Documentation policy test.

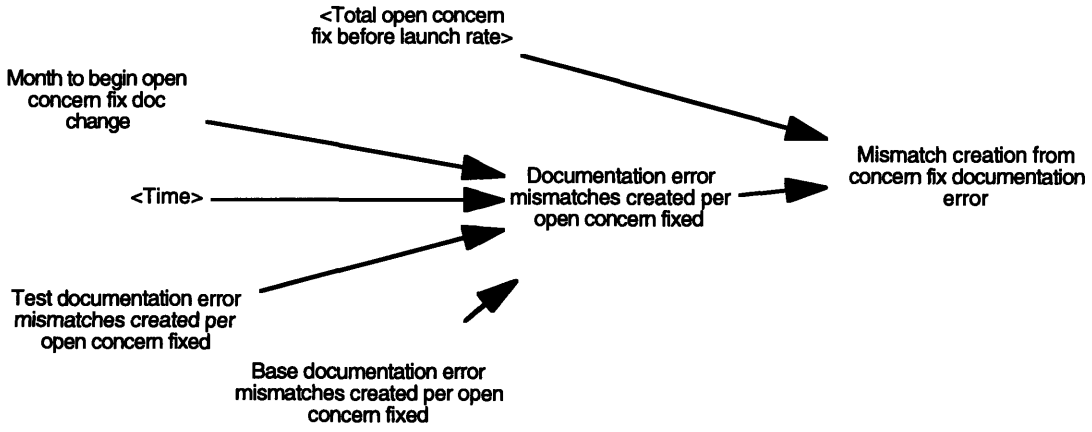
Default frac mismatches from containment not documented = 0.9
~ dimensionless
~ Rough estimate.

Test frac mismatches from containment not documented = 0.1
~ dimensionless
~ Ambitious level.

Month to begin documentation change = 1e+009
~ Months
~ Set so high so it is inactive unless test is run.

Mismatches created per defect contained = 0.9
~ Design/Defect
~ Rough estimate.

Figure 71




The third source of mismatch creation is documentation error. The creation rate is determined by the rate of fixing open concerns and the fraction of them not documented well enough to avoid a mismatch. The fraction varies between two values: one, a base value, and two, a test value that is used in the month that the test begins.

An alternative formulation for this structure would explicitly capture the time delay between someone submitting a design change and someone completing the design change, and the resulting interval when a mismatch existed.

Mismatch creation from concern fix documentation error =
Documentation error mismatches created per open concern fixed*
Total open concern fix before launch rate
~ Designs/Month
~ A source of mismatch creation, from design engineering not accurately or quickly documenting their changes to designs.

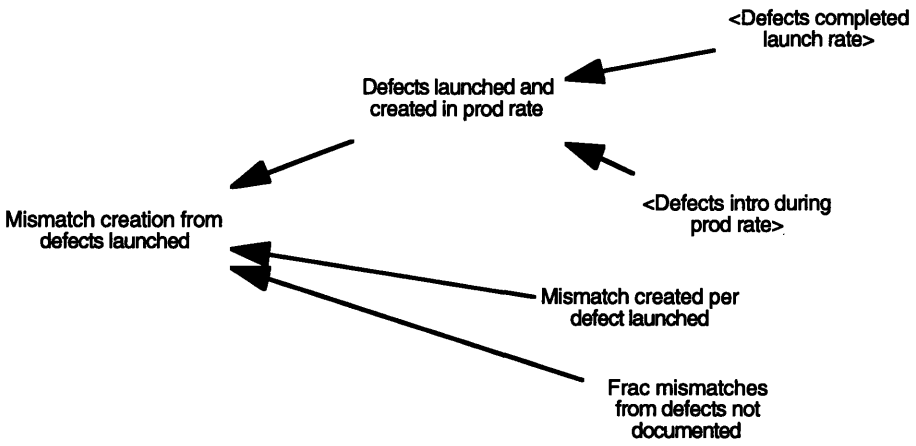
Documentation error mismatches created per open concern fixed =
if then else (Time>Month to begin open concern fix doc change,
Test documentation error mismatches created per open concern fixed,
Base documentation error mismatches created per open concern fixed)
~ Design/Defect
~ Formulation allows for the More Documentation policy run.

Month to begin open concern fix doc change = 10000
~ Months
~ Set so high so the test is not employed in the base run.

Base documentation error mismatches created per open concern fixed = 0.1
~ Design/Defect
~ 10% of design changes create mismatches. rough estimate.

Test documentation error mismatches created per open concern fixed = 0
~ Defects/Design
~ perfect documentation.

Figure 72



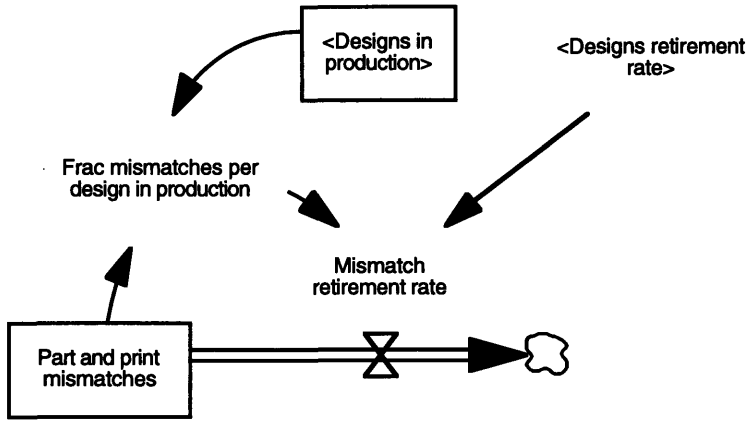

The fourth source of mismatch creation is defects entering production. Mismatches created is determined by the total number of defects launched or created in production and the number of mismatches created for every defect launched. The equation also includes the fraction of the defects that are not documented, but the value will always be one -- the defects are not discovered, and thus no one could document them.

Mismatch creation from defects launched = Defects launched and created in prod rate*
Frac mismatches from defects not documented*Mismatch created per defect launched

~ Designs/Month
~
Defects launched and created in prod rate = Defects completed launch rate+
Defects intro during prod rate
~ Defects/Month
~
Frac mismatches from defects not documented = 1
~ dimensionless
~ These defects are not discovered, thus no one could document them.

Mismatch created per defect launched = 0.9
~ Design/Defect
~ rough estimate

Figure 73

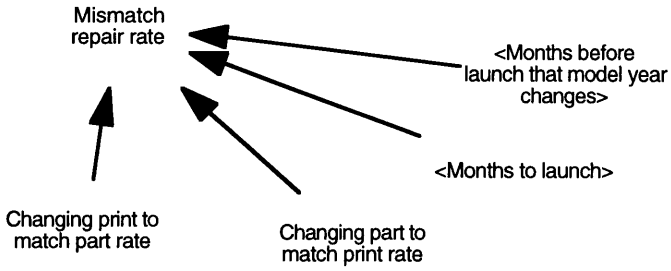



The first outflow of mismatches is the retirement rate. Similar to previous structures, it is part of a coflow structure driven by the retirement of designs. The number of mismatches that retire every year is equal to the number of designs that retire times the fraction of designs that are mismatched. The model assumes an equal distribution of mismatches across designs.

Mismatch retirement rate =Designs retirement rate*
Frac mismatches per design in production
~ Designs/Month
~ Part of a coflow.

Frac mismatches per design in production = zidz(Part and print mismatches,
Designs in production)
~ dimensionless
~ Part of a coflow.

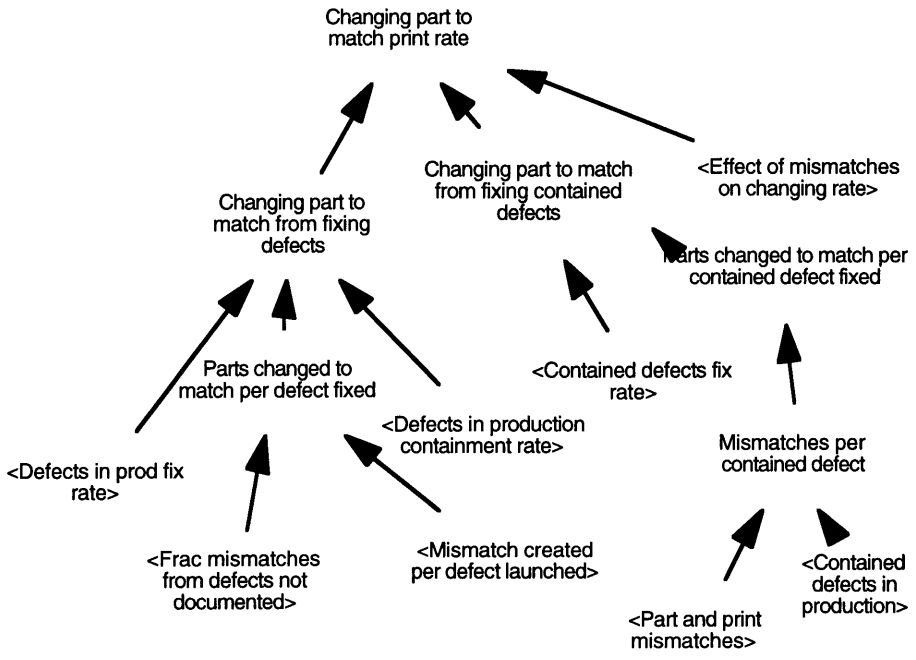
Figure 74




The repair rate is determined by two other rates: changing print to match part and changing part to match print. During the model year change, the repair rate is set to zero because the discrete flush of defects out of the stock will empty the entire stock in one time step. Any additional outflow such as through repairing would send the stock negative.

Mismatch repair rate = if then else(Months to launch=
Months before launch that model year changes,0,
Changing part to match print rate+
Changing print to match part rate)
~ Designs/Month
~ The rate is set to zero when the discrete "flush" of designs from the current year to the production year happens, so the stock does not go negative.

Figure 75




The rate of changing parts to match prints is determined by two main sources: from fixing defects and from fixing contained defects. The rate driven by fixing defects is determined

by two rates, defects fixed while in production and defects contained while in production. For each of those defects, a number of parts will be changed to match their print. The number changed to match is determined by the fraction created and fraction not documented.

The rate of changing parts driven by fixing contained defects is determined by the contained defect fix rate and the number of parts changed to match for every fix. The number changed to match for every fix is simply the fraction of mismatches per contained defect.

The overall changing rate is also determined by the number of existing mismatches. If there are few mismatches then the rate of fixing will fall. The structure is depicted in figure 76.

Changing part to match print rate = (Changing part to match from fixing contained defects+
Changing part to match from fixing defects)*Effect of mismatches on changing rate
~ Designs/Month
~ An alternative to changing print to match parts. Comes from fixing contained defects. The inclusion of an "effect" ensures that mismatches will be fixed only when there are mismatches present.

Changing part to match from fixing defects = (Defects in prod fix rate+
Defects in production containment rate)*Parts changed to match per defect fixed
~ Designs/Month
~ Fixing defects fixes mismatches.

Parts changed to match per defect fixed = Frac mismatches from defects not documented*
Mismatch created per defect launched
~ Design/Defect

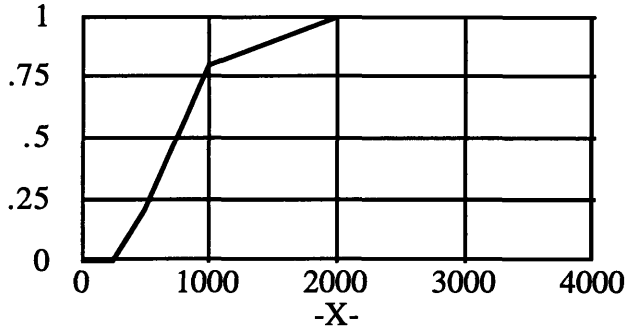
Changing part to match from fixing contained defects = Contained defects fix rate*
Parts changed to match per contained defect fixed
~ Designs/Month
~ Fixing contained defects fixes mismatches.

Parts changed to match per contained defect fixed = min(Mismatches per contained defect,1)
~ Design/Defect
~ This structure ensures that no more than one mismatch will be fixed per fixing of a contained defect.

Mismatches per contained defect = Part and print mismatches/
Contained defects in production
~ Design/Defect

BASE

Effect of mismatches on changing rate f

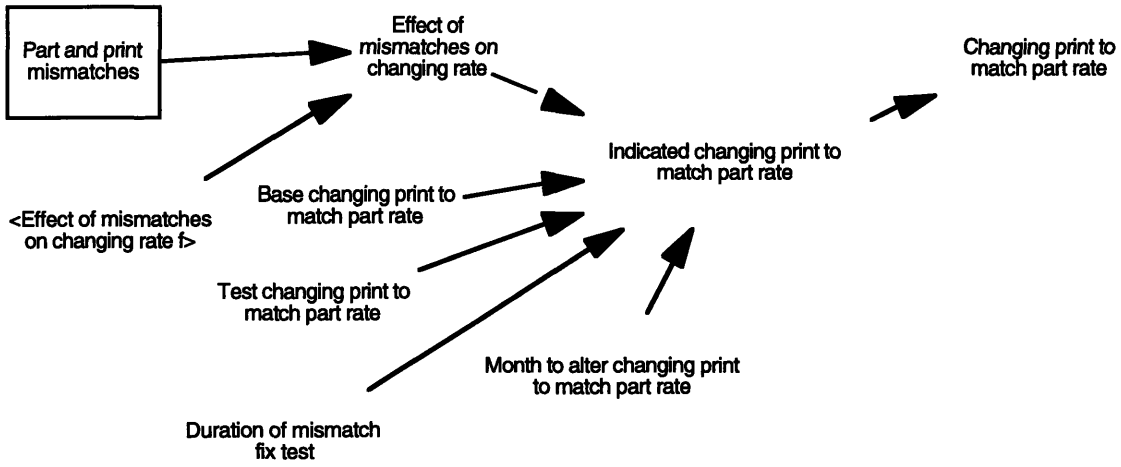



The effect of mismatches on the rate of fixing mismatches is operationalized as the function shown above. The domain of the function is the interval between 0 and 4000, and represents the number of mismatches. As the number of mismatches falls below 2000 (the initial value is 2380), fixing mismatches has less and less effectiveness. When there are only 250 mismatches left, they are so difficult to find that the effect is 0.

Effect of mismatches on changing rate = Effect of mismatches on changing rate f(Part and print mismatches)
~ dimensionless

Effect of mismatches on changing rate f ([(0,0)-(4000,1)], (0,0),(250,0),(500,0.2),(1000,0.8),(2000,1),(3000,1),(4000,1))
~ dimensionless
~ Table function -- as mismatches go to zero, the fixing rate goes to zero too.

Figure 76




The second way of repairing a mismatch is by changing the print to match the part. The rate is determined one of two values: the base rate, or the test rate. For a test, the rate changes to the second rate in the indicated month, for a certain duration. The change rate is

further determined by the effect of mismatches on the changing rate, because the rate of fixing mismatches will fall as there are fewer mismatches. The table function is shown in the previous section.

Changing print to match part rate = Indicated changing print to match part rate
~ Designs/Month
~ The second way a mismatch can be fixed, by changing the print to match how the part is actually being produced.

Indicated changing print to match part rate = Base changing print to match part rate+
(pulse(Month to alter changing print to match part rate,Duration of mismatch fix test)*
Test changing print to match part rate*Effect of mismatches on changing rate)
~ Designs/Month
~ This structure allows for the Fix Mismatches policy test.

Base changing print to match part rate = 0
~ Designs/Month
~ Turned on for a test.

Duration of mismatch fix test = 36
~ Months
~ For test. Rough estimate.

Test changing print to match part rate = 20
~ Designs/Month
~ Rough estimate.

Month to alter changing print to match part rate = 10000
~ Months
~ Set so high so it never happens unless explicitly tested.

## Totals

This sector totals the stocks and flows into aggregate numbers. The primary outputs are total defects (of both types) and total open concerns (both types). Because the equations are simple to understand and merely handle accounting, diagrams are not provided.

Total containment rate = Open concern contained before launch rate +
Open concern contained during prod rate
~ Defects/Month

Total defects in production = Contained defects in production+Defects in production
~ Defects

Frac total defects in designs in production = zidz(Total defects in production,
Designs in production)
~ Defects/Design

Total current mating part defect intro rate = Defects intro before B1 rate+
Defects intro before B2 rate+Defects intro before B3 rate+Defects intro before launch rate
~ Defects/Month

Total current open concerns = Open concerns awaiting Build 1+
Open concerns awaiting Build 2+Open concerns awaiting Build 3+

Open concerns in designs completed
~ Defects
~ Current means the year before launch.

Total open concern fix rate = Open concern fix during prod rate+
Total open concern fix before launch rate
~ Defects/Month

Total current defects = Defects awaiting Build 1+Defects awaiting Build 2+
Defects awaiting Build 3+
Defects in designs completed
~ Defects

Total open concern discovery rate = Open concern disc before B1 rate+
Open concern disc during prod rate+Open concern disc in B1 rate+
Open concern disc in B2 rate+Open concern disc in B3 rate
~ Defects/Month

Total Defect intro during year rate = Defects intro before B1 rate+
Defects intro before B2 rate+Defects intro before B3 rate+
Defects intro before launch rate+Defects intro during prod rate
~ Defects/Month

Total open concern fix before launch rate = Open concern fix before B1 rate+
Open concern fix before B2 rate+Open concern fix before B3 rate+
Open concern fix before launch rate
~ Defects/Month

Total current designs = Designs awaiting Build 1+Designs awaiting Build 2+Designs awaiting Build 3+Designs Completed
~ Designs

## Simulation Control Parameters

FINAL TIME = 400
~ Month
~ The final time for the simulation.

INITIAL TIME = 0
~ Month
~ The initial time for the simulation.

SAVEPER = 1
~ Month
~ The frequency with which output is stored.

TIME STEP = 0.00390625
~ Month
~ The time step for the simulation. So small because the discrete "flush" structures make the model prone to dt error.

## References


Adler, P. (1995). 'Interdepartmental Interdependence and Coordination: The Case of the Design/Manufacturing Interface.' *Organization Science*. Vol. 6, No. 2, March-April 1995.

Adler, et al. (1995). 'From Project to Process Management: An Empirically-Based Framework for Analyzing Product Development Time.' *Management Science*. 41:3:458-484.

Adler, P., A. Mandelbaum, V. Nguyen, and E. Schwerer (1996). 'Getting the Most out of Your Product Development Process.' *Harvard Business Review*, March-April 1996.

Bedworth, D., M. Henderson, and P. Wolfe (1991). *Computer-Integrated Design and Manufacturing*. New York, McGraw-Hill Inc.

Carter, D. E. and B.S. Baker (1992). *Concurrent Engineering: The Product Development Environment for the 1990s*. Massachusetts, Addison-Wesley.

Clark, K. B. and T. Fujimoto (1991). *Product Development Performance: Strategy, Organization, and Management in the World Auto Industry*. Cambridge Massachusetts, Harvard Business School Press.

Clark, K. B. and S. Wheelwright (1993). *Managing New Product and Process Development -- Text and Cases*. New York, The Free Press.

Dean, J. W. and G. Susman (1989). 'Organizing for Manufacturable Design.' *Harvard Business Review*. January-February 1989.

Deming, W. E. (1986). *Out of the Crisis*. Cambridge, MIT Press.

Easton, G. and S. Jarrell (1995). 'The Effects of Total Quality Management on Corporate Performance: An Empirical Investigation.' Working Paper, University of Chicago, Chicago, Illinois, 60637.

Engelke, W. D. (1987). *How to Integrate CAD/CAM Systems -- Management and Technology*. New York, Marcel Dekker Inc.

Ettlie, J.and H. Stoll (1990). *Managing the Design-Manufacturing Process*. New York, McGraw-Hill Inc.

Ford, D. N. (1995). *The Dynamics of Project Management: An Investigation of the Impacts of Project Process and Coordination on Performance*. Unpublished doctoral dissertation. Sloan School of Management, MIT, Cambridge, MA.

General Accounting Office (1991). 'US companies improve performance through quality efforts.' GAO/NSIAD-9-190 (2 May).

Greenwood, N. (1988). *Implementing Flexible Manufacturing Systems*. London, MacMillan Education Ltd.

Hartley, J. R. (1990), *Concurrent Engineering.* Massachusetts, Productivity Press.

Harvard Business Review (1991). *Managing Product Lifecycles From Start to Finish.* Cambridge, Harvard Business School Press.

Hawkes, B. (1988). *The CADCAM Process.* London, Pitman Publishing.

Jessen, S. (1992). *The Nature of Project Leadership.* Scandanavian University Press. Oslo, Norway.

Krubasik, E.G. (1988). 'Customize Your Product Development.' *Harvard Business Review.* November-December 1988.

Nevins, J. and D. Whitney (1989). *Concurrent Design of Products and Processes.* New York, McGraw-Hill Inc.

Patterson, M. L. (1993), *Accelerating Innovation: Improving the Process of Product Development.* New York, Van Nostrand Reinhold.

Repenning, N. P. (1996). *The Improvement Paradox: Three Essays on Process Improvement Initiatives.* Unpublished doctoral dissertation. Sloan School of Management, MIT, Cambridge, MA.

Richardson, G. P. and A. Pugh (1981). *Introduction to System Dynamics Modeling with Dynamo.* MIT Press. Cambridge, MA.

Shiba, S., D. Walden, and A. Graham (1993). *A New American TQM. Four Practical Revolutions in Management.* Portland, Oregon, Productivity Press.

Sterman, J.D. (1994). 'Learning in and about complex systems.' *System Dynamics Review,* Summer-Fall 1994.

Sterman, J. D., N. Repenning, and F. Kofman (1994). 'Unanticipated Side Effects of Successful Quality Programs: Exploring a Paradox of Organizational Improvement,' Working Paper #3667-94-MSA, Sloan School of Management, Cambridge MA 02142.

Thomas, H. and C. Napolitan (1994). *The Effects of Changes on Labor Productivity: Why and How Much.* Construction Industry Institute. Austin, Texas.

Ulrich, K. T. and S. Eppinger (1994). *Methodologies for Product Design and Development.* New York, McGraw-Hill.

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

Whitney, D. E. (1988). 'Manufacturing by Design.' *Harvard Business Review*, July-August 1988.