PHANTOM WORK: DESIGN ITERATION TIMING IN NEW PRODUCT DEVELOPMENT

by

Daniel J. McCarthy

B.S. Organizational Leadership, United States Military Academy, West Point (1990)
M.E. Systems Engineering, University of Virginia, Charlottesville (1999)

SUBMITTED TO THE SLOAN SCHOOL OF MANAGEMENT
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY IN MANAGEMENT

AT THE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2008

Signature of Author.....
Sloan School of Management
August 25, 2008

Certified by.....
Nelson Repenning
Associate Professor of Management
Committee Chair and Research Advisor

Accepted by.....
Ezra Zuckerman
Chair, Doctoral Program Committee

PHANTOM WORK: DESIGN ITERATION TIMING IN NEW PRODUCT DEVELOPMENT

by
Daniel J. McCarthy
Submitted to the Sloan School of Management
On May 1, 2008
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Management Science

ABSTRACT

As companies compete to gain market share, increase profits and affect growth they often turn to concurrent engineering in an effort to bring new products to the market more quickly. Despite many anecdotal success stories, implementation of concurrent engineering can often prove difficult. As the pressure to bring new products to market increases, companies often compress their design iteration cycle times in an effort to develop products more quickly. In many cases, design cycles may overlap creating situations where learning opportunities (e.g. through testing) are missed and/or ignored. More perversely, compressing design iteration cycles can cause the creation of "phantom errors" and unnecessary rework as concurrent design activities iterate at different speeds. In this research, I use a system dynamics approach to develop a stylized simulation model of the design-build-test iteration cycle to explore the effects of cycle timing on learning. Specifically, I look at the frequency and timing of integration (build) test events and their effect on new product delivery time, quality, and development cost. This research adds to the existing literature in new product development, concurrent engineering, and system dynamics. Ultimately, the results serve to inform new product development project managers of the implications of design iteration timing on project performance and assist in the scheduling of integration events.

Key Words:

Committee Chair: Nelson Repenning
Title: Associate Professor of Management Science and Organization Studies

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

"Great men rejoice in adversity, just as brave soldiers triumph in war."
-- Seneca, Roman Philosopher & Dramatist

An endeavor such as this is never an individual effort. Accordingly, I would like to thank a number of people who have encouraged, guided, cajoled and inspired me along the way.

First, I should thank the United States Army, the United States Military Academy, in particular the Department of Systems Engineering, and Brigadier General (Retired) Mike McGinnis for giving me the opportunity of a lifetime to study at the Massachusetts Institute of Technology.

I would also like to thank the other PhD students from the Sloan School of Management at MIT with whom which I had the pleasure to work and from whom I had the good fortune to learn. I'd especially like to recognize the other PhD students in the System Dynamics group: Hazhir Rahmandad, Jeroen Struben, Gokhan Dogan, Timothy Quinn, John Lyneis and Joe Hsueh. They all enriched my experience and learning at MIT in countless ways by providing both good friendship and great counsel.

Next, I would like to thank the folks at the Company that served as the basis of my research. I was very fortunate to work with people from the organization who were both interested in my work and extremely generous with their time. In particular, I am deeply indebted to Tony Wilcox and Charles Cooksey who spent countless hours with me both on-site and on the phone allowing me to bounce my ideas off of them and answering my many questions. I'm also grateful for the support of the many others at the Company who welcomed me into their organization and freely shared their time, thoughts and experiences with me.

Of course I am deeply indebted to the members of my dissertation committee who have been most patient and understanding as I have taken an atypical route to reach this point in my academic career and, in particular, my PhD program. As I tried to balance my responsibilities as a father, an Army officer, and student Professors Nelson Repenning, John Sterman, and Warren Seering have been nothing but helpful, accommodating, and understanding.

I will be forever thankful to Professors John Sterman and Nelson Repenning for giving me the wonderful opportunity to join the System Dynamics PhD program. They work very hard and are quite successful at making the SD group a great place to study, learn and grow. I could not have asked for a better experience.

As my dissertation advisor, Nelson Repenning deserves special thanks for the countless hours he spent looking at the many versions of my model and listening to me complain about the myriad of software glitches I encountered along the way. I am most thankful for his Herculean efforts to reel in my modeling complexity and his constant mantra of "Keep it simple". Despite this wise and often repeated counsel, I'm not sure I ever was able to achieve the simplicity he was looking for.

Lastly, I must thank the most important people in my life without whom I would not be where I am today. My family has sacrificed a lot so that I could take on this great endeavor. My boys – Matthew, Shane, Michael, and Jack – have had to contend with life without Dad on many occasions as I was off at work, school or holed up in the library reading and writing. Since we first started dating 25 years ago, my beautiful and loving wife, Patty, has and continues to pick up the slack in my life. She fills in for me in the many places where I fall short and has supported me in countless ways. I can never thank her enough for her love as well as the patience and understanding she has shown to me as I have dragged her and our family around the United States, indeed around the world, as I have pursued my career. Anything of significance that I have or will accomplish is due in no small measure to her.

Chapter 1: Introduction

> ***"The first guy to market cleans up. The second guy does OK, and the third guy barely breaks even. The fourth guy loses money."***
>
> -- Bill Schroeder, Chairman of the Board and Chief Executive Officer, Oxford Semiconductor[1]

MOTIVATION

Every year companies spend millions of dollars on research and development with the hope of successfully bringing new products to the market. In fact, more money is spent per day on product innovation in the G5 countries than was spent during the entire first Gulf War (Cooper 2001). In the automotive industry, the customer base expects new products every year in the form of model year updates, the occasional overhaul of an existing vehicle line, and the introduction of completely new vehicles. It is critical to companies in this industry to be able to bring development projects in "on time" as well as under budget while meeting quality objectives to meet the annual launch of new vehicles and thus gain their share of the market (Haddad 1996).

As companies compete in an increasingly global and competitive market, bringing new products to the market more quickly has become critical to gaining market share, increasing profits and creating and sustaining growth. The company that can develop products more quickly than its competitors can introduce new products sooner than its competitors and thus gain first mover

---

[1] Whiting, R. (1991). "Product Development as a Process: Electronic Industry Executives are Learning that Product Development is as Important as Quality of Manufacturing Prowess." Electronic Business **30**.

advantage. Early entry into the market often leads to higher profitability as revenues are generated sooner and the early company is able to grab a greater share of the overall sales. This is especially true when the product's life cycle (and thus total number of sales) is limited. Alternatively, the faster company may delay the start of a new product development project in order to gain additional market data. In this case the faster company is still able to introduce its product to the market at the same time as its slower competitors but it introduces a product that is more aligned with the market demands (Wheelwright and Clark 1992). Presumably this additional market data leads to a higher success rate of new product entries into the market and reduces the number of "misses" as a shorter time to market reduces the chances that the market will have changed significantly during the course of the development project.

(Cooper 2001) describes the competitive landscape of new product development as a battleground where winning is everything. In his view of *new products warfare*, the many companies competing for a better position, a higher market share or new territory in the marketplace are combatants and speed is considered to be a vital weapon (Cooper 2001). Practitioners and researchers alike (Bussey and Sease 1988; Stalk 1988; Blackburn 1991; Symonds 1991; McGrath 1996; Smith and Reinertsen 1998; Eppinger 2001; Langerak and Jan Hultink 2005) agree that speed, or a reduced time to market, is critical in determining success in new product development. In fact, recent research provides empirical evidence that indicates that companies that are able to develop products more quickly show better company financial performance and that development speed has a greater impact on financial performance than either new product profitability or R&D expenditures (Langerak and Jan Hultink 2005).

THE PROBLEM

It is no secret that many companies struggle with managing projects especially in the area of new product development. Given the vital nature of new product development to both the short-term health and long-term viability and growth of a company, it begs the question: *Why do companies with a recognized history of bringing new product development projects in late, over budget and with quality below expectations continue to do so?*

Over the past couple of decades, companies have undertaken any number of new product development improvement efforts aimed at improving their ability to deliver successful products to the market in a timely and cost-efficient manner. The results of a recent survey of new product development best practices sponsored by the Product Development & Management Association (PDMA) indicate that of all the new product development improvement efforts that have been undertaken, the most progress made has been in those aimed at reducing product development cycle times. In fact, on average the more than 383 companies surveyed across a variety of industries had reduced the development cycle time of their more innovative projects by approximately 1/3 during the previous five years (Griffin 1997).

Concurrent Engineering

There are any number of approaches in use by companies to accelerate their new product development efforts and reduce the time to market of their new products (Gold 1987; Millson, Raj et al. 1992; Droge, Jayaram et al. 2000; Langerak and Jan Hultink 2005). One of the most commonly cited techniques for speeding up new product development is concurrent engineering where development tasks are competed in parallel as opposed to sequentially in an effort to reduce the overall development time. (Takeuchi and Nonaka 1986) liken the concurrent development

process to a rugby game where a multi-disciplinary team with members from marketing, design, engineering, and manufacturing move "the ball" down the field working together in parallel.

The most common form of concurrent engineering discussed in the project management and new product development literature falls under the rubric of "activity concurrency" and involves the overlapping of development stages. For example, rather than waiting until the detailed design of a new product is completed one might begin the design/redesign of the corresponding manufacturing process. An even more basic form of concurrent engineering occurs when development tasks are overlapped or executed in parallel within a single development stage. A common example is when design tasks within the detailed design stage are subdivided and carried out simultaneously by different people.

Research has shown that the use of concurrent engineering can lead to dramatic reductions in development lead time (Takeuchi and Nonaka 1986) with some companies reporting as much as a 60% reduction in lead time (Rosenblatt and Watson 1991). While concurrent engineering offers the potential for greatly reducing the development time of a new product, the overlapping of development tasks does not come without risks. New product development managers must carefully consider which development tasks can be reasonably executed in parallel and the degree to which they should be overlapped understanding the potential for rework (design iterations) and additional cost. In fact, (AitSahlia, Johnson et al. 1995) show that given the uncertainty found in the product development environment there exist conditions under which the serial, or sequential, approach will always be less costly than the concurrent approach to engineering.

The Integration Challenge

Proponents of concurrent engineering recognize the significant integration challenge that is inherent when overlapping development stages. A common solution to integrating the efforts of

members of different functional groups (e.g. marketing, engineering, and manufacturing) is the creation of cross-functional project teams that include members from each of the functional areas of the organization that impact on the development project. However, the degree of commitment from each of the team members in terms of man-hours devoted to the project varies greatly as does the amount of authority and control of team activities afforded the project manager of the team.

When considering the overlapping of activities within a single development stage, specifically the parallel design of different subsystems in the detailed design phase, the project manager is faced with the dilemma of how to bring about effective integration between individual design engineers. In this case, individual engineers are working on different but coupled subsystems or components and the challenge is to integrate their design work so as to minimize the amount of time and effort (e.g. rework) required to complete the development project while meeting quality objectives. If two engineers designing separate but coupled components work in isolation and fail to integrate their design work, problems or issues of fitment and function are sure to arise. Unfortunately, because of the time pressure that many design engineers face they often fail to do the necessary upfront and regular coordination with other engineers in order to effectively integrate their designs with those of coupled components and subsystems.

The lack of day to day integration between engineers can lead the new product development project manager to increase the number of team level design reviews as a means of affecting integration. Preparing for these design reviews takes time and leaves engineers with less time to do actual engineering work, including integration. This, of course, increases the chances of integration problems occurring which, in turn, is likely to cause the project manager (or perhaps higher management) to call for even more design reviews.

Alternatively, project managers often use prototype builds as a means of integrating the various components and subsystems by building the entire system and then testing it for both fitment and function. It is when the engineers are forced to convert their designs into physical parts and integrate them with coupled components by building and testing the full system that they find where things don't fit and/or function together. These integration problems, now known and tracked at the project if not the organizational level, become very salient and force the engineers to work together, to coordinate and integrate their designs in an effort to resolve the known issues.

One can think of the detailed design phase of the development process as a series of design-build-test cycles or design iterations. At the core of the design-build-test cycle is the prototyping of designs and subsequent testing as described above. It is from this building and testing of designs that much of the learning within a product development project takes place. Indeed for reasons discussed above, these build events may be the only means by which a project manager can affect true integration of the engineering design work being done by different engineers. Additionally, individual components are built and tested in realistic conditions and the overall product can be verified and validated as to whether or not it meets the customer requirements. As Wheelwright and Clark point out, increasing the rate and amount of learning that takes place in each of the design-build-test cycles and then effectively sequencing and linking these cycles together "permits an organization to shorten the duration and number of cycles needed in order to develop high quality products and manufacturing processes and get to market significantly faster than competitors" (Wheelwright and Clark 1992).

Given the discussion above, the instinctual response of the project manager trying to bring about effective integration of components and subsystems in his or her new product development project might be to reduce the time between the system integration (e.g. prototype build) events

thus reducing the duration of the associated design-build-test cycles of learning. Faced with the integration challenge outlined above and since the build events are where most of the learning takes places and integration problems are discovered, a project manager may indeed choose to increase the number of build events. However, these build events and learning cycles are not without cost. First and foremost, there are direct costs to the organization. These include the cost of prototyping parts (whether built in-house or purchased through a supplier), the cost to shut down and reconfigure the manufacturing and production lines if used for building and assembling the full system prototype, and the cost of the use of test equipment, facilities and personnel.

Less recognized but perhaps more important are the indirect costs of executing these design-build-test learning cycles. There is significant time and effort required of the design engineers to prepare for and execute a build event and plan for and monitor the subsequent testing. This time and effort reduces the amount of time available for the engineer to do "real" engineering work. It reduces the amount of time engineers have available to analyze the results of the tests, to conduct fault analysis (DFMA), to make necessary design changes, and lastly to coordinate with the engineers working on coupled components or subsystems to integrate their designs. It reduces the time available to do this "real" engineering work not only on the project at hand but on other development projects to which he or she may be assigned as well. Additionally, in a multi-project environment where test resources are limited, executing more frequent learning cycles for one development project reduces the availability of resources (e.g. test facilities) for other projects.

Testing takes time. In the automotive industry it may take months to accumulate the test miles necessary to properly evaluate a design. However, as the test miles accumulate, failures occur and the number of issues to be resolved increases. As the number of issues requiring resolution increases, so too does the pressure to execute another design-build-test cycle to confirm that the

known issues have been resolved and no new problems or issues arise as a result of the changes made to resolve the known issues. This pressure to "build again" is particularly acute as the launch date of a product approaches. In many cases, the pressure to execute another design-build-test cycle is so great that the project manager schedules another build event to be executed before the completion of the current testing.

A decision to execute another build event before the completion of the previous learning cycle leads to overlapping design-build-test cycles and thus having multiple design prototype versions undergoing testing and analysis at the same time. This can prove problematic as design engineers "lose track of the project status, which problems have been solved and which should be given top priority" (Wheelwright and Clark 1992). However, faced with the pressure to reduce the time to market and an accumulation of identified issues to resolve from previous design-build-test cycles, companies may opt to overlap these cycles in an attempt to speed up learning.

Phantom Work

In addition to the increased burden on individual engineers and the additional direct and indirect costs to the organization, overlapping design-build-test cycles can lead to the perverse situation where unnecessary or *phantom* work is created. This phantom work arises in situations where different components and subsystems that are developed in parallel iterate through the design-build-test cycle at different speeds. The amount of phantom work generated is a function of the individual iteration speeds of the various subsystems, the speed imbalance between them, and the timing of the iteration cycles. There are two sources of this phantom work. The first source is unnecessary rework that is created when some components are ready for a build event and others are not. The second source of phantom work is the redesign of components after their design has been released for prototyping and before the next set of test results is available. While phantom

work will be discussed in greater detail in the next chapter, I submit that this unnecessary work can be treated as a cost of overlapping design-build-test cycles and should be considered by managers when setting the timing of integration (or build) events.

"Phantom" work presents a "real" problem for companies engaging in new product development. Clearly, doing unnecessary work has a direct cost to the organization in that engineers' time is being wasted and other resources (e.g. design, prototyping and test equipment) may be unnecessarily consumed. Indeed, with improved timing of design iteration cycles it might be possible to reduce the amount of unnecessary work and quite possibly reduce the number of learning cycles required to achieve a desired product quality level. This reduction is particularly important given the significant costs, direct and indirect, associated with build events. In addition, companies should consider how the effort and resources expended on phantom work might be better utilized – perhaps applied to another new product development project.

The Research Question

The integration challenge that results from implementing concurrent engineering and the potential for creating unnecessary or phantom work naturally leads to the question facing companies, research and development organizations, and new product development project managers in particular: ***How often should we execute a design-build-test cycle in order to minimize the new product development time and cost while attaining a desired level of quality?*** In particular, how many prototype build events should be executed given a fixed deadline (i.e. product launch date) and disparate design iteration cycle times between coupled subsystems that are developed concurrently? Furthermore, how should these build events be timed (or spaced out) in order to maximize new product development project performance? How are the answers to these questions affected by the degree to which the design iteration cycle times for the coupled

subsystems are different and the degree to which the subsystems are coupled? Lastly, what is the impact of adding resources to a given project on the overall project performance?

RESEARCH APPROACH

The purpose of this research is to examine the dynamic effects of varying the length and timing of the design iteration (learning) cycles in new product development on project performance. In this thesis, I examine the research questions outlined above in the context of a new product development environment in the automotive industry using a system dynamics modeling approach. The system dynamics methodology, originally developed by Forrester ((Forrester 1961), is particularly useful for modeling complex systems that include feedback loops, delays and nonlinear relationships between system variables all features of the new product development environment. A system dynamics modeling approach has been used successfully to study the management of large scale projects in general (Cooper 1980) as well new product development projects in particular (Cooper 1980; Ford 1995; Ford and Sterman 1998; Repenning 2001; Black 2002; Ford and Sterman 2003a).

In this thesis, I develop and analyze a stylized system dynamics simulation model based on field research that I conducted at a midsize company in the automotive industry. The simulation model developed here builds off of the previous system dynamics research in project management leveraging the model structure from existing product development models. The simulation model presented in this research captures the key features of the new product development process in use at the company I worked with over a two and half year period. It incorporates both the formal and informal development processes and decision rules in place. The simulation focuses on the detailed design stage of the development process and captures the dynamics of the design-build-

test learning cycles discussed above. The stylized model considers the detailed design of two different components, or subsystems, which are coupled together and developed in parallel. It considers the development of paper designs, the prototyping of parts, the ability to bench test designs/parts in order to check for component errors, and the execution of full vehicle builds and subsequent testing to check for both component and integration errors. Additionally, the model accounts for the rework that must be done by individual engineers in order to fix known errors. It also accounts for the required coordination among engineers that must be done in order to fix an integration error that exists between components. The model considers schedule pressure, a key feature of project management and product development, and its effects on worker productivity, the quality of work, and the rate at which individual parts are bench tested prior to a full vehicle build event. Lastly, the model accounts for the allocation of resources by explicating the decision rules used by individual engineers to allocate their time between the competing demands of completing initial designs, reworking designs with known errors, and coordinating known integration errors. The simulation model is used as the basis for analysis of the proposed research questions and to demonstrate the creation of phantom work when design iteration cycles are compressed and/or overlapped.

While the simulation model developed in this research is based on the development process in use at an individual company in the automotive industry, the process itself and the resulting model structure developed here are general enough that the results and conclusions are widely applicable.

SUMMARY

Reducing the time to market of new products is critical to the success of the many companies facing increasing competition and more discerning and demanding customers. In an effort to bring

new products to the market more quickly, many companies have turned to concurrent engineering as a means of speeding up their new product development processes. Along with the potential for considerable reductions in development time, concurrent engineering brings with it significant challenges. In particular, new product development managers must effectively integrate the work done in parallel by project team members from different functional areas and those working on different components or subsystems that are inherently coupled. Design iteration cycles lie at the heart of the development process and the pressure to bring products to market more quickly encourages the compression of these learning cycles and, in the extreme, to overlap them. Doing so exacerbates the integration challenge and has the potential to create unnecessary, or phantom, work and reduce overall project performance. Understanding the impact of the frequency and timing of the integration build events, which set the pace of the learning cycles, on product development time, cost and quality will serve to inform companies and, in particular, project managers faced with increasing pressure to bring new products to the market quickly.

In the next chapter, I will provide an overview of concurrent engineering as it applies in the detailed design stage of new product development and describe the integration challenge it presents to project managers. I will explicate the concept of phantom work and how it can arise when design iteration cycles are compressed. Additionally, I will review the relevant literature from concurrent engineering, design iteration and system dynamics as it pertains to the present research.

In Chapter Three, I will provide an overview of the new product development process that forms the basis of modeling and analysis effort of this thesis. I include a brief background of the company where I conducted my field research and its product development organization. I provide an overview of the formal methodology the company uses to manage its new product development projects and provide some insight into the "real" process in place.

Chapter Four details the system dynamics simulation model I developed based on my field research. Chapter Five provides an analysis of the simulation model including a discussion of the model's behavior comparing it to historical data on product development projects from the company I researched and the model's sensitivity to parameter values. Chapter Five also includes an analysis of a series of policies designed to answer the research questions surrounding design iteration cycle timing. Chapter Six summarizes the research effort including its contributions as well as its limitations. It discusses the impact of these results on project management, new product development, and future system dynamics modeling efforts. Finally, it concludes with a discussion of possible directions for future research.

Chapter 2: Background and Literature Review

> ***"If we knew what it was we were doing, it would not be called research, would it?"***
>
> -- Albert Einstein, Physicist and Nobel Prize Winner (1879 – 1955)

BACKGROUND

(Fine 1998) characterizes the *clockspeed* of an industry as the rate at which it evolves particularly its products, processes and its organizations. The two primary drivers of the clockspeed of an industry are technology and competition. A technological innovation within an industry generally causes the rate of evolution of its products, processes, and/or organizations to increase. Similarly, an increase in competition in an industry can cause an increase in its clockspeed. Beginning in the 1990's, an increase in global competition combined with innovations in information and communications technologies has caused the rapid acceleration of the clockspeed of virtually every industry. How then have companies been able to increase the rate of evolution of their products and processes in order to keep up with their competitors in their respective industries?

There are a great number and variety of techniques and approaches in use by companies to accelerate their new product development efforts and reduce the time to market of their new products (Gold 1987; Millson, Raj et al. 1992; Droge, Jayaram et al. 2000; Langerak and Jan

Hultink 2005). A hierarchy of approaches was developed by (Millson, Raj et al. 1992) that groups product development acceleration techniques into one of five categories: (1) those that simplify new product development operations, (2) those that eliminate unnecessary new product development activities, (3) eliminating delays in the new product develop process, (4) speeding up new product development tasks, and (5) performing new product development tasks in parallel. One of the most commonly cited techniques for speeding up new product development is concurrent engineering which falls in this last category.

Concurrent Engineering

Also referred to as parallel processing, concurrent engineering involves the overlapping of development tasks and activities that might normally be done in a sequential (or "over the wall") fashion in an effort to reduce the overall development time. While some researchers emphasize the overlapping of design activities in describing and studying concurrent engineering, others place more emphasis on the integration of members from different functional areas into a multi-disciplinary team. (Blackburn, Hoedemaker et al. 1996) provide a nice framework of concurrent engineering which distinguishes between *activity concurrency*, which refers to the design tasks and activities that are performed in parallel by different people, and *information concurrency*, which refers to the integrated, team approach to development which emphasizes the involvement of different functional areas throughout the development process through information sharing.

The most common form of concurrent engineering discussed in the project management and new product development literature falls under the rubric of activity concurrency and involves the overlapping of development stages. For example, rather than waiting until the detailed engineering design of a new product is completed a new product development project might begin the design/redesign of the corresponding manufacturing process. There is a good bit of literature,

which will be discussed below, that looks at this form of concurrent engineering. However, an even more basic form of concurrent engineering occurs within a single development stage when development tasks are overlapped or executed in parallel. A common example is when design tasks within the detailed design stage are subdivided and carried out simultaneously by different people. This means that product components or subsystems are designed by different engineers at the same time.

Research has shown that the use of concurrent engineering, more specifically the overlapping of development activities, can lead to dramatic reductions in development lead time (Takeuchi and Nonaka 1986). However, in many cases these reductions in lead time come at the expense of increased risk and, more specifically, increased development costs. Overlapping design activities, as opposed to following a sequential approach to design, calls for the start of a downstream (or dependent) development task before the upstream (antecedent) task is completed. This forces the downstream task to begin with only partial information regarding the upstream development activities. This creates a situation in which the downstream development activity may be forced to rework some or all of its tasks as it gets more information or new information (i.e. a change) about the upstream activity.

The likelihood for rework and the related costs increase with the degree of overlap in development activities. Considering a fairly broad set of development stages, the engineering group might begin designing the product before the marketing research is complete, the voice of the customer is known, and the product concept is completed and/or approved. The manufacturing engineers might begin designing the manufacturing process before the detailed product design is complete. In the extreme, all three development areas (marketing, engineering, and manufacturing) might be working simultaneously greatly increasing the chances that a change or

new information in an upstream activity will necessarily cause a change or rework in a downstream activity. Imagine if a major shift in the customer requirements was discovered after a large part of the design had been completed and the related manufacturing process had been designed and tooled.

While concurrent engineering offers the potential for greatly reducing the development time of a new product, it is clear that the overlapping of development tasks does not come without risks. New product development managers must carefully consider which development tasks can be reasonably executed in parallel and the degree to which they should be overlapped understanding the potential for rework (design iterations) and additional cost. In fact, (AitSahlia, Johnson et al. 1995) show that given the uncertainty found in the product development environment there exist conditions under which the serial approach will always be less costly than the concurrent approach to engineering.

The potential for creating rework in the more basic "within-stage overlap" form of concurrent engineering is also very real and perhaps even more likely. Consider the common scenario where a new product under development consists of various subsystems and different functional groups within an engineering organization are responsible for the development of each subsystem. The interaction of these subsystems causes a set of natural dependencies among them. In the simple case, two subsystems might need to "fit" together – the handlebars need to fit onto the frame of a motorcycle. Or perhaps the "function" of one subsystem might depend on the function of another – the headlamps on a car depend on the power provided by the electrical system in order to function properly. The interactions among system components and subsystems can be quite complex. The corresponding dependencies - often reciprocal - create a rugged landscape of potential rework as changes in any one component or subsystem may necessitate changes in any

number of other components or subsystems. In much the same manner, (Smith 1992; Eppinger, Whitney et al. 1994) characterize the information dependency between development tasks as independent, dependent or coupled. Figure 1depicts each of these information dependencies. Independent tasks do not depend on each other for information. A dependent task, however, requires information from another task. As depicted in Figure 1, task B requires information from task A in order to be completed successfully. Coupled tasks, on the other hand, depend on each other for information. Both A and B require information from the other in order to be completed successfully.

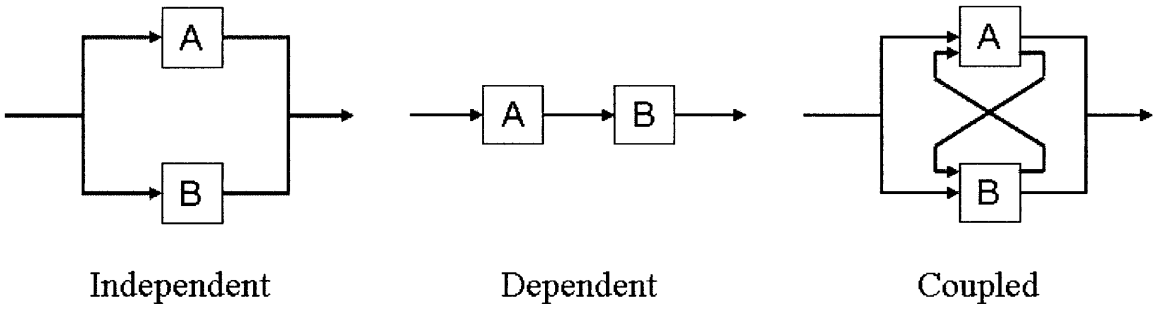


Figure 1 Information Dependency between Tasks

In practice, the development of the components and subsystems during the detailed design stage is most often done in parallel and by different engineers from different functional groups. In many cases these components and subsystems are tightly coupled – either by fit or function. For example, two coupled parts may need to fit together as part of the full system. Alternatively, one subsystem may be dependent on another as when the engine of a car relies on the fuel system (fuel pump) to provide it with the proper flow of fuel. Managing the parallel development of these coupled components and subsystems and ensuring that they are effectively integrated can prove to be challenging to say the least. This challenge can be even more daunting in a development

environment where there is a push to get the product to market as quickly as possible and design engineers are working on multiple projects at the same time. In this environment, there is often insufficient time for the day to day cross talk between engineers that is necessary to develop components in parallel greatly increasing the likelihood of integration problems between dependent or coupled subsystems.

The Integration Challenge

In the context of concurrent engineering, a common solution to the significant challenge of integrating the efforts of members of different functional groups (e.g. marketing, engineering, and manufacturing) is the creation of cross-functional project teams that include members from each of the functional areas of the organization that impact on the development project. However, as mentioned the degree of commitment from each of the team members to the project varies greatly as does the amount of authority and control of team activities afforded the project manager of the team. For example, (Clark and Wheelwright 1997) describe "lightweight" project teams where the team members physically reside in their functional areas but each functional organization designates a liaison person to represent it on a project coordinating committee to coordinate and integrate the different functions' activities. They contrast this project team with "heavyweight team" where a project manager is designated who has direct access to and responsibility for the work of all those involved in the project where the core members of the team from the different functional areas are dedicated to the effort and at times physically co-located.

Perhaps even more challenging, when conducting the detailed design of coupled components or subsystems in parallel, the project manager must work to coordinate the efforts of individual engineers so as to minimize the number of integration errors and issues that arise during the course of the development project. Unfortunately, faced with time pressure and competing commitments

engineers too often work in isolation and fail to coordinate their designs with those of the engineers working on other components or subsystems. Project managers often rely on formal and informal design reviews at the project team level as a means of forcing this integration between engineers. However, these reviews often amount to little more than a series of status reports given by each of the project team members with little true coordination between engineers aimed at integrating their design work accomplished. Additionally, preparing for these design reviews takes time and leaves engineers with even less time to do "real" engineering work.

Technology can assist in this integration effort with the use of computer-aided design tools that can combine CAD drawings of different components and subsystems and check the geometry and fitment of coupled components. However, while CAD tools are helpful for checking whether or not two components will fit together they only provide a static representation of the system and say little about how well two components or subsystems will function together.

Lastly, project managers often use prototype builds as a means of integrating the various components and subsystems by building the entire system and then testing it for both fitment and function. It is when the engineers are forced to bring their physical parts to the build event and integrate them with the other parts on the full system that they find where things don't fit and/or function together. These integration problems, now known and tracked at the project if not the organizational level, become very salient and force the engineers to work together, to coordinate and integrate their designs and resolve the issues.

The lack of day to day integration between engineers can lead the new product development project manager to increase the number of team level design reviews as a means of affecting integration. The lack of day to day integration is also likely to increase the chances of integration problems occurring. This increase in integration problems, whether found in a design review or

through a prototype build and subsequent testing, is likely to cause the project manager (or perhaps higher management) to call for even more design reviews and/or integration events further reducing the amount of time available to engineers to do "real" design work and the necessary integration which further increases the chances of design and integration problems in the future.

One can think of the detailed design phase of the development process as a series of design-build-test cycles. At the core of the design-build-test cycle is the prototyping of designs and subsequent testing. It is from this building and testing of designs that most of the learning within a product development project takes place. As Wheelwright and Clark point out, increasing the rate and amount of learning that takes place in each of the design-build-test cycles and then effectively sequencing and linking these cycles together "permits an organization to shorten the duration and number of cycles needed in order to develop high quality products and manufacturing processes and get to market significantly faster than competitors" (Wheelwright and Clark 1992).

Given the discussion above, the instinctual response of the project manager trying to bring about effective integration of subsystems in his or her new product development project might be to increase the number and frequency of system integration events (e.g. prototype build) and the associated design-build-test cycles of learning. However, these build events and learning cycles are not without cost. First and foremost, there are direct costs to the organization. These include the cost of prototyping parts (whether built in-house or purchased through a supplier), the cost to shut down and reconfigure the manufacturing and production lines if used for building and assembling the full system prototype, and the cost of the use of test equipment, facilities and personnel.

Less recognized but perhaps more important are the indirect costs of executing these design-build-test learning cycles. There is significant time and effort required of the design engineers to

prepare for and execute a build event and plan for and monitor the subsequent testing. Again, this time and effort reduces the amount of time available for the engineer to do "real" engineering work. It reduces the amount of time engineers have available to analyze the results of the tests, to conduct fault analysis (DFMA), to make necessary design changes, and lastly to coordinate with the engineers working on coupled components or subsystems to integrate their designs. It reduces the time available to do this "real" engineering work not only on the project at hand but on other development projects to which he or she may be assigned as well. Additionally, in a multi-project environment where test resources are limited, executing more frequent learning cycles for one development project reduces the availability of resources (e.g. test facilities) for other projects.

In addition, testing takes time. In the automotive industry it may take months to accumulate the test miles necessary to properly evaluate a design. However, as the test miles accumulate and the number of issues to be resolved increases, so too does the pressure to execute another design-build-test cycle before the completion of the current testing. This is especially true given an impending product launch date and the general pressure to bring products to the market as quickly as possible. A decision to execute another build event before the completion of the previous learning cycle leads to overlapping design-build-test cycles and thus having multiple design prototype versions undergoing testing and analysis at the same time. This can prove problematic as design engineers "lose track of the project status, which problems have been solved and which should be given top priority" (Wheelwright and Clark 1992). However, faced with the pressure to reduce the time to market and an accumulation of identified issues to resolve from previous design-build-test cycles, companies may opt to overlap these cycles in an attempt to speed up learning.

In response to a growing trend of late discovery of problems and the accumulation of issues, the new product development organization that I studied found its projects were executing an

increasing number of vehicle build events in order to discover and resolve issues in time to launch its products with confidence in their quality. It was in response to this phenomenon that the basic question of "How many builds?" was asked.

A perverse outcome of the managerial decision to increase the number of build events and/or reduce the amount of time between build events to the point where the design-build-test cycles overlap is that it can be a self-defeating proposition which ultimately can slow down the learning rate and lead to poor project performance. In particular, this seemingly logical approach to finding and resolving integration failures and issues can lead to products that have even more failures / issues to resolve at the time of product launch than they would otherwise have had. Understanding the structure that produces this type of unexpected behavior is greatly simplified using the causal loop diagram in Figure 2.

In a causal loop diagram, a causal relationship between variables is represented by an arrow where a change in the variable at the tail of the arrow causes a change in the same (+) direction or in the opposite (-) direction in the variable at the head of the arrow given all else remains constant. Additionally, when a series of variables are connected by causal arrows forming a complete loop, the loop is said to be positive, or reinforcing, when a change in a given variable can be shown to feedback on itself (through the intermediate variables in the loop) and cause a change in the same direction. A negative or balancing loop exists when a change in a given variable feeds back on itself to cause a change in the opposite direction as the original change. These feedback loops are represented in a causal loop diagram by a circular arrow with either an "R" (reinforcing) or a "B" (balancing) inside of them and are usually named appropriately. These circular loops are depicted in either a clockwise or counter-clockwise direction in order to indicate the pathway of the feedback loop it represents (Sterman 2000).

In order to understand the tendency for projects to execute an increasing number of builds and yet still have a high number of errors and unresolved issues at launch let us consider the causal loop diagram in Figure 2.

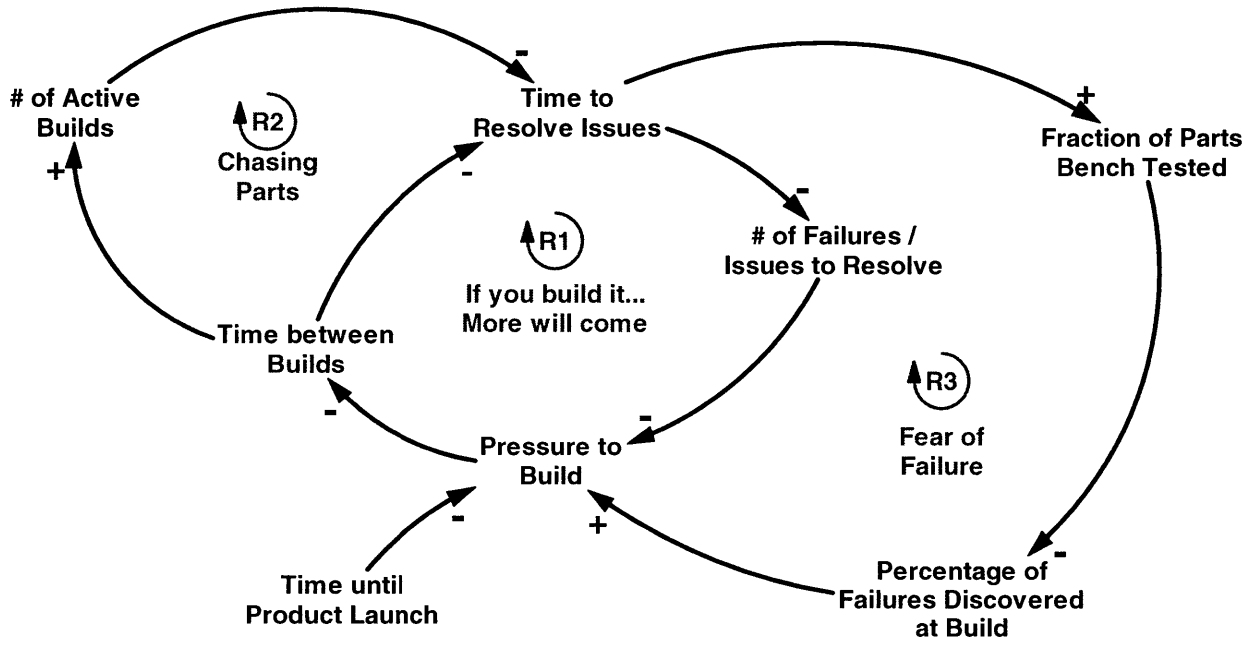


Figure 2 Builds beget Builds

To begin the discussion of the feedback structure, let us agree to the premise that build events are necessary in new product development. The building and testing of the full system or product (in this case, a vehicle) are perhaps the best means for validating and verifying that both the product and its associated manufacturing and assembly processes are able to meet their requirements. They are a primary means of discovering errors; indeed, they may be the only means of discovering certain types of integration errors. Additionally, successfully building and testing the full product generates overall confidence that the product will be ready for its launch. Absent of a successful build, as the time until product launch decreases the pressure to build (and test) increases. This causal relationship is indicated in Figure 2 by the arrow between the variables

Time until Product Launch and *Pressure to Build.* You'll note the minus sign at the head of the arrow indicating that a change in the first variable causes a change in the second variable in the opposite direction as outlined above.

The feedback structure in Figure 2 includes three feedback loops (R1, R2, and R3) all of which are reinforcing loops. The primary reinforcing feedback loop (R1), named *Builds beget Builds*, lies at the heart of the explanation for the tendency of new product development projects to experience and increasing number of builds. As mentioned, as the *Time until Product Launch* decreases the *Pressure to Build* increases when the product has errors and issues to be resolved. The increased *Pressure to Build* causes a decrease in the *Time between Builds* as the project manager chooses to move up the next scheduled build event or perhaps even add an additional build before the next scheduled build event. By decreasing the *Time between Builds*, design engineers have less *Time to Resolve Issues.* This means that they have less time to wait for the complete test results from the previous build event, less time to analyze these results, and less time to make the appropriate design changes. Given the decrease in the *Time to Resolve Issues*, one can expect an increase in the *Number of Failures and Issues to Resolve.* It is important to note here that a decrease in the *Time to Resolve Issues* does not necessarily cause an overall increase in the number of *Number of Failures and Issues to Resolve.* Indeed, we expect that the design engineers are using their time, however limited, to fix known integration errors and resolve identified issues.[2] Rather, a decrease in the time available to resolve issues would cause the number of failures and issues to resolve to be higher than *it would otherwise have been* if the time to resolve issues had not decreased. Lastly, a higher *Number of Failures and Issues to Resolve* causes an increase in the *Pressure to Build* thus completing the feedback loop. Retracing the loop, one sees that an initial increase in the *Pressure*

[2] One could make the argument that the decreased time to resolve issues will cause the engineers to cut corners and make mistakes that will create new errors. This possibility is, in fact, included in the model developed in this research.

to Build feeds back on itself to cause a further increase in pressure and a further reduction of the *Time between Builds.*

The second feedback loop (R2) is characterized by an overlapping of the design-build-test cycles; in particular, the overlapping of build and test events. As the *Time between Builds* decreases it causes, in the extreme, the *Number of Active Builds* to increase. This reflects the situation where a subsequent build is scheduled and executed before the previous build and testing is completed – not an uncommon practice in the company I studied. As the *Number of Active Builds* increases the *Time to Resolve Issues d*ecreases as engineers are forced to prototype and track the status of parts on multiple versions of the full vehicle and monitor, analyze and react to the test results of multiple vehicles. This decreased *Time to Resolve Issues* again causes an increase in the *Number of Failures and Issues to Resolve* which in turn increases the *Pressure to Build* resulting in a decrease in the *Time between Builds* further increasing the *Number of Active Builds.* This feedback loop (R2) entitled "Chasing Parts" is also reinforcing in nature.

The last feedback loop included in this simplified causal loop diagram (R3) reflects the designers' tendency to avoid bench testing when they are faced with schedule pressure. As the *Time to Resolve Issues* decreases design engineers tend to decrease the fraction of parts that they send for bench testing. This occurs for a number of reasons. Primarily, when given insufficient time to resolve issues designers will avoid bench testing their individual parts and wait until the next build event. Designers don't want to show up to the build event with a part that is known to have an error. If they don't feel they will have time to fix a failure or resolve an issue discovered in bench testing before the next build event, many designers will not "risk" bench testing their part. Additionally, most designers feel that full system testing is more realistic than bench testing and provides the best information which helps them justify avoiding bench testing when faced with

schedule pressure. So, a decrease in the *Time to Resolve Issues* causes a decrease in the *Fraction of Parts Bench Tested* which, in turn, results in an increase in the *Percentage of Errors Discovered at Builds* as opposed to in bench testing. This further reinforces in the mind of engineers and, more importantly, in the mind of the project manager the importance of build events thus increasing the *Pressure to Build.* In turn, this causes a further reduction in the *Time between Builds* and the *Time to Resolve Issues* thus completing the feedback loop. This reinforcing feedback loop is titled "Fear of Failure".

It is easy to see how these three reinforcing feedback loops can combine to cause the pressure to have a build event to grow causing a project manager to decrease the time between build events which only results in decreasing project performance as the number of failures and issues to resolve remains high as the product launch date approaches. However, the story here only considers the feedback loops acting in a vicious cycle. Given the right conditions, the reinforcing feedback loops have the potential to act as virtuous cycles. For example, an increase in the *Time between Builds* would cause an increase in the *Time to Resolve Issues* which would decrease the *Number of Errors and Issues to Resolve.* This in turn would decrease the *Pressure to Build* and allow the project manager to further increase the *Time between Builds.* In this case we would expect project performance to improve.

Of course, there are limits to these feedback loops. For example, the *Time between Builds* can only be so long given a fixed product launch date and the recognized need to conduct a build and testing in order to verify and validate the product and process. Conversely, the time between builds must reach some minimum value. In the limit, the time between builds would be infinitesimally small and would result in a continuous building of vehicles. Obviously, this is unrealistic. It is interesting to note that in the software industry, software developers will execute

daily builds where they integrate the various software modules. The minimum time between builds varies from across industries. For this industry it is reasonable to assume that conducting a build any less than a month after a previous build doesn't provide enough time to accomplish any real learning based on testing and to make any improvements. Even here, this depends on the scope of the development project and its inherent complexity. It may be reasonable to implement a small change to a mature design in less than a month. But in this case, the information gained after making such a change may not be worth the expense of an additional build. A detailed analysis of the effect of build timing on project performance is included in Chapter Five.

Phantom Work

In addition to the increased burden on individual engineers and the additional direct costs to the organization, reducing the time between build events and overlapping design-build-test cycles can result in different subsystems coming to a given build event at varying levels of readiness. This means that some subsystems are ready for the build while others are not because they didn't have enough time to finish their testing, analysis and necessary design work. However, those subsystems that aren't ready for this build event are ready for the next build while some of those that were ready for the initial build may not be ready for the next one. This was a common occurrence in the product development organization that I studied which indicated that learning was happening at different rates for different components and subsystems and learning at the product level was disjointed at best. This also meant that learning opportunities were either being missed or ignored. The result was a vicious cycle where coupled components and subsystems would induce changes in each other from build to build. In the words of one engineer, the learning within particular subsystems was "leapfrogging" each other.

This out of phase learning between components and subsystems is a result of the amount of time that elapses between build events and can lead to the perverse situation where unnecessary or *phantom* work is created. This phantom work arises in situations where different subsystems that are developed in parallel iterate through the design-build-test cycle at different speeds. The amount of phantom work generated is a function of the individual iteration speeds of the various subsystems, the degree of speed imbalance between them, and the timing of the iteration cycles. There are two sources of this phantom work. The first source is unnecessary rework that is created when some components are ready for a build event and others are not. The second source of phantom work is the redesign of components after their design has been released for prototyping and before the next set of test results is available.

To understand the sources of unnecessary work, consider the simple case of two subsystems that are being designed in parallel by two different engineers (or groups of engineers). The two subsystems are both dependent on each other (for either fit or function) and therefore a change in one subsystem can, with some probability, induce a necessary redesign of the other. Similarly, an integration problem between the two subsystems will cause one or both of the subsystems to require a redesign. Further, assume that the time it takes for one of the subsystems to complete a design-build-test cycle is significantly longer than the other. That means the time it takes to design (or redesign) the subsystem, build a prototype, test it and then receive and analyze test results is significantly greater for one of the subsystems than the other. For example, the time it takes to execute a design-build-test cycle for a motorcycle frame is significantly longer than it is for the motorcycle headlamp. Designing (or redesigning) the frame is considerably more complex and time consuming than designing the headlamp. The fabrication of a headlamp from paper design to physical part can be accomplished in a few days where it may take a month or more to fabricate a

motorcycle frame. Adequately testing the motorcycle frame requires the accumulation of thousands of road test miles which takes considerable time. Alternatively, the headlamp's significant testing might simply require assembling the vehicle to ensure the headlamp fits properly and then cycling the on/off switch a certain number of times to check its function both of which can be done much quicker than accumulating the miles necessary to effectively test the frame.

Given the case that it takes one subsystem significantly longer to iterate through the design-build-test cycle, consider the case where a build event occurs and an integration error between the two subsystems is discovered. For example, the headlamp does not fit properly into the bracket on the motorcycle frame. Once the integration error is discovered, the respective design engineers for the two components must come together to coordinate how the error should be resolved. The result of the coordination may be that one or both of the subsystems needs to be redesigned. Assume for the moment that both need to be redesigned prior to the next build event.

Based on the amount of time available until the next scheduled build event one of three scenarios is possible. In the first scenario, there is sufficient time until the next build for both the fast and the slow iterating subsystems to complete testing and analysis, redesign and prototype their components. This is obviously the best case scenario but it is not always possible given the pressure to compress learning cycles and decrease the time to market. In the second scenario, neither subsystem is able to complete its testing, analysis, redesign, and/or prototyping prior to the next build.[3] This is obviously the worst scenario and greatly decreases the value of the next build event. In the third scenario, there is sufficient time for the fast iterating component to complete its testing and analysis, redesign and prototyping but there is not enough time for the slower iterating

[3] In the company where I conducted my research, when there is insufficient time between build events the engineers will always ensure that they have a prototype available for the next build event. This means that given the time it takes to prototype a particular component, if necessary, they will prototype a partially redesigned component

component to get its work done. It is this third scenario that I have found to be prevalent in my research and the one in which unnecessary, or phantom, work is encountered.

Again, consider the example of the integration error (fitment problem) between the motorcycle frame and the headlamp where coordination between the design engineers determined that both components would need to be redesigned. To understand the first source of unnecessary or phantom work, let's apply the third scenario where the time until the next build event is insufficient for the slower iterating component. For the fast iterating component (in this case, the motorcycle headlamp), the design engineer gets the balance of his testing completed and is able to analyze the results, redesign and prototype the component in plenty of time for the next scheduled build. Meanwhile the slower iterating component (the motorcycle frame) might not be as "ready" for the next build event. The design engineer may still be waiting for the completion of testing – it takes time to accumulate miles – or may not yet be done analyzing the test results. Even if the design engineer of the slow iterating component has all the test results and has completed fault analysis on any errors found, the engineer might not be done with the redesign of the longer iterating component. In this case, in order to have a prototype frame at the next build event, the engineer of the slow iterating component will be forced send an incomplete design for fabrication, perhaps with a considerably longer lead time than the fast iterating component.

Given this scenario consider what happens at the next build event. The redesigned headlamp and the partially redesigned frame will be assembled as part of the full vehicle and have another integration error – the headlamp still does not fit into the bracket on the frame. Remember, we are assuming that the longer iterating component was not able to complete its redesign prior to the build and thus was unable to make the design changes necessary to fix its "share" of the integration error. This integration error again requires the two engineers to coordinate with each other to

consider the fitment problem and determine how to resolve it. The decision of how to resolve the problem in this scenario is not an easy one. One might think that the fitment problem is the "same" as the one from the last build but clearly that is not the case as the fast iterating component has completed a redesign and the slow iterating component may have at least partially redesigned its component. The components are in fact new and thus the error is indeed a "new" one.

The source of the new integration error is not easily discernible to the engineers. The fitment problem can be the result of the design changes that either has made or simply from the fact that the slower iterating component did not fully complete its redesign prior to the build. This makes resolving the new integration error problematic. There are two basic options. The first is to assume that any redesign that was done between the builds was done perfectly and conclude that the integration error is the result of the incomplete redesign of the slower iterating component. This assumption depends on both engineers knowing (which means the engineer of the slow iterating component must admit) that the slow iterating component's redesign is incomplete. This assumption might lead the engineers to "ignore" the new integration error and work off of the solution agreed upon from the first build. That would mean that the fast iterating component would stand pat with his or her design - assuming it has no other errors (e.g. a component error) to fix - while the slow iterating component would complete its redesign as agreed to after the first build, again assuming it has no other errors to fix or new information that would alter the agreed upon redesign. The second option is to treat the integration error between the slow and fast iterating component as a completely new one and, as with the original integration error, decide if one or both of the components needs to be redesigned.

It is my contention that the second option is the logical choice for the design engineers. First, it is not entirely obvious that the engineer of the slow iterating component will recognize that his

redesign is incomplete. The design is sent for prototyping based on the best information (i.e. test results) available to the engineer at the time. And, even if the engineer did know that the design was incomplete it is not likely that they would make that known to others. Secondly, it would be hard for the engineers to ignore the integration error in its present form. In all likelihood, the error would generate some type of test incident report that would raise the visibility of the problem and require some action to "answer the mail" to show the organization that the issue has been resolved. Lastly, as previously mentioned, the integration error found in the second build is truly a new error in that one or more of the interacting components has been changed since the original build and the original error was discovered. Based on these reasons, it seems likely that the engineers would indeed treat the integration error as a new one and coordinate for the redesign of one or both of the components in order to fix the error.

If it is decided that both components require redesign based on the coordination of this "new" integration error, I contend that the redesign of the fast iterating component represents potentially unnecessary rework and is the first source of what I call phantom work. This rework is unnecessary insofar as it would not have been required if there had been sufficient time between the first and second build event to allow the slow iterating component to complete its testing, analysis, and redesign. This, of course, assumes that both of the components are redesigned correctly thus eliminating the integration error prior to the second build.

As mentioned, the second source of unnecessary or phantom work is the redesign of components after their design has been released for prototyping and before the next set of test results is available. This source of unnecessary work is directly linked to the scenario outlined above where the slower iterating component has insufficient time between build events to complete its testing, analysis and redesign and must prototype a partially redesigned component for the next

build. Again, we are considering the case where an integration error occurred (the fit between the frame and the headlamp) and it was agreed between the engineers that both components would need to be redesigned in order to resolve it.

As mentioned, the design engineer of a particular component will ensure that a prototype of the component is available at the next build even if he or she knows that the component's design is incomplete (i.e. requires additional testing, analysis and/or redesign). This is precisely what happens with the slower iterating component in the scenario outlined above when there is insufficient time between builds to complete the design iteration cycle. The design engineer may be waiting on more testing or to complete fault analysis or redesign but because of the lead times required to prototype the component, they are forced to release the design before it is completed. Knowingly bringing an incomplete (i.e. partially redesigned) part to a build is certainly a problem but it creates another dilemma for the design engineer. What should he or she do after releasing the design for prototyping? The first option is to cease working on the component until the build event is executed and the new set of test results start to come in. Taken to the extreme, this would mean that the current round of testing (from the first build) would be stopped immediately and any incomplete analysis and redesign would be stopped as well. The argument to cease work is that releasing a new design for the next prototype build renders the current prototype obsolete. While there is some truth to this argument, it seems unlikely that a design engineer would choose to cease all development work on a component until the next build event and round of testing begins. At a minimum, it doesn't seem prudent to stop the ongoing testing of the first build prototype given that many of the problems (e.g. part fatigue) cannot be detected until late in the testing regime. It also doesn't seem prudent to not complete the fault analysis of known errors found from testing the first build prototype. It is likely that most of these errors will be found again in the next round of

testing (i.e. testing the second build prototype). After all, if the fault analysis was incomplete at the time of design release for the next build it is unlikely that the error will be resolved in the next prototype. At a minimum, there is much to be learned in conducting the fault analysis on the "obsolete" component that can be used in resolving similar problems in the future.

That leaves the question of what to do about the actual redesigning of a component in the time between releasing a new design for prototyping and the next build. Remember that the slower iterating components might need to release their design well in advance of the build event in order to have a sufficient number of prototypes on hand[4]. Given a backlog of known errors and related redesign work, it seems likely that the design engineer will choose to continue to redesign the component even after he or she has released its design for the next build. After all, the design engineer knows that the part they are prototyping for the next build is incomplete and they will no doubt want to avoid releasing another incomplete design for the subsequent build. Thus they will choose to redesign. It is this choice to continue redesigning that is the second source of unnecessary or phantom work.

To better understand why this redesigning effort between design release and the build event and testing can lead to unnecessary work, consider the integration error (fitment problem) between the motorcycle frame and the headlamp and what will happen at the next build event. As was previously discussed, because the slow iterating component (the frame) released an incomplete design for prototyping an integration error will necessarily exist in the next build between it and the fast iterating component (the headlamp). As discussed, this integration error most likely will be treated as a new integration error - indeed, it should be - requiring coordination and subsequent redesign. But wait. Because we have chosen to continue to redesign the component between

[4] In the company that I studied, a particular component had a lead time for prototyping as long as 3 months and had to be sent out to an external supplier to be prototyped while other components could be prototyped in-house in a single day.

design release and the build event, the design of the slow iterating component has already changed. Shouldn't we ignore this new error and wait until the next time we build a prototype incorporating these changes? Consider the options. We can ignore the "new" integration error and wait until the next build. Assuming that we don't make any other changes to either the frame or the headlamp that impact the fitment problem, this means that at the third build we will have a frame and a headlamp that are designed based on the test results and subsequent learning from the first build as it pertains to the fitment problem. The second option is to consider the integration problem from the second build as a "new" problem and disregard the changes that were made to the frame design (as they pertain to the fitment problem) after its design was released for prototyping for the second build. This would mean that for the third build we will have a frame and a headlamp that are designed based on test results and subsequent learning from the second build which incorporated learning from the first build. The choice seems clear. As was determined earlier for other reasons, the design engineers should not ignore this "new" integration error but should use it as the basis for coordinating the necessary redesign prior to the next build.

Given that the "new" integration error will be used as the basis for any changes prior to the next build, any redesign that was done to the slow iterating component (as it pertains to the integration error) between releasing the last design and discovery of the "new" integration error will be ignored. The effort expended in this redesign can be considered unnecessary and thus is a source of phantom work[5].

Phantom work presents a "real" problem for companies engaging in new product development. Clearly, doing unnecessary work has a direct cost to the organization in that engineers' time is

[5] One could argue that the learning that might have taken place in the process of redesigning the slow iterating component is valuable and perhaps could be incorporated into future redesigns. I will not argue against that but will claim that at least to some degree the effort expended in this redesign work was unnecessary. Additionally, taking a more global view, it is not unreasonable to believe that the effort expended in this redesign might have been better spent elsewhere (e.g. on another new product development project).

being wasted and other resources (e.g. design and test equipment) may be unnecessarily consumed. With improved timing of design iteration cycles a project manager can reduce or eliminate phantom work. In fact, it might be possible to reduce the number of learning cycles required to achieve a desired product quality level. This reduction is particularly important given the significant costs, direct and indirect, associated with build events. In addition, companies should consider how the efforts and resources expended on phantom work might be better utilized – perhaps applied to another new product development project.

RELEVANT LITERATURE

Overlapping Development Stages

There is a fair amount of research and literature that looks at the amount of overlapping between development stages and the subsequent trade-off between development time and cost. For example, (Krishnan, Eppinger et al. 1997) develop a methodology to overlap development activities based on the information exchanged between them and examine the effect of overlapping on performance. They develop a framework for considering the amount of overlap based on the speed of evolution of the upstream activity and the sensitivity of the downstream activity to changes in information from the upstream activity. A limitation of their model is the assumption that the dependency between development activities is one way. That is, changes in information in the upstream activity can cause design iteration in the downstream activity but the reverse is not true. In related work, (Loch and Terwiesch 1998) develop an analytical model of concurrent engineering which combines the decision to overlap development tasks with the communication pattern used before and during the period of overlap. Their model trades off the gains from overlapping with the rework that is generated by using preliminary (or uncertain) information in

downstream tasks. In their model they use communication, recognizing its associated costs (i.e. time), as a means of reducing the amount of downstream rework Like (Krishnan, Eppinger et al. 1997), in their model Loch and Terwiesch assume a strictly sequential dependence of development tasks and they focus on minimizing the overall development time. The task times in their model are deterministic and the only source of uncertainty in their model is the probabilistic generation of changes in the upstream activity which induce changes in the overlapped downstream activity.

Unlike the previous research, (Ha and Porteous 1995) assume a complete overlapping of development tasks is possible and develop a model that looks at the optimal number and timing of design reviews during which information is passed from the upstream activity (design) to the downstream activity (manufacturing) allowing the downstream activity to work concurrently with the upstream activity. Additionally, during these design reviews all design flaws in the upstream activity needing rework are discovered. While their model considers the time and effort it takes to prepare for and conduct a design review, it too assumes a one-way dependency between tasks. In this case, during the review the downstream activity (manufacturing) is able to identify necessary changes in the design work (e.g. a design that cannot be easily manufactured might be required to change). However, their model assumes that once a design flaw is discovered the design is revised fixing all errors and the design work completed to that point will never be flawed again. Additionally, the model assumes that the downstream work is always done correctly the first time and never requires rework.

Design Iteration

Rework, or design iteration, is recognized as a characteristic feature of development and design processes and has been the subject of significant research within the management science community. (Smith and Eppinger 1997b) develop a model based on a design structure matrix

(DSM) that considers purely sequential iteration where coupled design tasks are executed one at a time. In essence the design responsibility is passed or hit back and forth like a ping-pong ball. They model the iteration process by developing a Markov Chain with fixed task duration times and fixed rework (or iteration) probabilities represented by the transition probabilities. The model is then used to compute an expected duration of the overall development process and to suggest how design tasks might be initially ordered. In related work, (Smith and Eppinger 1997a) develop a model of parallel iteration where a set of coupled development tasks are executed simultaneously. Building on the DSM structure, they develop a work transformation matrix that captures the duration times of each of the tasks and the probability that a given task will create rework for other tasks. Interpreting the eigenstructure of the WTM allows them to determine the rate and nature of the convergence of the iterative design process and identify what design modes drive the iteration and convergence. While the probability of rework for each of the tasks is assumed to remain constant the duration of the iterative cycles is assumed to decrease linearly.

(Eppinger, Nukula et al. 1997) develop a more general model of design iteration using signal flow graphs where the branches represent the development tasks and the transmissions include both the task duration and the probability of executing the task. In this manner, the signal flow graph can be used to represent both parallel and sequential execution of development tasks. Analysis of the graph can yield expected duration (lead time) for the design process to converge and similar to (Smith and Eppinger 1997a) identify the group or groups of coupled design tasks that control the convergence of the design. As in the previous models, the task durations are deterministic and the probability of rework for a given task remains constant.

System Dynamics

Within the system dynamics literature there are a number of pieces of research on project management and new product development that relate to my work. One of the early uses of a system dynamics approach to modeling project performance was done by (Cooper 1980) to identify the causes of cost and schedule overruns on two multibillion dollar shipbuilding programs. This model develops the modeling structure of project management that captures the accomplishment and monitoring of work, the discovery and accomplishment of rework, and the impacts of labor, productivity, and technical requirements on each. The purpose of the model was to show how changes to the project scope and requirements requested by the customer induced significant cost and schedule overruns through rework.

(Ford 1995) developed a model of a multi-phase development project to investigate the impact of coordination policies on project performance. In his model, Ford incorporates a number of standard project structures into a single integrated model including the development process, work activities, labor resources, project scope and project targets. He models 4 development phases: product definition, design, prototype testing, and quality control in order to test the impact of various levels of coordination between phases on the project's cost, quality, and development time. He found that a policy of increasing the priority of coordination, while improving quality, led to the unexpected outcome of increased cycle time and project cost.

As previously mentioned, (Ford and Sterman 1998) developed a system dynamics model to study the management of new product development processes. This model builds on the basic work accomplishment structure from (Cooper 1980) and leverages Ford's earlier work (Ford 1995) incorporating distinct development activities or phases, iteration within and among those phases and concurrence relationships within and between development phases. The concurrence

relationships are treated as constraints limiting the amount of work that can be accomplished within a given phase based on the work already accomplished within that phase and by limiting the amount of work that can be accomplished in a downstream phase based on the amount of work accomplished in a coupled upstream phase(s).

(Repenning 2001; Repenning, Goncalves et al. 2001) model the multi-project development environment to investigate the dynamics of sharing resources among development projects. They develop a model in which two projects are competing for the same resources with one project being in its early development stage and the other project in its late development stage. They show how pulling resources from the early stage project to help fix problems in the late stage project can be self-defeating and lead to persistent firefighting. They show that a tipping point exists beyond which firefighting will spread through multiple projects and the development organization will not recover without a significant change in resources and/or project scope.

Building on their previous model, (Ford and Sterman 2003a) develop a system dynamics model of a concurrent development project to demonstrate how the overlapping of activities can speed a development project but at the cost of decreased quality causing unplanned iterations as the discovery of rework is delayed. In the extreme, concurrency can result in releasing a product to the customer with lower quality and undiscovered errors that may require a recall.

(Black 2002) examines the relationship between the discovery of rework and the frequency, timing and type of design reviews in the same engineering organization that I am researching. Her model distinguishes between the different type of design reviews that are held (from a review of paper designs to a full vehicle build and test) and suggests that the ability to discover errors in work is limited to those reviews where members from both design and manufacturing are involved and where the objects of review are most salient to all participants. In this regard, her findings

suggest that holding more design reviews does not necessarily improve project performance but rather holding more salient reviews (i.e. vehicle builds) earlier in the process can provide significant improvement. Her model is greatly simplified in that it considers a set of independent tasks to be accomplished within a single stage of product development. It does not allow for coupling of design tasks and the potential for associated integration errors. Nor does the model allow for overlapped build events and subsequent testing. Her model also assumes that resources are always adequate and design work and testing is limited only by the amount of work available.

Chapter 3: New Product Development Process

> ***"It's an emotional process carried out by emotional people to create an emotional product for an emotional customer."***
>
> -- Senior Design Engineer, describing the new product development process used at his company

The research questions outlined in the first chapter were developed as a result of field research I conducted within the engineering organization responsible for new product development in a midsized company in the automotive industry. Because of the sensitivity of some of the information reported here, the identity of the research company is not made known in this report and will henceforth be referred to as the Company. My involvement with the Company spanned a three year period and included a significant number of site visits to the engineering organization as well as to one of its manufacturing and assembly plants.

The research began with a review of primary source documents from within the organization. These documents included internal reports documenting the performance of individual new product development projects as well as annual assessments summarizing overall new product development performance within the Company. I was given access to data from the Company's internal reporting system for development projects. I also reviewed academic research previously conducted within this particular engineering organization.

In addition to examining these documents, I conducted extensive and repeated interviews of members inside the engineering division responsible for new product development of the Company. The interviewees included various members of upper level management within the engineering organization, platform and system group managers who were senior engineers, former and current new product development project leaders, test engineers, individual design engineers, as well as each of the members of the product development office (PDO) which is responsible for the Company's product development process and providing oversight to new product development projects.

Early in my research effort, I coordinated and helped facilitate a meeting of a large group of experienced engineers and project leaders during which they were introduced to a system dynamics simulation of a development project. Using the simulation, the participants were able to assume the role of project leaders and try to manage a project. During this engagement, I was able to solicit feedback from the participants on the key features of development projects and the role of project leaders within the Company that needed to be captured in the structure of such a simulation model.

Over the course of my involvement with the Company, I spent considerable time on-site attending and observing design review meetings for various projects and conducting follow-up interviews with participants. I was also allowed to sit in on and observe a detailed internal review of the new product development projects for a given model year certifying their readiness to launch. This review was conducted by a group of senior leaders including the Vice President for Engineering, the Vice President for Marketing, the Vice President for Manufacturing, the Vice President for Quality, Reliability & Technical Services, and others. This leadership group is

responsible for overseeing the quality and reliability of the Company's new products throughout their development.

During the course of my research effort I maintained contact with a few key "insiders" from the Company who were able to provide me with regular updates on activities and changes within the engineering organization. In particular, I had regular phone conversations with a member of the product development office who was also serving as a member of the Core Team on a unique development project that was attempting to develop a major new product using a development timeline that was significantly shorter than the normal timeline under which a project of this scope would be developed. Through these conversations, he was able to provide me with an insider's view in "real time" of an ongoing development project that was experiencing many of the behaviors I was attempting to capture in my model of new product development projects.

During the course of my research effort I had three different primary points of contacts with the Company as a result of personnel changes within in the organization. Thankfully, each of the engineers designated to serve as my point of contact had a working knowledge of the system dynamics approach to modeling and simulation as well as a keen interest in my research. Having three different points of contact during the course of my research allowed me to gain different and insightful perspectives on the problem I was studying. This provided a good means of "triangulating" my view of the problem and the system dynamics simulation model I developed to study it.

Lastly, I was fortunate in that my research advisor had a past and ongoing research relationship with the Company. As a result, he was very familiar with the Company, its engineering organization, and its internal processes including its new product development process. As such, my advisor was able to provide me with invaluable insight during the course of my research effort.

He also maintained relationships with key members of the Company and was able to assist me in gaining access to the Company and its key leaders.

THE COMPANY

The Company where I conducted my research is a mid-sized firm in the automotive industry. The Company has been in business for over 100 years and throughout most of its history the Company has maintained a strong position in its market relative to its competitors. The Company arguably has the most recognized brand name in its industry. The Company has an extremely loyal customer base that has aided in seeing the Company through a series of challenges it has faced throughout its history including problems with quality and strong competition from foreign entrants into the marketplace.

After surviving near bankruptcy in the 1970's, the Company overcame significant challenges with respect to the quality of its vehicles in the 1980's by undertaking an aggressive approach to improving quality in its manufacturing processes. The Company's embrace of the quality movement met with great success and despite strong competition from its competitors the Company entered an extended period of sustained growth and profitability. Over the 15-year period between 1985 and 2001, the Company's volume of sales grew from roughly 35,000 units to 235,000 units – a sustained annual growth rate of 13.5% (Internal Company Document, 2003).

One of the new challenges the Company faced during this period of growth was meeting the increasing demands of a growing market. One of the major selling points for the Company is the uniqueness of its products. Indeed, this is the source of much of its customer loyalty as having a unique vehicle is a point of pride for the owner. As the market grew so too did the demands for new and unique products. Simply selling more of the same products or a few new products in

larger quantities was not acceptable. In this industry, this means that the Company faced pressure to introduce new products to the market every year in the form of model year updates, adding new model variants to existing platforms, overhauling the existing vehicle lines, and the introduction of completely new vehicle platforms.

The challenge of meeting a growing demand for new products is not met simply by shear numbers of new products introduced to the market. More importantly, these new products must meet the needs and desires of the customer. In the case of the client Company, the customer base is extremely discerning in its demands. Fortunately, the Company maintains a strong connection to its customer base and is "dialed in" to the unique demands of the market. The Company invests significant time, money and effort staying connected to the customer base, understanding the "voice of the customer" and translating it into a set of product requirements that define what the potential users of the product want. In addition to participating in formal trade shows, company representatives from the highest levels down regularly attend informal gatherings of product owners. A majority of the company's employees are also owners of one or more of its products. It is not uncommon for senior Company leaders to return from such informal gatherings with an idea or a concept for a new product that resulted from a direct interaction with a member of the customer base. Because of its strong connection to its customers, the Company has been extremely successful in matching its new products to the needs and desires of the market. The new products the Company has developed have, for the most part, been universally accepted in the market place.

Despite its many successes, the Company has a recognized history of struggling to bring new product development projects in on time and within budget while meeting scope and quality objectives. It is not uncommon for new product development projects in this Company to have a large number of unresolved issues late in the development lifecycle that threaten the viability of a

planned launch. In many cases, the late discovery of technical issues and/or the large number of unresolved issues for a given new product development project require either a massive influx of company resources (usually at the expense of other new product development projects), a reduction in project scope (resulting in a product that meets fewer customer requirements), or a delay in the launch of the new product. In the very rare case, and only as a last resort, the Company will cancel a product development project.

The worst case scenario for this Company and its competitors in the market is to launch a product that is later found to have a problem or issue(s) significant enough to require a recall. A recall in the automotive industry is extremely costly and the costs are both direct and indirect. Perhaps more important than the direct costs of executing a recall that come from fixing the problem are the indirect costs incurred by the company in terms of lost goodwill with customers and damage to the company reputation. As the pressure grows to bring new products to market more quickly and the trend of discovering issues late in the development lifecycle continues, the risk of launching a product that will ultimately require a recall becomes more and more real.

In an effort to meet the demands of a growing market and overcome the problems habitually experienced in the execution of new product development projects, beginning in the mid-1990's and continuing through today the Company has placed considerable resources and effort into improving its product development capability. The Company built and subsequently expanded a state-of-the-art product development facility to house its product development activities. Included in this new facility was a new test facility that is able to conduct component level bench testing. In addition to its test capabilities, the Company also added a limited capability to do rapid prototyping of some of its components in-house as well as the ability to build 3-D physical prototypes to support mock-up vehicles. Lastly, the Company invested considerable resources in developing its

computer-aided design (CAD) capabilities to include the development of a CAD database that allows engineers to access the latest version of designs of all vehicle parts.

In addition to the physical and technological improvements, the Company also made a couple of significant organizational changes in an effort to bolster its product development capability. These organizational changes, which will be discussed in more detail below, include reorganizing the engineering division responsible for new product development into a matrix organization and formalizing (and codifying) its new product development process.

THE ENGINEERING ORGANIZATION

The engineering organization in the Company I researched is "responsible for the conversion of customer (and/or stakeholder) requirements into drawings and specifications which, when produced within the specified limits, meet or exceed said requirements" (Internal Company document, 2003). The organizational structure of the engineering organization is self-described as a matrix organization. A depiction of a generic matrix organization similar to the one used in the Company is included in Figure 3. While the exact organizational structure of the Company I researched is not included here because of a privacy agreement, I will discuss the application of the matrix structure within the organization in some detail below.

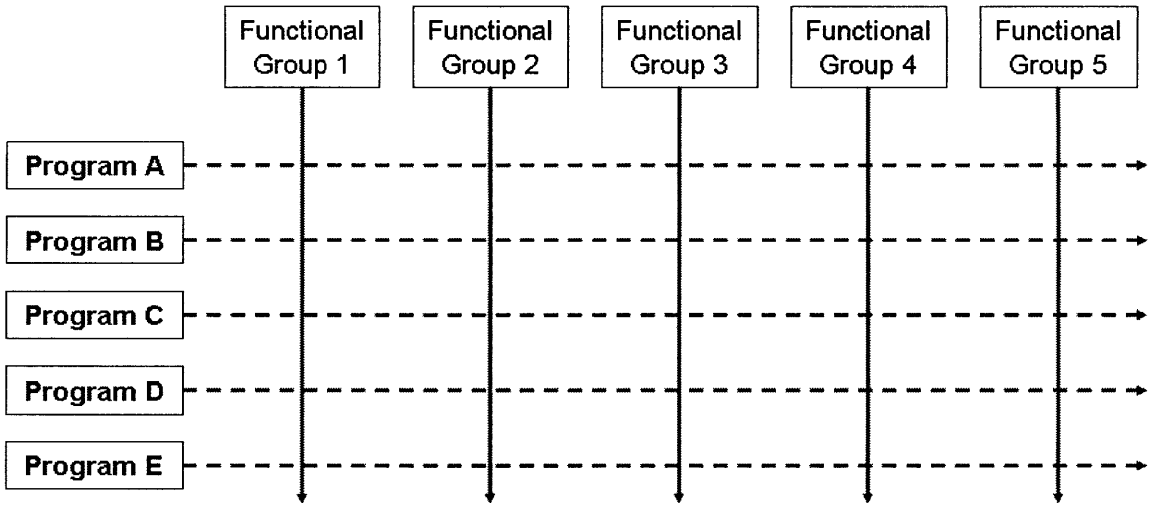


Figure 3 Generic Matrix Organization

The two primary precursors to the matrix organization are the *functional group* organizational form and the *pure project* organizational form. In the former structure, the organization is divided into functional groups that focus on specific disciplines or technologies. The latter organizational structure assigns people to multi-disciplinary project teams which are formed based on the needs of the present project. While the functional group organizational form allows engineers to develop expertise in a given discipline and stay abreast of the latest technological innovations, it creates integration problems as multi-disciplinary projects are undertaken that require communication across functional groups. Unfortunately, these types of complex, multi-disciplinary projects are the norm rather than the exception in most R&D organizations. Creating multi-disciplinary project teams, as is done in the pure project form of organizations, is a means of overcoming these integration difficulties. However, assigning engineers full-time to these project teams often diminishes their ability to maintain ties to their particular discipline often resulting in poorer technical performance than under the functional group organizational form (Marquis and Straight 1965).

The matrix organizational form was first developed and used in research and development organizations in an effort to bridge the gap between these two organizational forms and perhaps enjoy the best of both worlds while minimizing the difficulties associated with each (Marquis 1969; Kingdon 1973; Allen 1977). In the matrix organization, individuals are aligned under (and often formally assigned to) functional groups. These functional groups make up the vertical portion of the matrix structure – the columns if you will. Program managers are then charged with the responsibility of serving as integrating forces across the functional groups to bring multi-disciplinary projects (or products) together. The program managers and their staffs comprise the horizontal portion of the matrix organization – the rows. In the new product development organizations within the automotive industry, there is usually a program manager and staff of engineers for each of the vehicle platforms (or major vehicle models).

This is true in the engineering organization of the Company I researched. In this case, the program managers (referred to as "Platform Managers") for the various vehicle platforms have small design engineering staffs permanently assigned for ongoing platform maintenance, project leadership and project participation. It is from this small group of engineers that project leaders are selected for product development projects that focus on a particular platform but span multiple vehicle systems. The functional groups of the generic matrix organizational structure represent the various vehicle system groups (e.g. powertrain, chassis, electrical systems, styled surfaces, etc.) that have technical design responsibility for the major vehicle systems in the client Company. A majority of the design engineers within the engineering organization in the Company are assigned to these system groups. Engineers assigned to the system groups provide support as full or part-time members of project teams. Alternatively, they can be assigned to do more basic research and development work within their vehicle system group. In addition to these vehicle system groups,

the matrix organization in the Company I researched also includes separate functional groups for Test & Operations as well as for Parts & Accessories.

Lastly, the engineering organization in the Company also includes the product development office (the PDO) which works directly for the Vice President of Engineering. This group of individuals with experience as both design engineers and project leaders is responsible to lead the planning, execution and continual learning involved in launching new and better products into the market place. In particular, the PDO is responsible for providing oversight of the ongoing new product development projects and for publishing the formalized new product development process.

NEW PRODUCT DEVELOPMENT PROJECTS

New product development projects within the client Company are classified into one of 6 categories, or "bins", based on the complexity, risk and size (i.e. resource requirement) of the project. Designating projects into bins allows the Company to set the timeline for a given project – that is when the product will launch and thus how much time there is to complete the development project. Designating the project into a particular bin also establishes the degree to which the project will be held to the fixed product development methodology. Based on the size of the project, its complexity, the timeline for the project and the degree to which the project will follow the standard methodology, the Company is able to assess the risk of a given project.

An additional benefit of designating projects into bins is that it allows the client Company to manage a portfolio of new product development projects. Over time, the Company has developed a good sense for the number of projects of various bin sizes that it can execute at any one time. For example, the Company has learned that the engineering organization can generally only execute a single Bin 6 project (the largest and most complex of projects) in a given year. Armed with this

information, the Company is able to develop a product plan that allows it to manage the lifecycles of its various products by including development projects of various bin sizes sequenced over time to ensure each vehicle platform remain relevant in the market.

In the client Company once projects are placed on the product plan project teams are created that are comprised of design engineers from the various vehicle system groups and/or platforms as necessary to meet the needs of the development project at hand. In most cases, although not always, the project manager (referred to as the "Project Leader") assigned to lead the project team comes from one of the platform managers' staff. For example, in the matrix organization depicted in Figure 4, the project leader for Project 1 would most likely be a design engineer from the staff of Platform A and he or she would be responsible for integrating the efforts of engineers from System Groups 1, 2 and 3. However, Project 2 which might represent a specific technical upgrade that is intended for Platforms B through E might be led by a design engineer from System Group 1. In the case of Project 3 which spans multiple system groups and multiple platforms, the project leader could come from either one of the affected platforms or system groups.

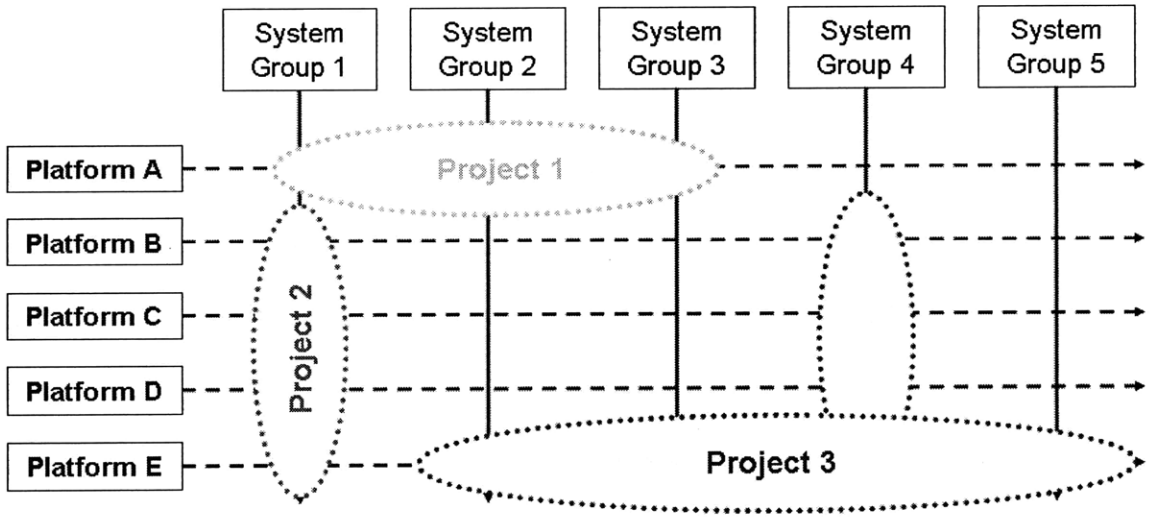


Figure 4 Projects within the Matrix Organization

Herein lays the dilemma. In a matrix R&D organization, the individual project team members are "by design" beholden to two different mangers – their program or project manager and their functional group manager. This holds true in the matrix engineering organization of the Company I researched. In this case, the design engineers assigned to work on new product development projects often must answer to two different bosses – the project leader and their system group manager. This often creates a source of conflict as individual engineers must choose between differing guidance. The project leader has a clear focus on the project at hand and will direct the engineers assigned to his or her project team accordingly, often focusing on meeting project deadlines and delivering a product that satisfies the customer. On the other hand, the vehicle system group managers are primarily concerned with their arm of the organization, specifically their technological discipline or functional area, and will guide their assigned engineers as they deem appropriate, often on longer-term objectives such as developing new technologies. In many instances in the Company, the situation is even worse as individual design engineers are assigned to more than one new product development project at the same time and are forced to allocate their time between multiple projects as well as fulfill their system group responsibilities.

The type of project team described above is lies somewhere in between what Clark and Wheelwright would categorize as a lightweight and heavyweight team. In a lightweight team, the members physically reside in their functional areas but each functional organization designates a liaison person to represent it on a project coordinating committee to coordinate and integrate the different functions' activities. In the heavyweight team there is a project manager who has direct access to and responsibility for the work of all those involved in the project where the core members of the team may be dedicated to the effort and at times physically co-located (Clark and Wheelwright 1997).

In the client Company, smaller scale new product development projects usually result in a project team comprised of design engineers that are allocated on a part-time basis to the project. During the course of the project, these engineers usually reside (i.e. their work desk is physically located) within their assigned system group area. In many cases, the design engineers from the systems groups working on the project will change during the course of the project with no "dedicated" engineer ever assigned to the project.

However, large product development projects are normally staffed with full-time, dedicated design engineers from each of the involved system groups. Additionally, the product development center at the Company maintains a number of dedicated areas for the project teams of large scale projects. In this case, design engineers assigned to the team reside full time in the dedicated project team work area.

MANAGING NEW PRODUCT DEVELOPMENT PROJECTS

Given the nature of the matrix structure of the engineering organization responsible for new product development within the client Company, the project leader is clearly faced with a difficult challenge in managing the efforts of the members of his or her project team. This challenge takes on a number of forms. The project leader must compete for the best engineers to be assigned to his or her project, compete for their time and efforts once assigned to the project, and then manage the direction of their work.

First, the project leader must compete with other project leaders to have the "best" engineers assigned to their particular project. As most engineers are formally assigned to system groups and project leaders are merely allocated "resources" in the form of full-time employee (FTE) equivalents, the project leaders must rely on their ability to influence the system group managers to

fill these FTE slots with the "best" engineers. In the best case, a project leader may be able to get a system group manager to fully dedicate his or her most seasoned and technically proficient engineer to the project. At the other extreme, the system group manager may choose to not even allocate an engineer "by-name" to the project and cover the work on an ad-hoc basis for the duration of the project.

Once the project leader has been allocated his or her project team members, he or she must work to manage the level of effort that these individual engineers direct towards the given project. Remember that in the matrix organization engineers working on multi-disciplinary project teams may also still carry responsibilities to their system group. Managing the level of effort is made more difficult for the project leader when engineers on the team are assigned to more than one project at the same time. Project leaders must rely on their ability to influence the individual engineers to allocate time and effort toward their particular project. The project leaders may also rely on their influence with system group managers in order to get more dedicated effort from individual engineers assigned to their project.

An additional challenge that the project leader must face is in managing the direction of efforts of their individual engineers. Again, in the matrix organization the project leaders and the system group managers often have competing agendas. A system group manager may push an individual engineer towards a specific technical solution (e.g. getting the latest technology into the product) that might be outside of the scope, budget, or timing limitations of the project that the project leader is managing. In this case, the project leader must rely on his or her ability to influence the system group managers in aligning their functional area agendas with that of the project in order to provide clear and non-conflicting guidance to the engineers.

In addition to managing the efforts of project team members, the project leader also faces the challenge of managing the scope of the project. Alluded to earlier, the system group managers may have agendas that lie in contravention to that of a given project. This dilemma can also be true of projects that cross programs, or platforms. As the project leader works to manage the push and pull of the various agendas that can impact the scope of his project, he must rely on the influence he has with the various system group and platform managers.

Clearly, a project manager who maintains a strong network of professional ties to both functional and program group managers, as well as to individual project team members, will be better able to manage the many challenges of project management in the R&D matrix organization. By building and maintaining a strong network, a project manager can come to rely on the gained social capital to influence behaviors within the project team as well as to secure resources, mediate conflict and run interference external to the project team. In fact, project performance and the locus of influence in a matrix-structured R&D organization have been studied by (Katz and Allen 1985). They found that when R&D project team members perceive their project manager to have a stronger organizational influence than their respective functional managers in the matrix organization, project performance is higher.

Perhaps the most significant challenge faced by the project leaders in the client Company is to bring about effective integration of the design efforts within their project teams. The nature of the integration challenge was described in detail in Chapter Two. Internal company assessments of past development projects, previous academic research within this organization (Black 2002), and my own interviews with project leaders and involvement in design reviews in the client Company all highlight the difficulty of integrating the design efforts of individual engineers working on various vehicle components that are coupled with components being designed by other engineers.

Given the busy work environment of design engineers faced with competing demands, it is often difficult enough for project leaders to get individual design engineers assigned to their project to dedicate time and effort to work on their project let alone to get two or more engineers to work in coordination with one another. It is often not until faced with an impending deadline (normally a full vehicle build event) that engineers are able to block time to work on a given project. Faced with increasing time pressures, they often focus their efforts on completing the design of their individual part or component without regard for the need to integrate their design with the designs of coupled components. As one design engineer put it: "All I care about is getting my part to the build on time."

Given this mindset, it is not hard to see why there are a large number of integration errors discovered in build events. Often times this is the first time that the individual designs, now in the form of prototyped parts, are assembled and tested together. These integration errors invariably require design changes in one or more components. These design changes that result from a lack of integration reinforce the tendency of individual engineers to design their parts or components in isolation as they conclude that integration efforts, especially early in the development process, are a waste of time. As one design engineer asked: "Why should I worry about integrating my part with theirs since I know their design is going to change anyway?"

It is in this context that the project leader must try to bring about effective integration within his or her development project. As was mentioned previously, the tendency of project leaders is to increase the number and frequency of formal design reviews as a means for bringing together the individual assigned engineers in an effort to affect integration. However, in my experience and based on interviews of participants, these design reviews very often amount to little more than a series of reports by individual engineers on the status of their individual designs / parts. Very little

real "integration" is accomplished in these reviews. However, the increased capability and use of computer-aided design (CAD) in the Company has improved the amount of integration that a project leader is able to affect in these reviews. At a minimum, the CAD capability allows the project leader and the design engineers to see a 3D virtual model of the latest version of those parts with designs in the CAD database. The CAD capability also allows the engineers to virtually build various parts of the vehicle and check the geometry for fitment of parts. This capability allows for the discovery of certain fitment errors prior to the physical prototyping of parts and the full vehicle build event.

Despite the growing capability and use of CAD engineering, most of the integration issues are still discovered during vehicle build events and subsequent full vehicle testing. Past research (Black 2002) and my own experiences within the client Company indicate that these build events are focal points within the new product development organization and the best indicators of the true progress of product development projects. Indeed, in the most recent changes to the codified development process in the Company the build events have been formally recognized as the key events that pace product development projects. With this in mind, let us look at the new product development process in use at the client Company. In the next two sections, I will review the formal codified development process as well as review some of the ground truth of product development that I discovered during my time in the client Company.

THE FORMAL PROCESS

The formal process adopted by the Company is known as the Concurrent Product & Process Delivery Methodology (CPPDM). The methodology has been in place within the Company for over 15 years. It has gone through a number of revisions over that period and at the time I

conducted my research the Company was using its 7th version of the methodology. Incidentally, the Company recently implemented its 8th version of the CPPDM. However, the methodology I will describe below and that I based the development of my simulation model is reflective of Version 7.

The methodology applies to all development projects that result in a change to original equipment (OE) vehicle products. The intent of this methodology, as outlined in internal Company documents, is to "define the overall product development effort and clarify the roles and relationships between engineering, purchasing, production, and various other departments." The tools set forth in the methodology enable and encourage communication between the departments. Presumably this interdepartmental communication is intended to facilitate the concurrent execution of development tasks in an effort to streamline the development process and minimize the time to market of new products.

The CPPDM can best be characterized as a stage gate development process. The methodology consists of 5 phases each with its own set of requirements. While I do not intend to provide a detailed account of the activities and requirements of each of the phases included in the CPPDM, I will provide and overview of the phases in development process and highlight the key activities and requirements in each.

Prior to the beginning of any formal development activity, indeed before a product development project falls under the formal methodology by entering the first phase, a number of new product ideas circulate throughout the company competing for attention, support, and legitimacy. It is amidst this swirl[6] that members of the organization learn about the idea, support for the idea grows, and coalitions build as competing ideas are assessed against one another. It is

[6] Swirl is a term used by a senior manager in the engineering organization to describe the early and ongoing conversations about ideas for new products vying for support and traction within the Company.

from this swirl that a champion for the idea emerges who can sell the idea up, down and across the organization ultimately resulting in its formal acceptance and inclusion on the lifecycle plan. It is at this point that the idea or concept becomes a development project and is subject to the formal methodology beginning with Phase 0.

Phase 0: Concept Definition

The primary purpose of Phase 0 is to identify the expectations of the project. This involves developing a business rationale for the proposed idea or concept including quality, cost and timing (QCT) objectives. A project leader is designated and a core team consisting of a representative from each of the affected system and functional groups is authorized. A visual or physical representation of the part or vehicle is created. Additionally, full vehicle, system, and sub-system level requirements are defined and a feasibility plan is developed. Lastly, an initial project plan that includes estimates for project staffing, test facilities and funding required to design, develop, and produce the product for Phases 1 through 4 are produced. At this point if the project is approved, it is placed on the Product Plan and enters Phase 1.

Phase 1: Feasibility & Design

The purpose of this phase is to show that the project is feasible with enough confidence that a launch date can be set for the new product. At least one design alternative must be demonstrated to meet project requirements (both the customer and business requirements) with a high level of confidence. It is during this phase that the business case is developed representing the rationale for the project and the criteria for success of the project is defined. The core project team is expanded and cross functional resources are assigned with the requisite skills necessary to define, design, develop, and produce the product. Developmental part numbers are assigned, bills of material are developed, design drawings and CAD models are generated. Ultimately, a physical mock-up of

the new product representative of the latest design intent is constructed that demonstrates proper form and fit and the requirements developed in Phase 0 are translated into specifications. During this phase, the project team should also be able to demonstrate that the product and process requirements can be met within 18 months – the scheduled time between Phase 1 exit and the start of full vehicle production.

Phase 2: Integration & Verification

In this phase, the project is verified to show that the design solution meets the expectations of the project. The design, development, and vehicle integration of components and subsystems for both the product and process is demonstrated and documented as complete. The design integrity of system, subsystem, and component parts is proven and verified by test data and analysis. Perhaps the most significant event that occurs during this development phase is the Design Intent Build (DIB). It is at this build event, which normally occurs about 10 months out from the intended product launch date that the integration of systems, subsystems, and components which represent the design intent are verified through building and testing full vehicles. While the actual building (assembly) of the vehicles is normally done on the factory floor to test the process plan, the individual components or parts need only be representative of design intent and do not have to be manufactured on-site as part of the build event or produced by a supplier using their finalized production tooling and process. At the conclusion of Phase 2, the project team must collectively determine that the design intent has been verified to meet project requirements and that the process is able to consistently deliver the product at the intended full-scale production rates while meeting the cost and quality levels established in the business case. For all intents and purposes, the design and process intent should be frozen at the completion of Phase 2.

Phase 3: Validation

The primary purpose of Phase 3 of the CPPDM is to validate the project and show that it is ready for full-scale production. In other words, the new product is shown to be ready for launch. During this phase all components, sub-systems, and systems have documentation showing their compliance with all requirements. The process plans are all in place and all plant personnel training is completed. The production systems for all components that support the product are shown to be stable and meeting the requirements and specifications at the required production rate. Validation of the product and process is achieved in large measure through the First Production Event (FPE) build during which vehicles are built and produced from production tooled and production intent processed parts on the actual assembly lines using production intent processes at the required production rates. The FPE build normally occurs four months out from the intended product launch date. Subsequent testing of the vehicles validates that production parts and processes will comply with all requirements. Lastly, the project team reassesses the business case to confirm that all requirements and targets are met. Exit from Phase 3 of the CPPDM is approved only after a comprehensive review by a group of senior company leaders, normally at the vice president level, responsible for providing quality and reliability oversight for the Company. If successful, this review results in the approval of the product to launch.

Phase 4: Make Good & Closure

The primary purpose of Phase 4 is to close out the product development project. In order to complete Phase 4, the project team must ensure that all project issues, high priority test incident reports (TIR's), and manufacturing issues, engineering orders (EO's) and test reports are addressed and closed. They must also make sure that all support material, service parts and core accessories are complete and available and that the required mechanic/technician training plans have been

developed. Lastly, in order to close out the project, the project team conducts an after action review to assess what happened (good and bad) during the course of the project and capture lessons learned.

THE REAL PROCESS

The formal process, the Concurrent Product and Process Development Methodology (CPPDM), described in the previous section guides the execution of new product development projects in the client Company. In this section, I will provide some insights into the way that the methodology is carried out and how it translates into the day to day execution of a product development project. The collection of insights included below was gathered from the interviews that I conducted with engineers and project leaders in the Company and from a review of the internal assessments of past development projects. It is not intended to be all encompassing but rather highlights those features of the "real" development process in place in the Company that informed the development of the simulation model described in the next chapter.

Project Commitment

New product development projects in the Company range from as short as 24 months for a small scale project (Bin 1) to as long as 6 years for the largest and most complex (Bin 6) projects. The variation in the length of project derives from the amount of time a project stays in Phase 0 (Concept Definition) and Phase 1 (Feasibility & Design). Projects that have difficulty settling on a design that proves feasible can spend as long as three years in Phase 1. It isn't until a project exits Phase 1 that people in the organization outside of the core project team recognize it as a "real" project and begin to commit to it by supporting it with resources. Ironically, this reluctance to

commit to projects still in Phase 1 contributes to the extended time that some projects spend in this phase.

Once a project exits Phase 1 and enters Phase 2 (Integration & Validation) it is placed on the Product Plan with an expected product launch in 18 months. With this "hard date" set, not only do individuals commit more fully to the project but so does the Company. At this point, it is extremely rare for the Company to cancel a project no matter how poor its performance. As the product launch date approaches the Company has a recognized history of going to Herculean efforts to rescue those projects that aren't ready for launch by diverting resources from other projects to assist the struggling projects. As one Company project leader stated: "You want to be the biggest project that is furthest behind and closest to launch because that is the project that is going to get all of the resources." The impact of this persistent firefighting has been studied by (Black and Repenning 2001; Repenning 2001)

Product Launch

The Company launches almost all of its products, certainly all of its significant new products, at the same time each year. This is done to take advantage of the seasonal nature of demand and to leverage the exposure its new products gain at some of the larger gatherings of its customers. Having the Company's products launch at the same time and each product development project follow the same methodology and timeline inherently causes surges in demand for resources and ultimately a backlog or wait for some of the resources. Perhaps the most significant of these bottleneck resources is the vehicle test facility. A backlog at this facility delays the test results that are essential to the effective development of new products.

Design Engineer Work Priorities

All of the engineers that I interviewed and observed at the Company came across as committed to the Company and their work. They firmly believe in the products the Company sells and most are owners themselves of one or more of the Company's products. Once they become invested in a new product development project, the engineers work tirelessly to make sure the project succeeds. For most engineers, they become invested in a development project when it exits Phase 1 and is placed on the Product Plan and scheduled for a product launch date – a sign that the Company has committed to the project. For the core members of the project team, commitment to the product idea originates in Phase 0 or perhaps even earlier as the idea gained traction amidst the "swirl" of product ideas. Once committed, the engineers genuinely "buy in" to the new product they are developing, confident that in the end the product they are developing will be a market success.

Unfortunately, many design engineers are working on and committed to multiple projects. As such, they are forced to prioritize their work efforts between projects. Most of the engineers I spoke with admitted to using an "earliest due date" approach to prioritizing their work. The significant due dates for these engineers are the dates on which their parts are due at the site of the next build event. The engineers backwards plan their work schedule based on the dates of the vehicle build events. Whichever project they are working on that has the next scheduled build becomes their priority.

The engineers' primary concern is to make sure they have their part at the build on time. Almost everything else appears to be subordinate to this priority. As mentioned, when faced with schedule pressure (too much work and not enough time) these engineers will design their part in isolation at the expense of the necessary coordination with other engineers working on coupled

components in order to make sure that they have their part at the build on time, albeit with a high probability of the part having an integration error. In the mind of the design engineer, it is better to have a poorly integrated part prototyped and ready for the build than to have a well-integrated design that is not prototyped in time for the full vehicle build. This line of thinking does not seem unreasonable given the amount of resources the Company invests in build events. An engineer does not want to be responsible for delaying or canceling a build event because they don't have their parts available.

Bench Testing

Despite a significant investment by the company into developing in-house bench test capabilities at the new product development center, a large percentage of components are not bench tested prior to the design intent build (DIB). In some projects, it was reported that less than 10% of the component parts had been bench tested prior to the design intent build. This is particularly troubling since the DIB is intended as a validation of individual parts and to test the integration of those parts in a full vehicle. The fact that design engineers are using the DIB and subsequent vehicle testing as the first means of testing their individual components explains why the designs are rarely frozen, as intended, at the DIB and that a large number of failures and issues arise.

A few of the reasons that design engineers avoid bench testing their component parts were discussed in Chapter Two. A lack of available time is most often cited by both design engineers and test engineers as a reason. It is not uncommon for engineers to complete their designs with just enough time get the corresponding parts prototyped in time for the next scheduled build. This doesn't leave them with enough time to bench test their parts prior to the build. Once a full vehicle is built, engineers see little reason to bench test their component parts as they view the full vehicle

testing as much more reliable and more realistic providing richer information than bench testing. Some engineers that I talked to felt even more strongly and indicated that they didn't trust the results of bench testing. They felt that the bench test conditions were unrealistic and that the test engineers tended to "over test" their parts and generate specious errors.

Build Events

While there are only two official builds mandated by the formal methodology, design intent build (DIB) and first production event (FPE), some projects have executed as many as eight build events. A common feature of struggling projects is the need to schedule a second DIB when a large number of issues are discovered during the initial DIB. In the extreme case, a development project may be forced to execute a second FPE to validate the product and process design prior to launch if significant issues still remain at the original FPE. Some projects struggle early in the development process and execute a number of prototype builds during Phase 1 in an effort to prove the feasibility of the product design. Alternatively, some projects are able to effectively use early prototype builds to set the course for a successful development effort.

As has been mentioned throughout this report, build events are a key feature of the development process at the Company. Much of the learning takes place in conjunction with build events and the subsequent vehicle testing. Project managers use builds to affect integration in their project teams. Design engineers prioritize their work around build events. Much of the development cost of a new product is tied up in vehicle build events. The Company's development resources are significantly allocated around build events and vehicle testing. In sum, the pace of the new development effort in the Company is set by the build schedule.

Chapter 4: System Dynamics Model

"All decisions are based on models ... and all models are wrong."

-- John D. Sterman, J. Spencer Standish Professor of Management and Director of the System Dynamics Group at the MIT Sloan School of Management

INTRODUCTION

Over the past few years, the Company has observed an increase in the late discovery of technical problems and the accumulation of unresolved issues very late in their product development projects which threatens the viability of products launches. They have also observed an increase in the number of full vehicle builds required by development projects in order to validate and verify their product and process designs. Additionally, many of the development projects have reported difficulty in timing their design iteration cycles as the learning rates of the various components and subsystems within a given project differed and were often out of synch with each other and the build events leading to the propagation of late changes among coupled components. In order to assist the company in identifying and understanding the underlying causes of these problems I developed a stylized system dynamics model of the Company's new product development process.

I based the new product development model (NPD Model) on the field research that I conducted at the Company. It is based on both the formal development process (CPPDM) and the

informal practices and decision rules that I observed in the Company. The NPD Model developed here also leverages the basic structure of the project model developed by (Ford and Sterman 1998) to study new product development processes.

The main focus of the model I developed is on the design-build-test learning cycles of two coupled subsystems that iterate at different speeds. As in the Ford and Sterman (1998) model, it accounts for the development tasks of initial design work, rework (or redesign) and coordination. The present model also includes the building of prototype parts based on designs and the subsequent testing of those parts to discover errors. It accounts for component errors (an error in a given component) and integration errors (an error between coupled components) in both designs and prototyped parts. The model allows the state of paper designs to evolve while their corresponding prototype parts, representing the state of the designs at the point when they were built, undergo full vehicle testing. This means that design and/or redesign work can be done at any time including during the vehicle builds and subsequent vehicle testing. The model allows for the overlap of learning cycles meaning that multiple vehicle builds and subsequent testing can be ongoing simultaneously. The model allows for the execution of any number of learning cycles highlighted by build events during the course of the development project. The timing of each of these learning cycles, set by the build dates, can vary individually as well. The model incorporates resource management in terms of design engineering manpower allowing for the investigation of the impact of different resource levels and allocation schemes on project performance. The decision rules built in to the model structure that handle the allocation of resources recognize that designers must allocate their time between design, redesign and coordination work. The model allows the quality of design (and redesign) work to vary based on work pressure and distinguishes

between component-specific quality (how well I design my component to perform its functions) and integration quality (how well I design my component to interface with coupled components).

The model was developed as a stylized model as it is based on the development process used in the Company. However, the model is not intended to capture the entire development process nor is it intended to entirely replicate a specific portion of the development process. Perhaps most notably, the NPD Model was developed to capture the interaction of only two components, or subsystems, whereas the typical new product development project at the Company consists of many more interacting components or subsystems. In this manner, the model was developed to demonstrate to the client company how the structure of the development process, even in this simplified case, can lead to the observed behavior in past new product development projects. The model was also developed to demonstrate the impact that each of the key parameters and decision variables can have on project performance as a means for the client company to consider potential policy changes. Lastly, the model serves as a "proof of concept" and a means for explicating the notion of phantom work.

MODEL OVERVIEW

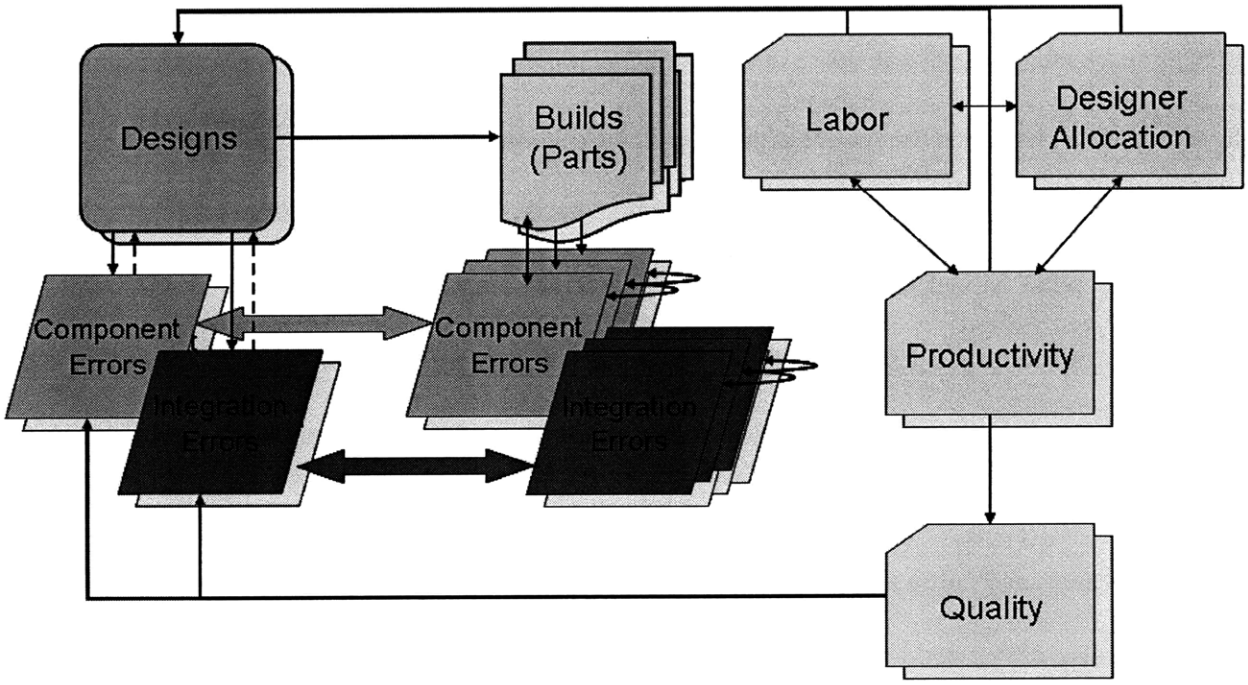


Figure 5 High Level Model Structure

At a very high level, the NPD model can be viewed as a set of paper designs that are constantly evolving and a set (or sets) of parts that correspond to the state of the paper designs at the time they were prototyped for a build event. Along with the set of paper designs are a corresponding set of component errors and integration errors. These errors are created and fixed as part of the ongoing design and redesign of the paper designs. Similarly, there are sets of component and integration errors that correspond to the set of build parts. These errors reflect the errors that existed in the paper designs at the time the parts were prototyped. As component and integration errors are discovered in parts through vehicle testing their corresponding paper designs are sent for rework. Because it is possible to overlap the design-build and test cycles, there can be multiple versions of build parts undergoing vehicle testing. This means that there can be multiple versions of component and integration errors associated with those parts. Some of these errors are common

across versions of the build parts. In addition to the two main sectors (designs and parts, with their corresponding errors) of the NPD Model, there are four other significant sectors which control the flow of designs, parts, and errors in the model. The first is the Labor sector which determines the number of designers that are available to design redesign and coordinate work for each of the components based on the amount of work to be done and the time remaining until the product is scheduled to be launched. The second sector is the Productivity sector which determines the productivity rates for the design engineers in conducting design and redesign work accounting for the schedule pressure inherent in the new product development environment. The Designer Allocation sector models the process by which designers allocate their time between working on design, redesign and coordination tasks. Lastly, the Quality sector is used to model the quality of design and redesign work done by the design engineers as they respond to schedule pressure. This sector obviously affects the rate at which errors are created and fixed in the evolution of paper designs. Given that the NPD model simulates two coupled components being designed in parallel, these main sectors as well as the other sectors of the model are replicated for each of the components.

In total there are 13 sectors of the model each of which will be described below. Each sector is designed to capture a specific physical process or decision rule that is found in the development process at the client company. In this chapter, a section is provided for each of these sectors describes the basic model structure and provides the rationale for its use. A more detailed description of the model, including a listing and description of the model equations as formulated in the simulation model is provided in the appendices.

In order to make the model documentation provided in this chapter more readable, *italics* are used in the text when specifically referring to a model variable. Additionally, references to

functions used in the model equations that are available in the VENSIM simulation software are written in ALL CAPS.

DESIGN SECTOR

Design Sector Overview

The design sector of the NPD Model is structured to account for the current status of the set of designs for a new product development project during its detailed design phase. This sector includes the set of designs (paper drawings, CAD drawings, or prototyped parts) corresponding to the individual parts that make up each component of the new product under development. These designs reflect the latest version (revolution) of the design for each given part. The design sector also accounts for the component errors and the integration errors associated with these designs. Figure 6 provides a high level depiction of the design sector of the NPD Model.

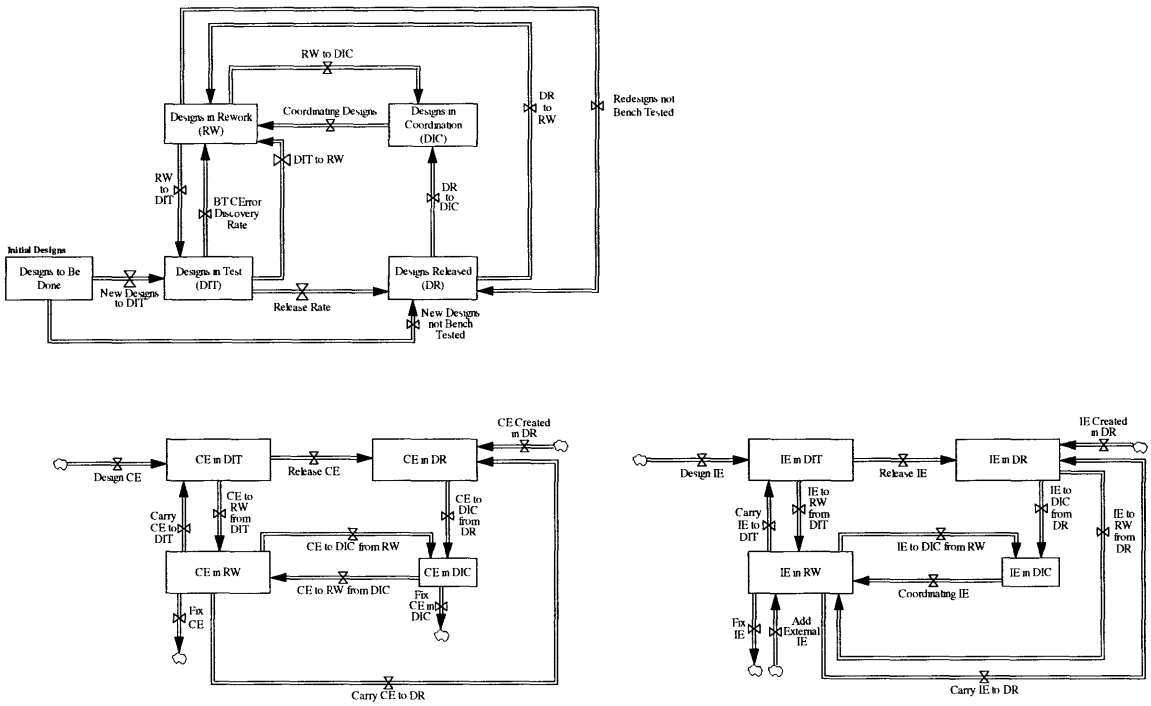


Figure 6 Design Sector

The basic structure of the design sector consists of three stock and flow chains. The main stock and flow chain accounts for the set of designs as they are developed during the detailed design phase of the new product development project. This stock and flow chain is the in the upper left of Figure 6. There are 5 stocks used in the main stock and flow chain: *Designs to Be Done, Designs in Test, Designs in Rework, Designs in Coordination* and *Designs Released.* These stocks are depicted by boxes with the variable names inside of the box and are used to represent the different states that a given design can be in at any given time. For example: a design that has not yet been worked on would reside in the stock of *Designs to Be Done.* As the designs are worked on and evolve over time, they can transition between the different states. For example, a design from the stock of *Designs to Be Done* that has been completed can be sent for bench testing and would transition, or flow, into the stock of *Designs in Test.* The flow of designs from one stock to another is depicted by the arrows that connect the boxes with the flow assumed to be in the direction of the

arrow. These arrows can be thought of as pipes with control valves. The variable names that control the rates of flow can be found next to the hourglass shaped symbol on each of the arrows.

Co-flows are stock and flow chains that correspond to a main stock and flow chain in a system dynamics model. Co-flows are used to account for properties of the items in the main stock and flow chain. In the case of the NPD Model, two co-flows are used in the design sector to account for the errors, both component and integration, associated with the designs in the main chain. These stock and flow chains are nearly identical to each other and are depicted at the bottom of Figure 6. With the exception of the stock of *Designs to Be Done*, there is a corresponding stock of errors for each of the stocks of designs in the main stock and flow chain. Because there are no errors in a design that has not yet been worked on, there is no need to maintain a corresponding stock of errors for the stock of *Designs to Be Done*. Additionally, there is a flow between each of the stocks in the co-flow structure of the component errors (CE) and integration errors (IE) corresponding to the flows between the stocks in the main stock and flow chain of designs.

A more detailed description of the Design Sector (and each of the other sectors of the NPD Model) follows. More technical documentation to include a detailed description of each of the model sectors, the rationale used in modeling each of the simulation variables and their respective equations, as well as a complete equation listing can be found in the appendices.

The NPD Model is developed for a new product that has two main components – component A and component B. As such, there are actually two identical sets of stock and flow chains used in the design sector and indeed throughout much of the NPD Model. The simulation software, VENSIM, handles multiple instances of stocks, flows, and auxiliary variables using subscripting.

The use of subscripts is prevalent throughout the NPD model. There are two main uses for subscripting. The first, as mentioned, is to distinguish between the two components simulated in

the model. Indeed, almost all of the variables in the model are subscripted for the different components – A and B. This is certainly true for the stocks and flows of designs as well as their related build parts which will be discussed in the Build Sector section of this chapter. In most cases the auxiliary variables are also subscripted to account for differences between the two components. The second use of subscripts in the model is to distinguish between multiple vehicle builds. The stocks and flows of the build parts are subscripted for this purpose. In this case, these variables are subscripted for both the components and the builds. For ease of exposition, except where necessary subscripts will not be included in the description of the NPD Model nor in the equations provided as part of the text. As mentioned earlier, technical documentation of the model is included in the appendices to this report. As a reference, Appendix B includes all of the variables and equations in the model with full subscripting included. Where it is not obvious in the discussion, the reader should refer to this appendix to verify the use of subscripting.

Designs

The main stock and flow chain in the Design Sector of the NPD Model accounts for the set of designs as they are developed during the detailed design phase of a new product development project. Figure 7 depicts the main stock and flow chain for a set of designs for a single component or subsystem. As the model includes two components, the stocks and flows are subscripted to account for the set of designs for each component. The designs are tracked through the various stages of development as they are initially worked on by designers, as they are tested either through bench testing or vehicle testing, as they are released awaiting a build event, as they are coordinated when found to have an integration error, and as they are redesigned in an effort to fix known errors.

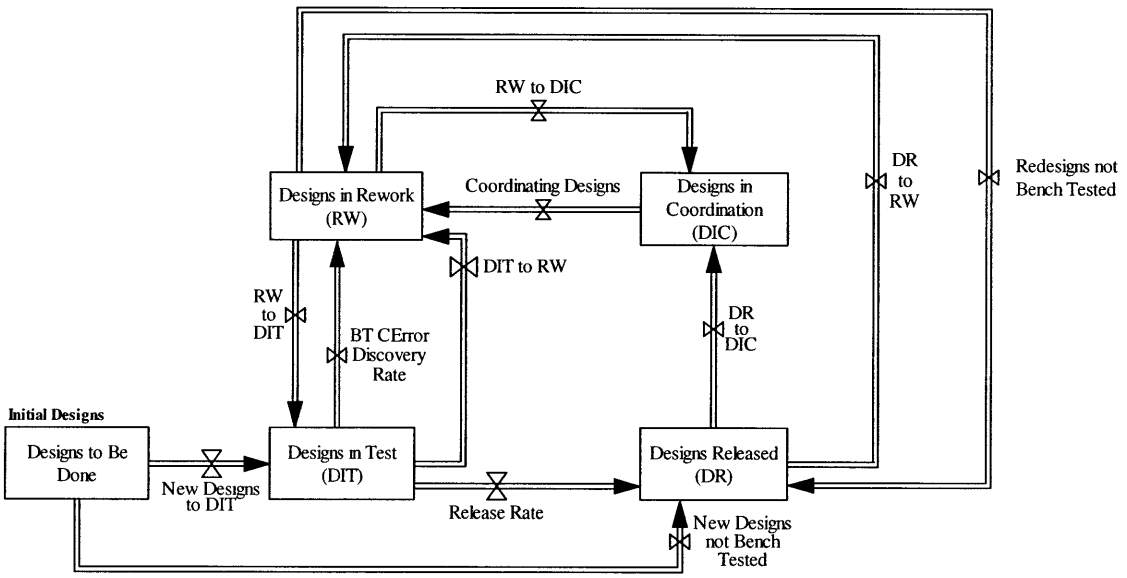


Figure 7 Design Stock and Flow Diagram

Consider a typical component (or sub-system) such as the frame of a motorcycle in the detailed design phase of a new product development project. There is an initial set of designs that must be completed for the motorcycle frame as part of the project. These required designs all begin in the stock of *Designs to Be Done*. As new designs are completed, they can either be sent for bench testing (represented by the flow *New Designs to DIT* in Figure 7) or they can be released without bench testing (represented by the flow *New Designs not Bench Tested*) awaiting the full vehicle build event and subsequent vehicle testing. The stock *Designs in Test* represents those designs that are in the queue for bench testing. As a result of bench testing, a given design might be found to contain a component error and is thus sent to the stock of *Designs in Rework*. The rate at which these component errors are discovered is represented by the flow *BT CError Discovery Rate*. Those designs that are bench tested where no component error is discovered are sent to the stock of *Designs Released* to await the next build event. This is depicted as the flow *Release Rate* in Figure 7. Those designs in the stock of *Designs in Rework* are in the queue awaiting redesign. Just as in

the case of new designs, once a redesign is complete the design can either be sent for bench testing (represented by the flow *RW to DIT*) or they can be released without bench testing (represented by the flow *Redesigns not Bench Tested*) to await the next build event.

During a build event, all of the designs that are in progress are physically built as parts and the full vehicle, including all components, is assembled[7]. Once the full vehicles are assembled they are sent for vehicle testing where both component errors and integration errors can be discovered. The model structure for vehicle builds and vehicle testing will be described in greater detail in the Build Sector and Vehicle Testing sections of this chapter.

As a result of vehicle testing, designs will be found to have either component errors, integrations errors, or both. Depending on the type of error(s) discovered, the designs with errors will be sent either directly to the stock of *Designs in Rework* or to the stock of *Designs in Coordination*. The stock of *Designs in Coordination* is the queue for those designs that have been found to have an integration error and require coordination between designers from the two components involved in the integration error before it can be redesigned and the integration error fixed. As designs are coordinated they are sent to the stock of *Designs in Rework* to await redesign. This process is represented by the flow *Coordinating Designs* in Figure 7.

While those designs found to have only a component error will be sent directly for redesign (*Designs in Rework*) and those found to have only an integration error will be sent for coordination (*Designs in Coordination*), those designs that have both types of errors may be sent for coordination or directly for redesign. The decision about where designs with both types of errors will be sent upon discovery will be discussed in the Coordination Fraction section below. As an example, in Figure 7 the flow *DR to DIC* from *Designs Released* to *Designs in Coordination*

[7] Designs in progress are those designs that, at a minimum, have a completed initial design. This includes those designs in the stocks of Designs in Test, Designs in Rework, Designs Released, and Designs in Coordination.

represents the rate at which designs released are sent for coordination as they are discovered to have an integration error in vehicle testing.

You should note that even designs in the queue for redesign (*Designs in Rework*) may be sent for coordination (*RW to DIC*). This is possible because at the time of a vehicle build all of the designs are built and assembled as part of the vehicle – regardless of their current state. This means that in addition to the stock of *Designs Released*, the designs that are in the queue for bench testing (*Designs in Test*), in the queue for coordination (*Designs in Coordination*), and those in the queue for redesign (*Designs in Rework*) are built and subsequently vehicle tested. As a result, a design in the queue for rework with a component error, perhaps discovered in bench testing, may also contain an integration error that is subsequently found in vehicle testing. As a result of the discovery of the integration error, the design may be sent for coordination prior to being redesigned (represented by the flow *RW to DIC*) or it may remain in the stock of *Designs in Rework* to be redesigned without coordination.

Given this brief overview of the main stock and flow chain of designs, we will now consider each of the stocks of designs and their associated inflows and outflows in greater detail. There is a subsection included for each of the 5 main stocks: *Designs to Be Done*, *Designs in Test*, *Designs in Rework*, *Designs Released,* and *Designs in Coordination.*

Designs to Be Done

The first set of designs in the NPD model to be considered is the stock of Designs to Be Done. This stock and its associated flows can be seen in Figure 8.

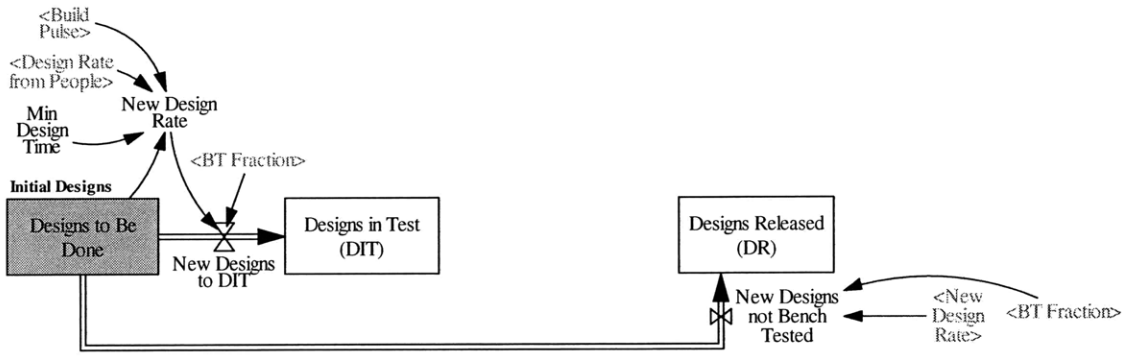


Figure 8 Designs to Be Done

This stock represents the set of paper designs that still need to be completed at any point in time given an initial set of required designs and the amount of new designs that have been completed to date. The stock of *Designs to Be Done* for each component has an initial value based on the variable *Initial Designs*, which is set to 1000 designs for both components A and B. It also has two outflows: *New Designs to DIT* and *New Designs not Bench Tested.*

Both of the outflows from the stock of *Designs to Be Done* are based on the rate at which new designs are completed (*New Design Rate*). The variable *BT Fraction* determines the fraction of new designs that are sent for bench testing upon completion with the complement (*1-BT Fraction*) of new designs being released to await the next build event without bench testing. The fraction of designs that are sent for bench testing is based in large part on the amount of time remaining until the next build event with the fraction decreasing as the amount of time decreases.

The new design rate for each component is set at the minimum of the design rate that is possible based on constrained resources and the productivity of designers (*Design Rate from People*) and the design rate that is possible based on the amount of work available (*Designs to Be Done*) and the minimum time it takes to complete a design (*Min Design Time*). The minimum design time is set at one week for both components and represents the minimum time required to complete a design without any resource constraints. The variable *Design Rate from People* will be discussed in further detail in the Productivity Sector section of this chapter.

The NPD model uses *Build Pulse*, a switch variable of sorts, as a means for shutting off the flow of new designs at the instant of a build event. While shutting down this flow at each build event allows the simulation model to handle the accounting of transferring paper designs into build parts, it also reflects the reality of designers ceasing all other work in order to actively participate in and/or monitor the build event.

Designs in Test

The next set of designs in the main stock and flow chain in the Design Sector NPD model to be considered is the stock of *Designs in Test*. This stock and its associated inflows and outflows are depicted in Figure 9.

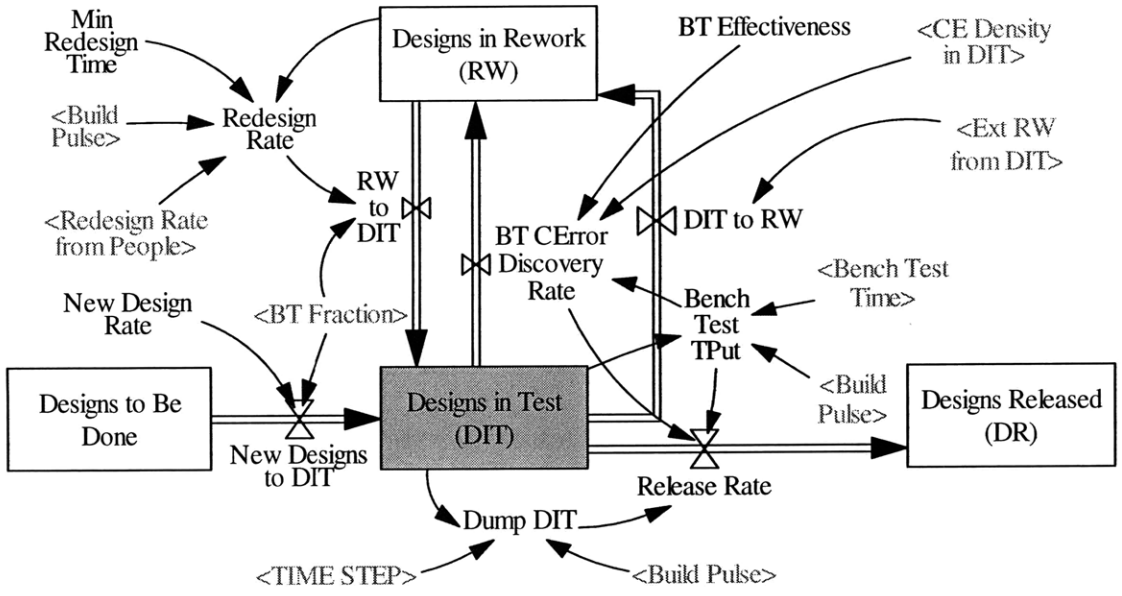


Figure 9 Designs in Test (DIT)

The stock of Designs in Test (DIT) represents those designs that are in the queue for bench testing. The stock of *Designs in Test* begins with an initial value of zero[8] and integrates two inflows (*New Designs to DIT* and *RW to DIT*) as well as three outflows (*BT CError Discovery Rate*, *Release Rate*, and *DIT to RW*).

As discussed earlier, the inflow *New Designs to DIT* represents the rate at which new designs are completed and sent for bench testing. In the same manner, the inflow *RW to DIT* represents the rate at which completed redesigns from the stock of *Designs in Rework (RW)* are sent for bench testing as determined by the redesign rate and the bench test fraction. Just as in the case of the new design rate, the variable *Redesign Rate* is set at the minimum of the redesign rate that is possible based on constrained resources and designer productivity (*Redesign Rate from People*) and the redesign rate that is possible based on the amount of work available (*Designs in Rework*) and the

[8] With the exception of *Designs to Be Done*, all stocks in the NEW PRODUCT DEVELOPMENT model begin the simulation with an initial value of zero.

minimum time it takes to complete a design (*Min Redesign Time*). The minimum redesign time is set at one week for both components and again represents the minimum time required to complete a redesign without any resource constraints. The variable *Redesign Rate from People* will be discussed in detail in the Productivity Sector section of this chapter.

The two main outflows from the stock of *Designs in Test* are *BT CError Discovery Rate* and *Release Rate*. Both of these flows are the result of bench testing and represent the designs in which a component error is discovered in bench testing and sent for rework and those designs that are released because no error is found. The primary driver of these flows is the variable *Bench Test TPut* which represents the throughput rate at which designs are bench tested. The throughput is modeled as a simple first-order delay using an average bench test time which may be different for the two components. The difference in the average bench test times is one of the sources for the different design iteration speeds between components. The other differences that account for the potential mismatch of design iteration speeds that are accounted for in the NPD model are the design and redesign rates as well as the average vehicle test time. The differences among these speeds will be discussed in the Speed Factor section of this chapter. As in the case of the designing and redesigning, bench testing is shut down at the instant of a vehicle build using the *Build Pulse* variable.

In addition to the throughput, there are two main factors that determine the rate at which component errors are discovered in bench testing. The first is the fraction of the stock of *Designs in Test* that actually has a component error. This fraction is represented by *CE Density in DIT* in the model. The second factor is the effectiveness of bench testing at identifying component errors and is captured by the variable *BT Effectiveness* in the model.

The fraction of designs in bench testing that actually has a component error is calculated by dividing the number of component errors associated with the designs in test (*CE in DIT*) by the number of designs in the queue for bench testing (*Designs in Test*). The second factor is the effectiveness of bench testing at identifying component errors. This factor is captured by the variable *BT Effectiveness* in the model and is set at a constant value of 0.5 for the base case. In effect, this means that only half of the component errors that exist in the designs that undergo bench testing are identified with their associated designs being sent for rework.

Those designs that undergo bench testing where no component error is discovered are released to the stock of *Designs Released* to await the next build event. The release rate of these designs is simply the bench test throughput minus the rate of discovering component errors. An additional component of the overall *Release Rate* is the dumping of those designs in the queue for bench testing into the stock of *Designs Released* at the time of a build event. This is done to reflect the standard practice in the client company where design engineers remove their designs from the bench testing queue once a build occurs because they expect more complete and reliable test results from vehicle testing.

The third outflow from the stock of *Designs in Test* is *DIT to RW*. This flow represents the rate at which designs in the stock of *Designs in Test* are sent for rework because they are paired with a design from a coupled component that was found to have an integration error and in the process of coordinating the fix for the integration error, it was determined that the designs from both components would need to be reworked. This type of rework is referred to in the model as external rework and will be discussed in greater detail in the External Rework section of this chapter.

Designs Released

The next set of designs in the main stock and flow chain in the Design Sector of the NPD model is the stock of *Designs Released (DR).* This stock and its associated inflows and outflows are depicted in Figure 10.

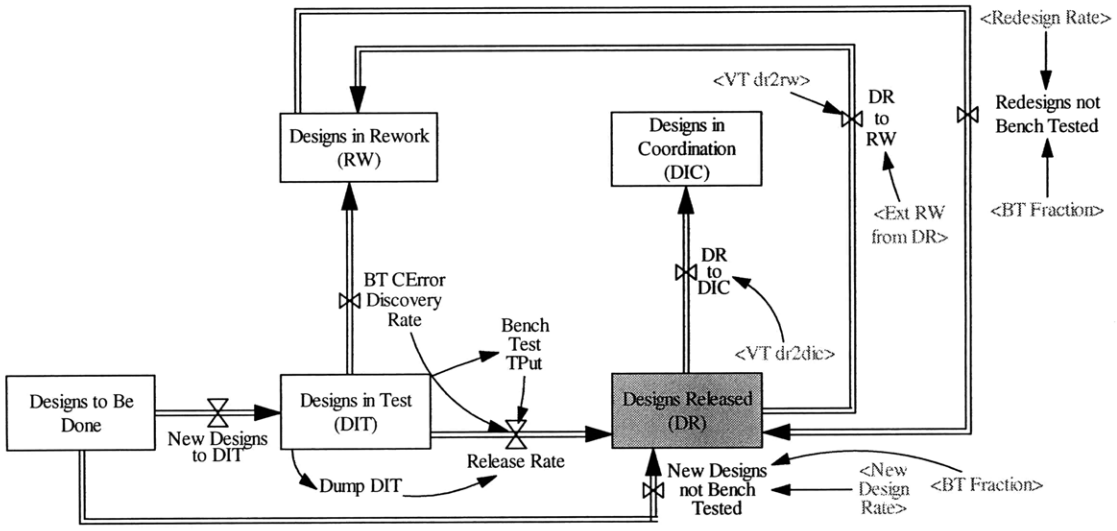


Figure 10 Designs Released (DR)

This stock represents the set of designs that have been released by the designers awaiting the next vehicle build event. As you can see in Figure 10, there are three inflows (*Release Rate*, *New Designs not Bench Tested*, and *Redesigns not Bench Te*sted) and two outflows (*DR to DIC* and *DR to RW*) to this stock.

Two of the inflows - *Release Rate* and *New Designs not Bench Tested* were discussed previously and won't be repeated here. The inflow *Redesigns not Bench Tested* to the stock of *Designs Released* represents the rate at which designs are reworked and subsequently released without being sent for bench testing. This inflow is based on the rate at which designs in rework are completed (*Redesign Rate*) and the variable *BT Fraction* which determines the fraction of

redesigns that are sent for bench testing upon completion. As is the case with new designs, the fraction of redesigns that are sent for bench testing is based in large part on the amount of time remaining until the next build event with the fraction decreasing as the amount of time decreases.

The two outflows from the stock of Designs Released – *DR to DIC* and *DR to RW* – represent the rates at which designs that have been previously released are sent for coordination and/or rework as they are discovered to have errors. The primary means for the discovery of errors in the designs that have been released is the vehicle testing that follows each build event. Vehicle testing is able to identify both component and integration errors in designs and, unlike bench testing, vehicle testing is presumed to be perfect (i.e. all errors are discovered) in the NPD model. As a result of discovering component and/or integration errors, designs are sent either directly to rework (DR to RW) or for coordination first (DR to DIC). A discussion of vehicle testing and the decision on whether to send designs with integration errors for coordination (VT dr2dic) or directly for rework (VT dr2rw) will be discussed in detail in later sections of this chapter.

The second component to the flow of designs from the stock of *Designs Released* to the stock of *Designs in RW* is the external rework generated by coordinating designs with integration errors from a coupled component as discussed earlier.

Designs in Rework

The next set of designs in the main stock and flow chain in the Design Sector of the NPD model is the stock of *Designs in Rework (RW).* This stock accumulates those designs that have been designated for rework and represents the queue of those waiting for redesign. The stock of *Designs in Rework* and its associated inflows and outflows are depicted in Figure 11.

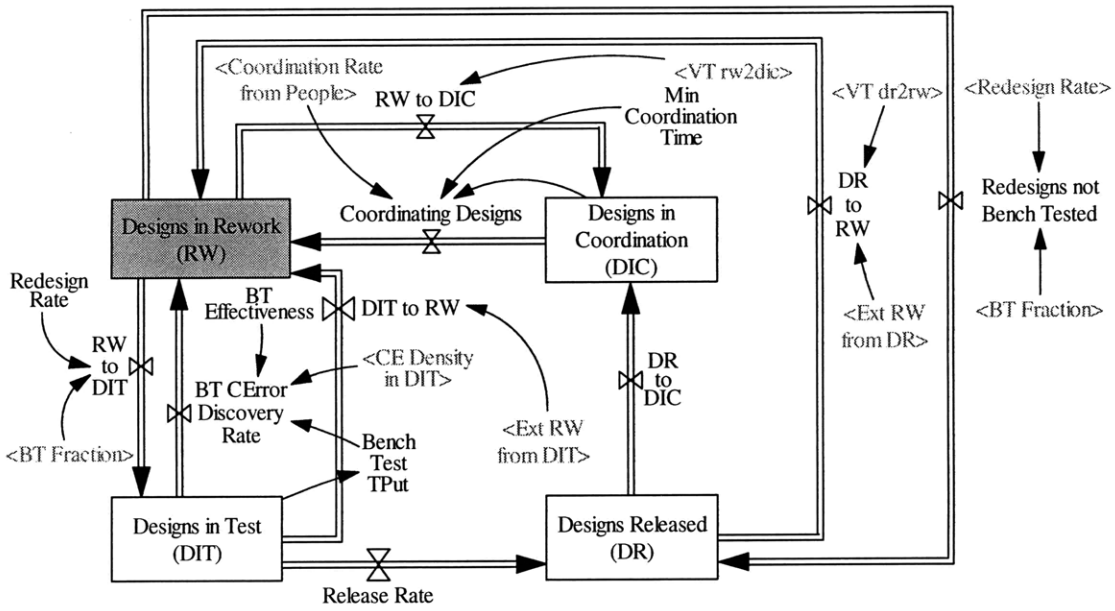


Figure 11 Designs in Rework (RW)

Altogether there are 4 inflows to the stock of *Designs in Rework – BT CError Discovery Rate, DIT to RW, DR to RW* and *Coordinating Designs*. All but the flow of *Coordinating Designs* have been discussed previously and won't be repeated here. There are 3 outflows to the stock of *Designs in Rework (RW) – Redesigns not Bench Tested, RW to DIT,* and *RW to DIC*. The first two have been discussed previously and won't be repeated here.

The inflow of *Coordinating Designs* is simply the inflow of designs from the stock of *Designs in Coordination* as they are coordinated and sent for rework. As defined, an integration error is a mismatch between two designs from coupled components. By convention, only one of the designs (either from component A or B) has the integration error tied to it and as this design is coordinated it is necessarily sent for rework. This process is represented by the flow *Coordinating Designs* in the model. As mentioned earlier, with some likelihood the design from the coupled component

will also be sent for rework. This is referred to as external rework and will be discussed later in this chapter.

The rate of *Coordinating Designs*, much like the new design and redesign rates, is set at the minimum of the coordination rate that is possible based on constrained resources (*Coordination Rate from People*) and the coordination rate that is possible based on the amount of work available (*Designs in Coordination (DIC)*) and the minimum time it takes to coordinate the fixes required for an integration error (*Min Coordination Time*). The minimum coordination time is set at one week for both components and represents the minimum time required to coordinate a design without any resource constraints. The variable *Coordination Rate from People* will be discussed in detail in the Productivity Sector section of this chapter. As in the case of designing and redesigning, the rate of coordinating designs falls to zero at the instant of a build event using the *Build Pulse* variable.

There are 3 outflows to the stock of *Designs in Rework (RW) – Redesigns not Bench Tested, RW to DIT,* and *RW to DIC*. The first two have been discussed previously and won't be repeated here. The third outflow *RW to DIC* is much like the outflow *DR to DIC* from *Designs Released* in that it represents the rate at which designs are found to have an integration error while undergoing vehicle testing and are subsequently sent for coordination. Given that the designs in this case are flowing from *Designs in Rework* they are already identified as requiring rework for having a component error and/or an integration error that has yet to be coordinated.

Designs in Coordination

The last set of designs in the main stock and flow chain in the design sector of the NPD model is the stock of *Designs in Coordination (DIC).* This stock accumulates the designs with an identified integration error that are in the queue for coordination between designers from coupled components. The purpose of this coordination is to identify the source of the integration error and the required rework to fix it. The stock of *Designs in Coordination* and its associated inflows and outflows are depicted in Figure 12.

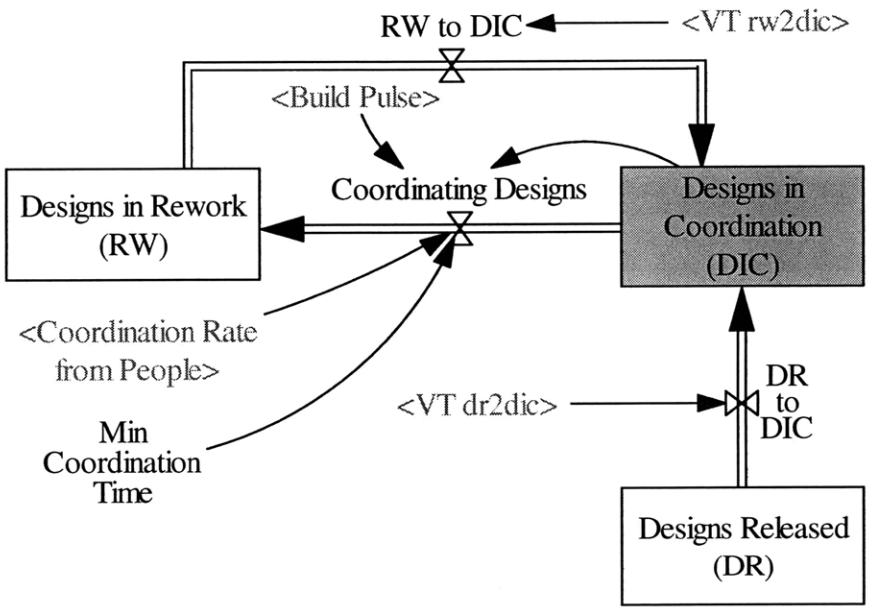


Figure 12 Designs in Coordination (DIC)

There are two inflows to the stock of *Designs in Coordination – DR to DIC* and *RW to DIC.* Both of these flows have already been discussed in detail above. They represent the discovery of designs with integration errors through vehicle testing and the subsequent decision to add these

designs to the queue for coordination. The outflow to the stock is *Coordinating Designs* which was also discussed above. As designs are coordinated they are added to the stock of rework and are added to the queue awaiting redesign.

Error Co-Flow Overview

In addition to the main stock and flow chain which accounts for the set of designs for a given component, the NPD model uses a standard co-flow structure to account for the errors associated with the set of designs. In Figure 6 these co-flows are depicted below the main stock and flow chain. You'll note that there are two co-flow chains used – one for each type of error, component and integration, associated with the designs in the main chain.

Component errors are those errors that arise in a particular component and in the case of the NPD Model to a particular design. Component errors can be of a wide variety but at a basic level can be thought of as a design (or part) that has an error in its "form" or "function" – either it doesn't look the way it should (e.g. in size, shape or style) or it doesn't do its job. Component errors are an inevitable outcome of design work but their likelihood of occurring increases as individual designers are rushed for time and take shortcuts in their design work. By definition, a component error in the NPD model is associated with a single design and each design can have only a single component error at any given time.

Integration errors, on the other hand, are those errors that involve a mismatch between coupled designs. These mismatches, or errors, can be for any number of reasons but on a very basic level they involve two component parts that must either "fit" or "function" together. An integration error occurs when either one of the coupled designs introduces a mismatch by failing to fully

integrate its design work with that of its coupled design. Like component errors, integration errors are an inevitable outcome of design work especially in highly complex and coupled systems. However, the likelihood of introducing integration errors dramatically increases as component designers conduct more and more of their design work in isolation with limited interaction with designers working on coupled designs. The tendency for designers to work in isolation and focus solely on their component design increases significantly as designers are rushed for time and get overwhelmed by their work.

By definition, an integration error requires two designs (or parts) to exist. In the NPD model there are two components, A and B, each with a set of designs to be completed. Because these two components are coupled, integration errors can exist between pairs of designs from the different components. That is: an integration error can exist between a design from component A and a design from component B[9]. In the NPD model, each design from a given component can be paired up with at most a single design from the coupled component and each "design pair" can have at most a single integration error between them at any given time. For accounting purposes, the NPD model assigns an integration error, if one exists, to only one of the two coupled designs.

A design can have a component error, an integration error, or both errors and the process for fixing each of these types of errors is different. As such, it is necessary to track the component errors and the integration errors associated with the designs in the main stock and flow chain using a separate co-flow structure for each. These stock and flow chains are nearly identical to each other. The sections below will discuss the structure of the co-flows beginning with the co-flow for

[9] For the purposes of this model and analysis, integration errors between designs within a single component (e.g. two designs from component A that have a mismatch between them) are not considered.

component errors. In describing the co-flow structure for integration errors, the discussion of the structure that is identical to that of the component errors is limited and focuses instead on those areas where the structure is different.

The overall structure of both of the error co-flows is very similar to that of the main stock and flow chain of designs. With the exception of the stock of *Designs to Be Done*, there is a corresponding stock of errors for each of the stocks of designs in the main stock and flow chain. As mentioned, there is no need to maintain a corresponding stock of errors for the stock of *Designs to Be Done* because there aren't any errors in a design that has not been worked on yet. Additionally, there is a flow between each of the stocks in the co-flow structures of the component and integration errors corresponding to the flows in the main stock and flow chain between the stocks of designs. As is the case for the main stock and flow chain for designs, the co-flows for component and integration errors is subscripted in the model to account for both components A and B.

Component Errors

As mentioned, a component error is one that is isolated to a particular design and can be thought of as an error in "form" or "function". By definition, a component error in the NPD model is associated with a single design and each design can have only a single component error at any given time. A high level view of the co-flow structure that is used to account for the component errors associated with the set of designs is depicted in Figure 13.

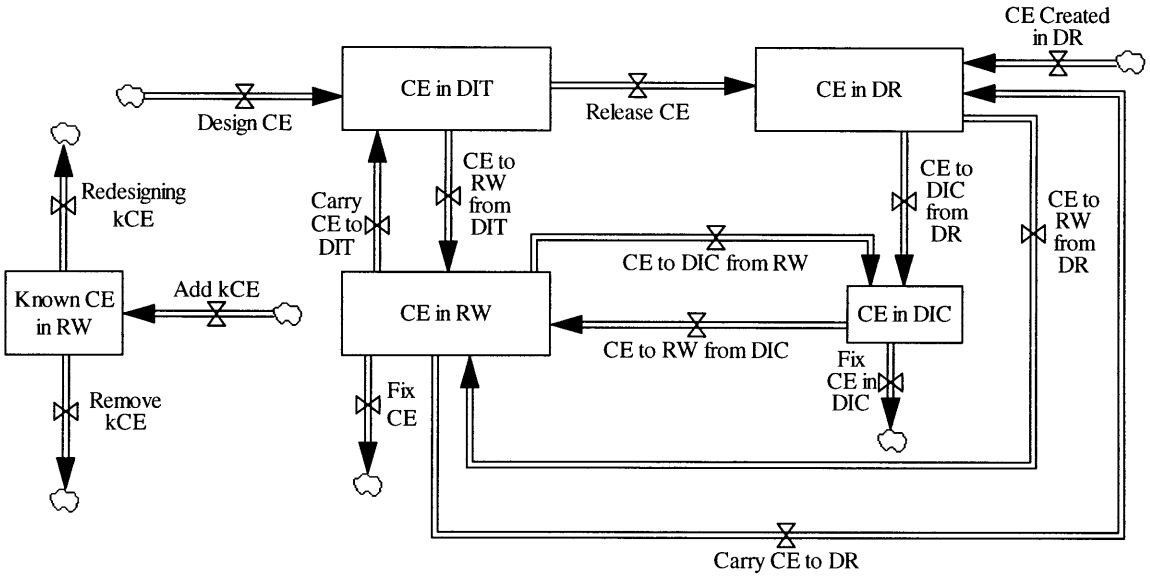


Figure 13 Component Error Co-Flow Structure

Using a co-flow structure, the component errors are tracked along with their associate designs as they progress through the various stages of development. This includes accounting for the generation or creation of component errors as designs are initially worked on and the fixing of component errors as they are redesigned. The NPD model also accounts for component errors that are generated in the process of rework. The co-flow structure for component errors has four main stocks. These include: the stock of component errors associated with the designs in the stock of *Designs in Testing* (*CE in DIT*), the stock of component errors associated with the designs in the stock of *Designs Released* (*CE in DR*), the stock of component errors associated with the designs in the stock of *Designs in Rework* (*CE in RW*) and the stock of component errors associated with the designs in the stock of *Designs in Coordination* (*CE in DIC*). There are inflows and outflows for each of these component error stocks that correspond to the flows between the stocks of designs.

Unlike the main stock and flow chain, which conserves the number of designs at its initial level, the co-flow structure allows errors to be added to and taken out of the system. This is indicated in the depiction of the co-flow structure with arrows coming out of or going into small clouds. For example, in the upper left of Figure 13 the arrow representing the variable *Design CE* emanates from a small cloud indicating that component errors can be added to the system through this flow. On the other hand, the arrow representing the variable *Fix CE* in the bottom center of Figure 13 is pointing into a cloud indicating that component errors can be removed from the system through this flow.

Additionally, the co-flow structure for component errors includes a stock of known component errors in rework (*Known CE in RW*). This stock of component errors does not have a corresponding stock of designs in the main stock and flow chain. Indeed, it is a co-flow structure of sorts for the stock of *CE in RW*. This stock is needed to account for the known component errors in rework. As will be discussed below, not all component errors in the stock of rework will be known. As the model presumes that only known errors can be fixed, it is important to account for the density of known component errors.

We will now consider each of the stocks of component errors and their associated inflows and outflows in greater detail. There is a subsection included for each of the 5 CE stocks: *CE in DIT*, *CE in DR*, *CE in RW*, *CE in DIC,* and *Known CE in RW*. Again the technical documentation for the model is included in Appendix A and Appendix B.

Component Errors in Designs in Test

The first set of component errors in the co-flow structure to be considered is the stock of *CE in DIT*. It represents the stock of component errors associated with the stock of *Designs in Test* (those designs in the queue for bench testing) for a given component. The stock of *CE in DIT* and its associated inflows and outflows are depicted in Figure 14 below.

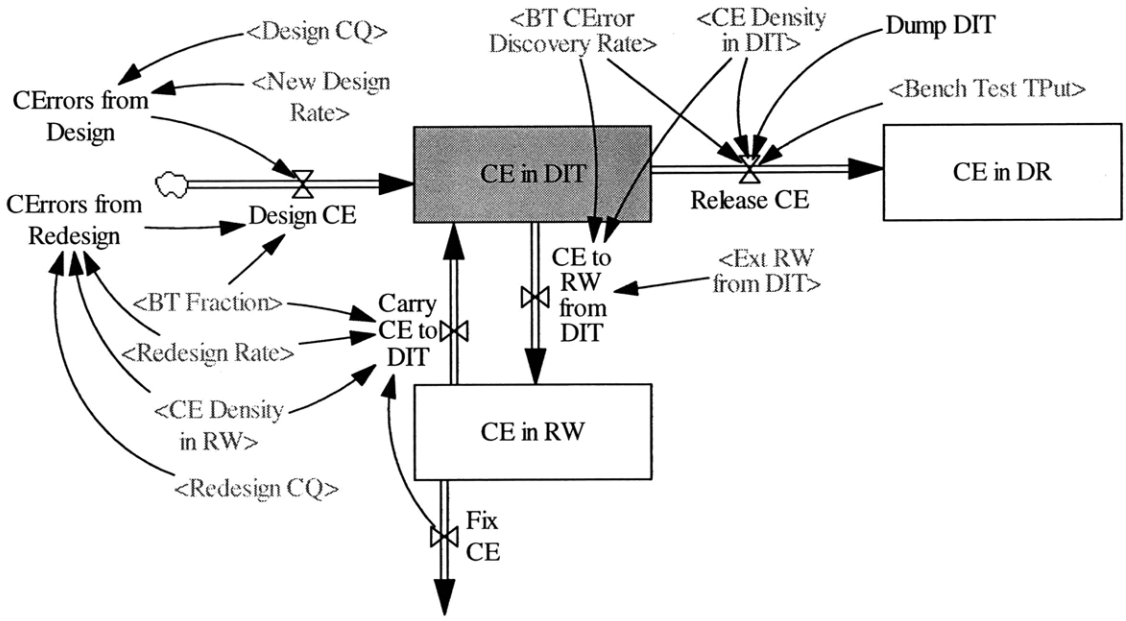


Figure 14 Component Errors in Designs in Test (CE in DIT)

There are two inflows to the stock of *CE in DIT – Design CE* and *Carry CE to DIT* – and two outflows – *Release CE* and *CE to RW from DIT*. The main inflow to the stock of *CE in DIT* is *Design CE*. This variable represents the addition of component errors to the stock of *CE in DIT* as they are created as designs are either initially designed (*CErrors from Design*) or redesigned (*CErrors from Redesign*).

The rate at which component errors are generated in design or redesign is dependent on the rate of design (or redesign) and the associated component quality of design (*Design CQ* and *Redesign CQ* respectively).

The equations for *New Design Rate* and *Redesign rate* have been previously discussed and so won't be repeated here. The *Design CQ* is a variable that represents the component quality of design. The formulation for Design CQ (and Redesign CQ) will be discussed in detail in the Quality section of this chapter. Valued between 0 and 1, Design CQ represents the fraction of designs completed without introducing a component error. Thus the complement (*1- Design CQ*) when multiplied by the *New Design Rate* generates the number of component errors introduced while completing new designs. The formulation for *CErrors from Redesign* is equivalent to the one for new designs except that the term (*1-CE Density in RW*) is added. In the case of redesign, new component errors can only be introduced in those designs in rework that do not have a component error. The NPD model assumes that each design can only have one component error. The variable *CE Density in RW* represents the density of component errors in the stock of *Designs in Rework* thus the complement (*1-CE Density in RW*) is the fraction of designs in rework that do not have a component error and are eligible to have one created in the process of redesign.

Not all of the component errors created in design and redesign are added to the stock of *CE in DIT* because not all initial designs and redesigns are sent for bench testing. Thus, we include *BT Fraction* in the equation for *Design CE* effectively adding a proportionate amount of component errors to the stock of *CE in DIT* to the fraction of designs and redesigns that are sent for bench testing.

The second source of component errors being added to the stock of *CE in DIT* is "carrying over" component errors from the stock of *CE in RW* (those associated with the stock of designs in rework) represented by the variable *Carry CE to DIT* in the model. "Carrying over" an error effectively means that a design with an error is redesigned without fixing it. This can occur in two ways. The first occurs when a design in the stock of rework has an unknown error. Again, the model presumes that only known errors can be fixed and thus a design with an unknown component error will see the component error carried over. The second way that an error is carried over is when a known error is redesigned but not fixed due to poor quality.

As in the case of creating new component errors through redesign, not all component errors that are carried over are added to the stock of *CE in DIT* but rather the amount of carried over CE added to the stock of *CE in DIT* is proportionate to the fraction of redesigns that are sent for bench testing.

The first outflow from the stock of *CE in DIT* to be considered is the releasing of component errors to the stock of *CE in DR* (*Release CE* in the model). This occurs in two ways. The first is when designs in the stock of *Designs in Test* that have component errors are bench tested but the component error is not discovered and thus the designs and their associated errors are released. The second way in which component errors are released to the stock of *CE in DR* is in conjunction with dumping the designs in the bench test queue upon a build event. This is calculated simply by multiplying the rate of dumping designs (*Dump DIT*) by the density of component errors in the stock of *Designs in Test* (*CE Density in DIT*).

The second outflow from the stock of *CE in DIT* is the rate at which component errors are sent to the stock of *CE in RW* as their associated designs are sent for rework. This rate is represented by the variable *CE to RW from DIT* in the model.

There are two processes by which component errors and their associated designs are sent for rework. The first is when designs are discovered to have component errors in bench testing (*BT CError Discovery Rate*) which was discussed earlier. The result is that a component error is moved to the stock of *CE in RW* from the stock of *CE in DIT* for every design that is discovered to have a component error in bench testing and is subsequently sent for rework. The second way that a component error is sent to the stock of *CE in RW* is when its associated design is sent to rework as a result of coordinating an integration error in a design from a coupled component (*Ext RW from DIT*). The formulation for Ext RW from DIT will be discussed in the External RW section of this chapter. In this case, the number of component errors that are sent along with the designs is proportionate to the fraction of designs in *DIT* that have component errors (*CE Density in DIT*).

Component Errors in Designs Released

The next set of component errors in the co-flow structure is the stock of *CE in DR*. It represents the stock of component errors associated with the stock of *Designs Released* (those designs released awaiting the next build event) for a given component. The stock of *CE in DR* and its associated inflows and outflows are depicted in Figure 15.

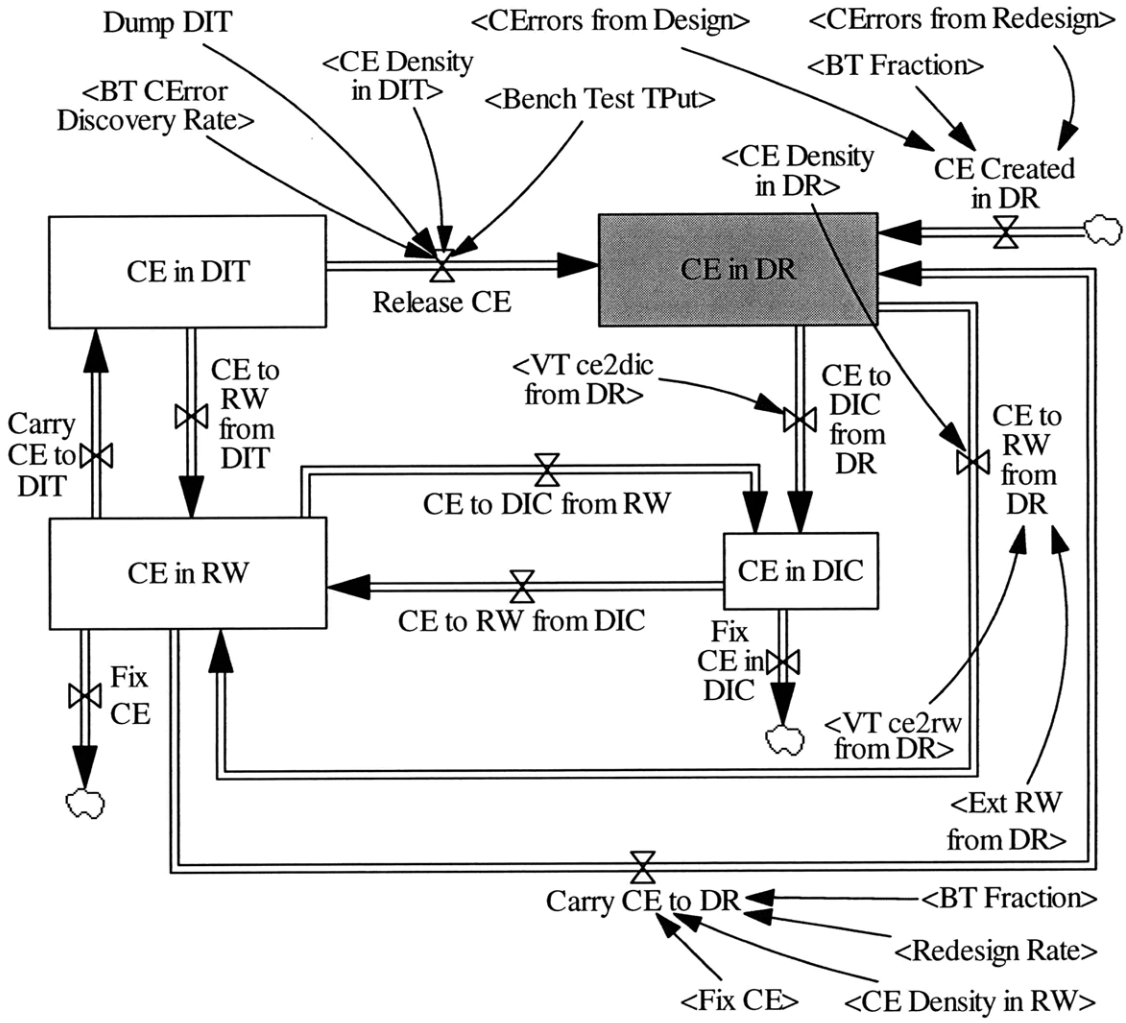


Figure 15 Component Errors in Designs Released (CE in DR)

There are three inflows to the stock of *CE in DR – Release CE, CE Created in DR* and *Carry CE to DR* – and two outflows – *CE to DIC from DR* and *CE to RW from DR.*

The inflow *Release CE* was discussed earlier. It represents the flow of component errors that are released along with their associated designs when component errors go undiscovered in bench testing and when designs are "dumped" from the bench test queue at the time of a build. The

inflow *CE Created in DR* is analogous to the inflow of *Design CE* to the stock of *CE in DIT* except the component errors in this flow are those associated with the designs that are not sent for bench testing upon initial design or redesign. The third inflow to *CE in DR* is *Carry CE to DR* and represents the "carrying over" of component errors from the stock of *CE in RW*.

The primary reason that component errors flow out of the stock of *CE in DR* is the discovery of these errors through vehicle testing. When component errors are discovered in vehicle testing, the errors and their associated designs are either sent directly for redesign (*CE to RW from DR*) or in some cases they are sent for coordination prior to redesign (*CE to DIC from DR*) if they are found to have both a component error and an integration error. The component errors sent to the stock of *CE in RW* also include those component errors associated with the designs from the stock of *Designs Released* that are identified for rework based on the coordination of an integration error in a coupled design. Vehicle testing and the variables *VT ce2rw from DR* and *VT ce2dic from DR* will be discussed in the Vehicle Testing section of this chapter.

Component Errors in Designs in Rework

The next set of component errors in the co-flow structure is the stock of component errors that are associated with the designs in the queue for redesign (*Designs in Rework*). The stock of *CE in RW* and its associated flows is depicted in Figure 16.

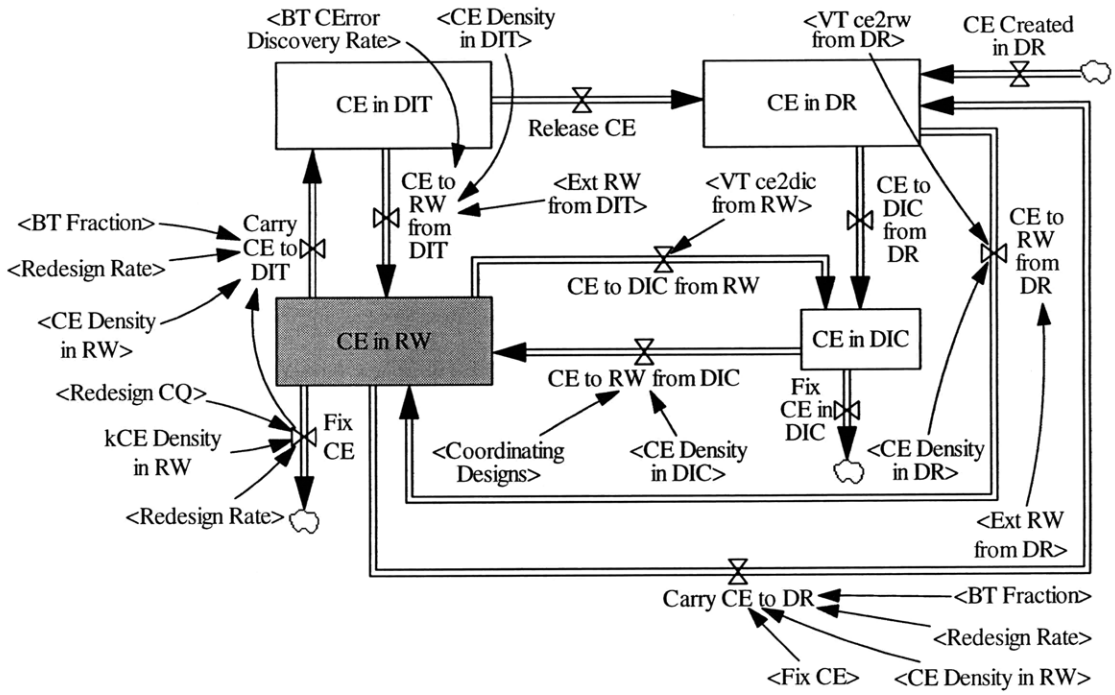


Figure 16 Component Errors in Designs Rework (CE in RW)

There are three inflows to the stock of *CE in RW – CE to RW from DIT, CE to RW from DR* and *CE to RW from DIC* – and four outflows – *Fix CE, CE to DIC from RW*, C*arry CE to DIT,* and *Carry CE to DR.* The primary inflow of component errors to the stock of *CE in RW* is the result of bench testing and is represented by the flow *CE to RW from DIT* in the model. The formulation for this flow was discussed previously. You'll remember that this flow also includes those component errors associated with designs identified for external rework. The second inflow of component errors, *CE to RW from DR*, was discussed in the previous section and includes component errors discovered in vehicle testing as well as component errors associated with released designs that are identified for external rework. The third inflow to the stock of *CE in RW* comes from component errors associated with designs that are coordinated prior to being sent for rework and is represented by the flow *CE to RW from DIC* in the model.

The primary outflow of component errors from the stock of *CE in RW* is the result of fixing component errors in the process of redesign. This process is captured by the flow *Fix CE* in the model. Prior to discussing in detail the formulation for fixing component errors, it is important to note that those component errors that are not fixed while their associated designs are redesigned are carried over. This process results in the outflow of component errors from the stock of *CE in RW* to either the stock of *CE in DIT* (if the associated design is sent for bench testing after redesign) or the stock of *CE in DR* (if the associated design is not bench tested). The formulation for these two flows was discussed previously. The third outflow of component errors from the stock of *CE in RW* is the result of discovering integration errors through vehicle testing and the subsequent decision to send the associated designs for coordination. This process is represented by the flow *CE to DIC from RW* in the model.

The variables Redesign Rate and Redesign CQ have been discussed previously. The variable *kCE Density in RW* represents the fraction of designs in the stock of Designs in RW that have known component errors. By multiplying this fraction by the *Redesign Rate*, we can calculate the rate at which designs with known component errors are being redesigned. By further multiplying by the component quality of redesign (*Redesign CQ*) we can calculate the rate at which component errors are being fixed.

The formulation of *kCE Density in RW* is analogous to all of the other component error densities in the co-flow structure. However, this formulation requires us to distinguish the known component errors - those discovered through bench testing and/or vehicle testing – from the unknown component errors associated with the set of designs in rework. Since the stock of *CE in RW* includes both known and unknown component errors we need to maintain a separate co-flow

for the known component errors associated with the stock of *Designs in Rework.* This separate co-flow structure will be discussed in the next section.

Known Component Errors in Designs in Rework

In order to be able to distinguish between the known and unknown component errors associated with the stock of Designs in Rework, we need to maintain a separate co-flow structure for the known component errors. This is done with the stock of *Known CE in RW* and its associated inflows and outflows, a picture of which is included in Figure 17.

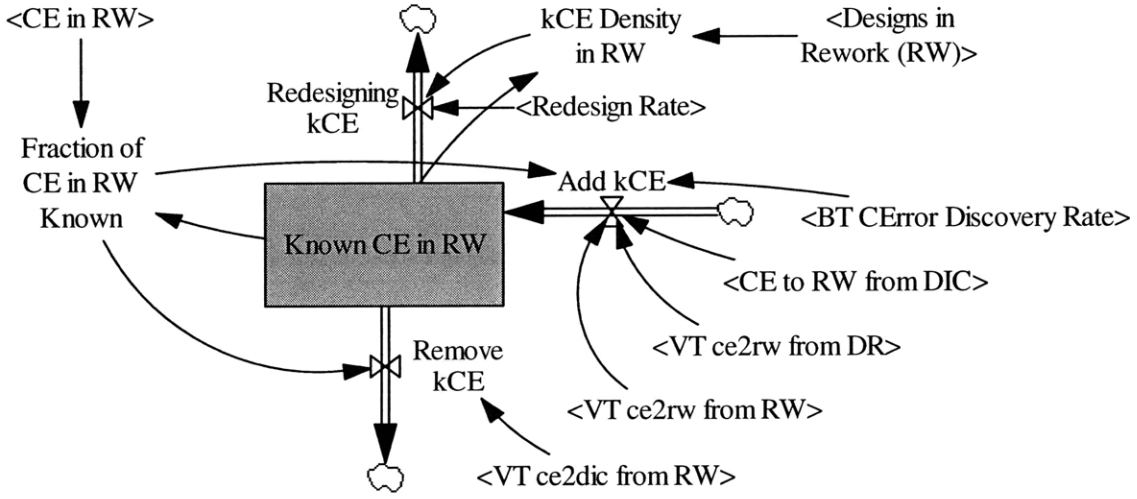


Figure 17 Known Component Errors in Designs in Rework (Known CE in RW)

There is one inflow to the stock of *Known CE in RW – Add kCE* which represents the addition of component errors to the stock of known component errors in rework upon discovery either through bench testing or vehicle testing. There are two outflows to the stock of *Known CE in RW.* The main outflow, *Redesigning kCE*, as the name implies represents removing component errors

from the stock of known CE as designs in rework are redesigned. The second outflow, *Remove kCE*, is the result of discovering an integration error in a design with a known component error and sending it for coordination.

Adding component errors to the stock of *Known CE in RW* is done as component errors are discovered either in bench testing or vehicle testing and as designs with known component errors and integration errors are coordinated.

BT CError Discovery Rate represents the rate at which component errors are discovered in bench testing and their associated designs are sent for rework; thus each component error discovered in this manner is added to the stock of *Known CE in RW.* The variable *CE to RW from DIC* represents the component errors that are sent to the stock of *CE in RW* as designs are coordinated. The NPD model assumes perfect vehicle testing which means that all component and/or integration errors are discovered in vehicle testing. It also assumes that a design that has both types of error will have both errors discovered at the same time. As such, for any design that has a known integration error – the reason for coordination – and a component error, we can assume that the component error is also known. Therefore, every component error that is moved to the stock of *CE in RW* as its associated design is coordinated (via *CE to RW from DIC*) can be added to the stock of *Known CE in RW. VT ce2rw from DR* represents those component errors associated with released designs that are discovered through vehicle testing and sent for rework along with their associated design.

The last way that the model adds known component errors to the stock of *Known CE in RW* represents the discovery of previously unknown component errors in the designs in the stock of

rework. The way that a design makes its way into the stock of *Designs in RW* with an unknown component error is through the process of external rework. As designs are designated for rework as a result of coordinating integration errors in designs from a coupled component, they take with them their associated component errors. These component errors may or may not be known. One can imagine a design with a component error that was released without bench testing and thus the component error is unknown. If a corresponding design from a coupled component has an integration error that is discovered, the resulting coordination may require both designs to be reworked. Thus the released design with the unknown component error would be sent for rework and its component error would remain unknown until it is discovered in subsequent testing.

The main outflow from the stock of *Known CE in RW* is *Redesigning kCE.* As the name implies, this occurs when designs with known component errors are redesigned and thus leave the stock of Designs in Rework along with their associated errors. The second outflow from the stock of *Known CE in RW* is *Remove kCE* and represents the rate at which known component errors are sent along with their designs for coordination upon discovery of an integration error in vehicle testing. The variable *VT ce2dic from RW* included in the equation for *Remove kCE* below represents the rate at which designs in rework with both types of errors – component and integration – are discovered in vehicle testing and subsequently sent for coordination. However, only some of the component errors discovered in the designs in rework were previously known and thus the rate at which these component errors are discovered and sent along with their designs sent for coordination must be multiplied by the fraction of component errors in rework that are known (*Fraction of CE in RW Known*) in order to determine how many should be removed from the stock of *Known CE in RW.* The formulation for *VT ce2dic from RW* will be discussed in the Vehicle Testing section of this chapter.

Component Errors in Designs in Coordination

The last stock in the component error co-flow structure is that of component errors in designs in coordination (CE in DIC). The stock and its associated flows are depicted in Figure 18.

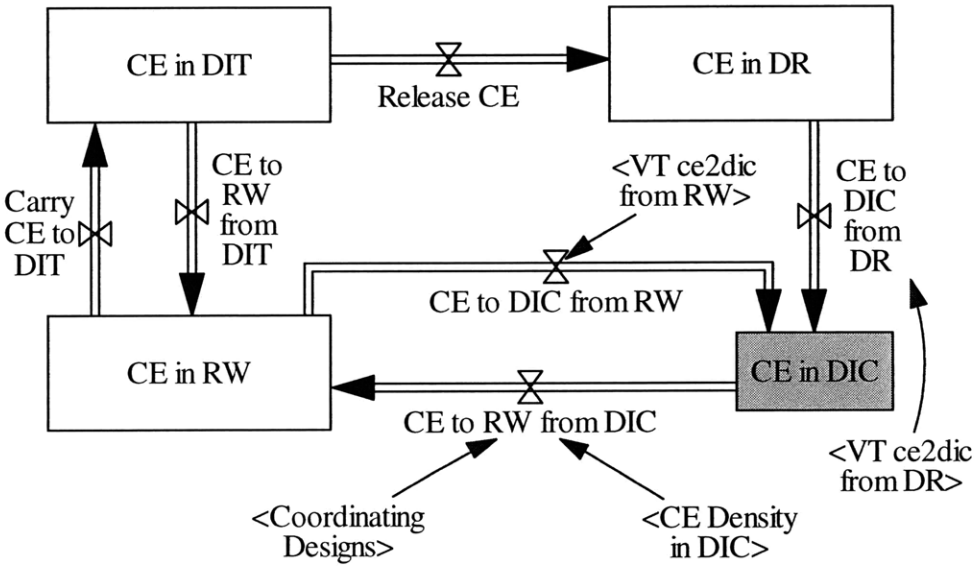


Figure 18 Component Errors in Designs in Coordination (CE in DIC)

The stock of component errors in designs in coordination has two main inflows – *CE to DIC from DR* and *CE to DIC from RW*. These represent the discovery of integration errors through vehicle testing in designs that also have component errors. The only outflow from the stock of *CE in DIC* is the flow of component errors to the stock of rework as designs with both types of errors are coordinated represented by the variable *CE to RW from DIC* in the model.

Integration Errors

As mentioned, integration errors are those errors that involve a mismatch between coupled designs that must either "fit" or "function" together. An integration error occurs when either one of the coupled designs introduces a mismatch due to poor integration quality. In the model, each design from a given component can be paired up with at most a single design from the coupled component and each "design pair" can have at most a single integration error between them at any given time. The NPD model assigns an integration error, if one exists, to only one of the two coupled designs. The formulation for how this accounting is done will be discussed below.

A design can have a component error, an integration error, or both errors and the processes for identifying and fixing each of these types of errors are different. As such it is necessary to track the integration errors associated with the designs in the main stock and flow chain using a separate co-flow structure from the component error co-flow just discussed. A high level view of the co-flow structure that is used to account for the integration errors is depicted in Figure 19.

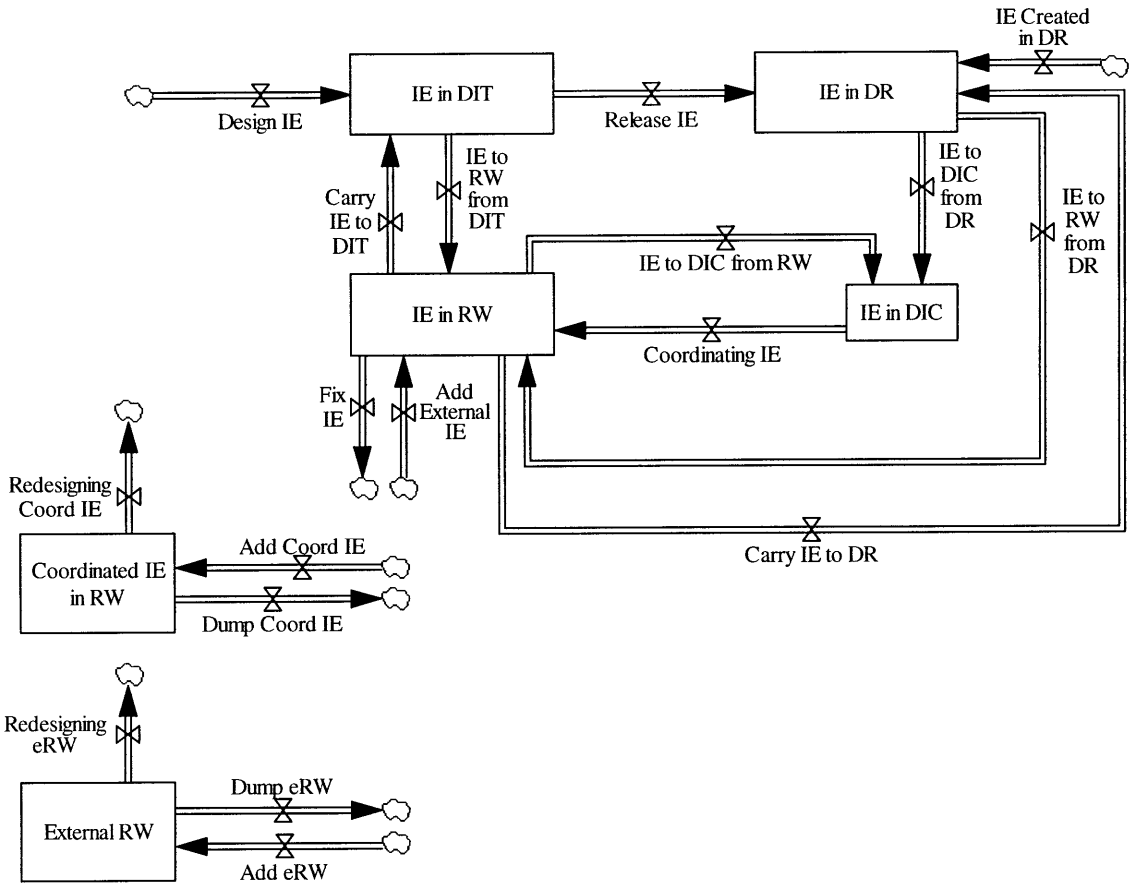


Figure 19 Integration Error Co-Flow Structure

You should note that the co-flow structure used to account for the integration errors associated with the designs in the main stock and flow chain is almost identical to the co-flow structure used for components errors as depicted in Figure 13. There are 4 stocks of integration errors (IE) that correspond to the stocks in the main chain of the designs in progress. The stock of *Coordinated IE* is analogous to the stock of *Known CE in RW* from the component error co-flow structure. Like known component errors, only coordinated integration errors can be fixed through redesign. The additional stock of *External RW* is needed to account for the additional errors and rework (i.e.

Phantom Work) that are created when components iterating at different speeds don't get all of the necessary rework done prior to a build event.

We will now consider each of the stocks of integration errors and their associated inflows and outflows in greater detail. There is a subsection included for each of the 6 IE stocks: *IE in DIT*, *IE in DR*, *IE in RW*, *IE in DIC*, *Coordinated IE in RW*, and *External RW*. Again the technical documentation for the model is included in the appendices.

Integration Errors in Designs in Test

Analogous to the stock of CE in DIT, the stock of integration errors in the designs in test (*IE in DIT*) represents the stock of integration errors associated with the stock of *Designs in Test* (those designs in the queue for bench testing) for a given component. The stock of *IE in DIT* and its associated inflows and outflows are depicted in Figure 20.

There are two inflows to the stock of *IE in DIT – Design IE* and *Carry IE to DIT* – and two outflows – *Release IE* and *IE to RW from DIT*. While the inflows and outflows to the stock of IE in DIT are analogous to those for CE in DIT, their formulations are different. Each will be discussed below.

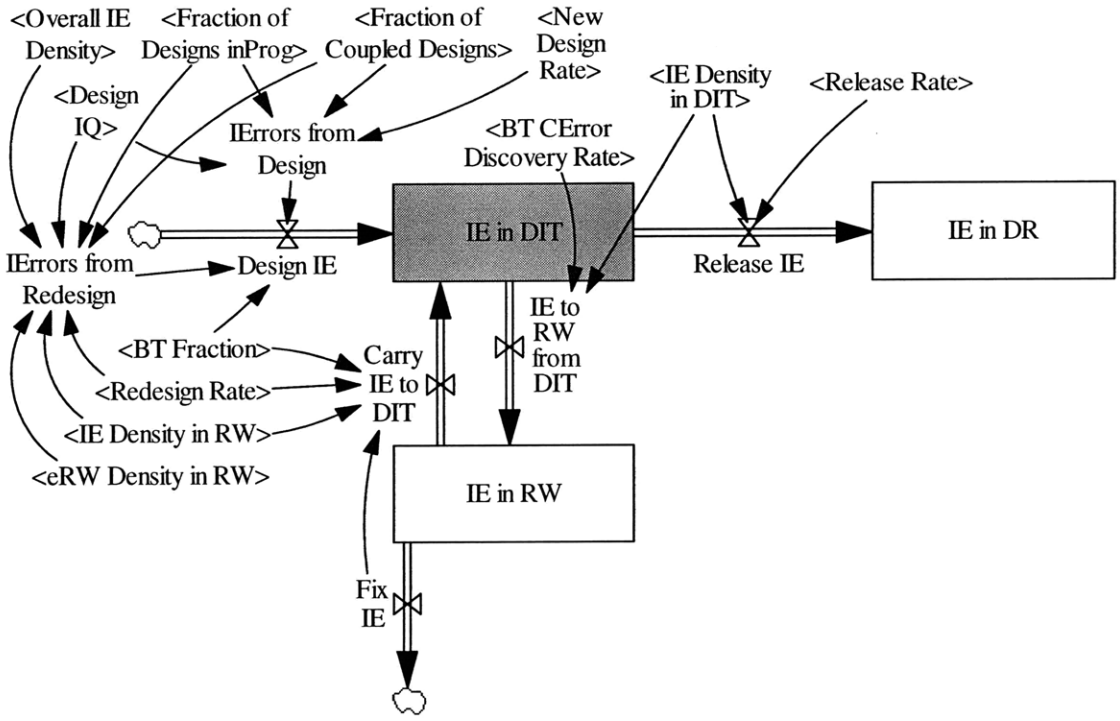


Figure 20 Integration Errors in Designs in Test (IE in DIT)

The inflow *Design IE* is exactly analogous to that of *Design CE* and is made up of integration errors from design and redesign that are associated with the designs that are sent for bench testing. The driving processes behind the generation of integration errors in design and redesign are the New Design Rate and the Redesign Rate respectively, and a term (*1-Design IQ*) used to account for the level of integration quality used during design and redesign activity. The formulation for the design and redesign rates has been discussed and the formulation of integration quality is completely analogous to that of component quality. You'll note that there are separate equations required for the generation of integration errors for components A and B from both design and redesign. While the equations are similar to those in the component error co-flow there are important differences. These will be discussed below.

In determining the rate of generating integration errors from designing, we use the *Fraction of Coupled Designs* to account for the probability that a given design being completed is in fact coupled with a design from the other component and thus able to generate an integration error. The *Fraction of Coupled Designs* for each component is a fixed number between 0 and 1 that represents the fraction of designs from a given component that are coupled with a design from the other component.

Additionally, because an integration error by definition requires two coupled designs to exist we use the fraction of designs in progress (*Fraction of Designs inProg*) of the OTHER component to account for the probability that a given design is coupled with a design from the other component that has at least been initially designed. This means that of two coupled designs, the first one to be completed CANNOT, by definition and formulation, generate an integration error. It is not until the second of the two coupled designs is completed that an integration error can be generated.

The formulation for integration errors from redesigning for components A and B is similar to that from designing above. Each of the formulations includes the rate of redesign, a term to account for the integration quality, the fraction of coupled designs, and the fraction of designs from the coupled component that are in progress. Analogous to the equation for component errors from redesign, there is a term included (*1-IE Density*) to account for the fact that there can only be one integration error per design and some of the designs that are being redesigned might already have an integration error.

A new term that is included in the formulation of integration errors from redesign is (*1-eRW Density in RW*). This term is used to prevent the generation of an integration error in the redesign of a design that is in rework due external rework. You'll recall that external rework, by definition, means that the given design is coupled with a design from the other component that already has an integration error attached to it. Therefore, a new integration error cannot be generated for the given design.

The inflow *Carry IE to DIT* is the rate at which integration errors are "carried over" to *IE in DIT*. This is the rate at which designs with existing integration errors are redesigned where the integration errors are not fixed. The formulation is completely analogous to the corresponding inflow for component errors which was discussed previously.

The outflow *Release IE* is the rate at which integration errors are released to the stock of integration errors in designs released (*IE in DR*). This occurs as designs with integration errors are released after bench testing or the designs are released without bench testing upon a build event (*Dump DIT*). The formulation is straightforward in that it assumes that integration errors will be released along with designs in proportion to the density of integration errors in the stock of designs in test (*IE Density in DIT*).

The outflow *IE to RW from DIT* is the rate at which integration errors in designs in testing (*IE in DIT*) are moved to the stock of integration errors in rework (*IE in RW*). This occurs when designs with integration errors also have component errors that are discovered in bench testing and are moved to rework. Like releasing integration errors, the formulation assumes that integration errors will be sent to rework along with designs discovered to have component errors through

bench testing (*BT CError Discovery Rate*) in proportion to the density of integration errors in the stock of designs in test (*IE Density in DIT*).

Integration Errors in Designs Released

Those integration errors associated with the set of designs released awaiting the next build event are represented by the stock of integration errors in designs released (IE in DR) which is depicted along with its inflows and outflows in Figure 21. The structure is analogous to that of the stock of CE in DR discussed previously.

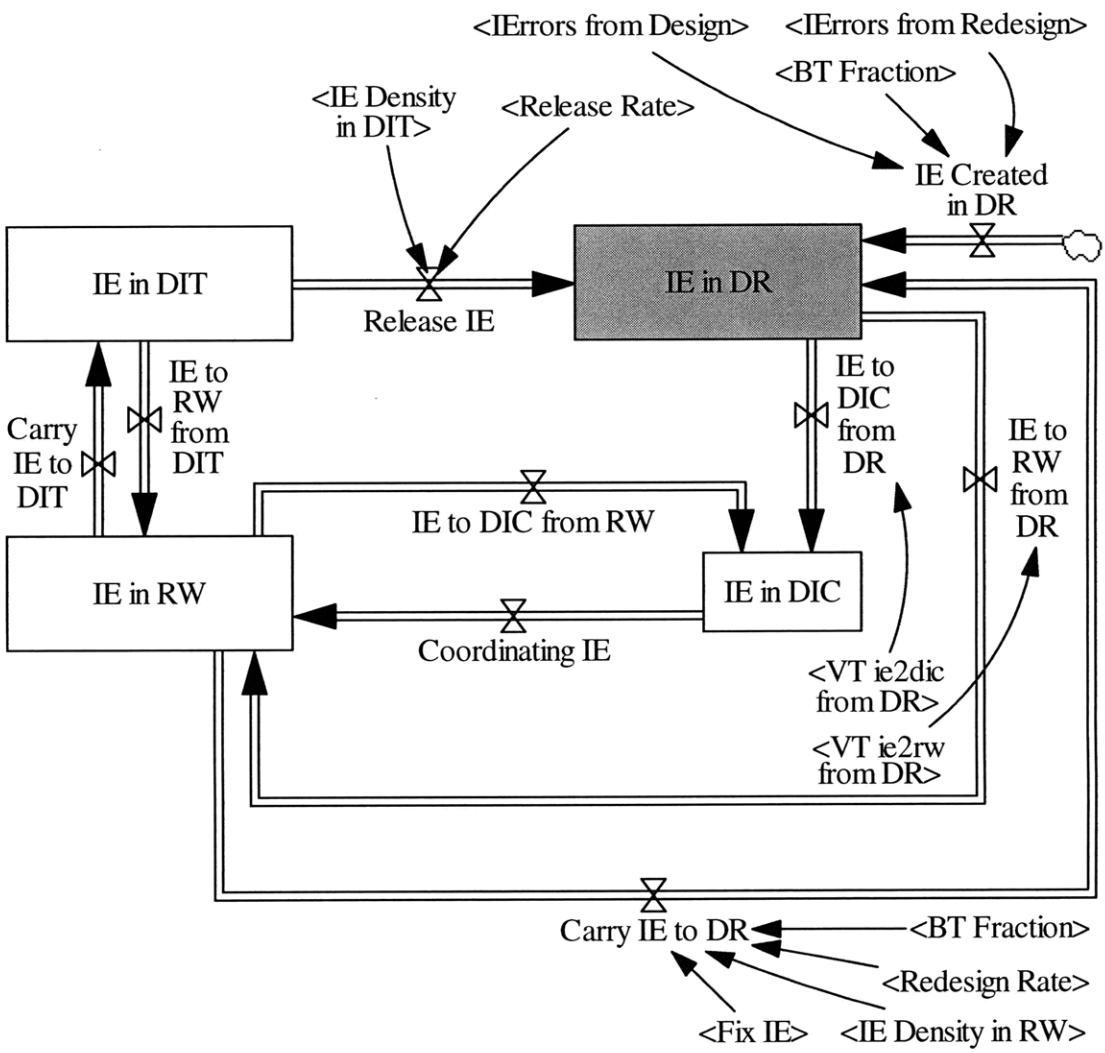


Figure 21 Integration Errors in Designs Released (IE in DR)

The first of three inflows for *IE in DR*, *Release IE* was just discussed. The second inflow, *IE Created in DR*, is analogous to the corresponding equation from the component error co-flow. It represents the creation of integration errors through design and redesign in those designs that are not sent for bench testing. The third inflow to *IE in DR* is represented by the variable *Carry IE to DR* which again is analogous to the corresponding flow in the component co-flow structure.

There are two outflows to the stock of *IE in DR – IE to DIC from DR* and *IE to RW from DR.* Both of these outflows are analogous to the corresponding flows from the component co-flow and their formulations are analogous as well. They represent the rate at which integration errors are discovered in vehicle testing and their corresponding designs are sent for either coordination or directly to rework. The formulation for *VT ie2dic from DR* and *VT ie2rw from DR* will be discussed in the Vehicle Testing section of this chapter. You'll recall that designs that are found to have both an integration error and a component error may be sent directly for rework without coordination based on the time available until the next build and available resources. The formulation for this decision process will be discussed as part of the Coordination Fraction section of this chapter.

Integration Errors in Designs in Coordination

The next set of integration errors to be considered are those that are associated with the set of designs in coordination and are represented by the stock of *IE in DIC.* This stock of integration errors and its associated inflows and outflow is depicted in Figure 22.

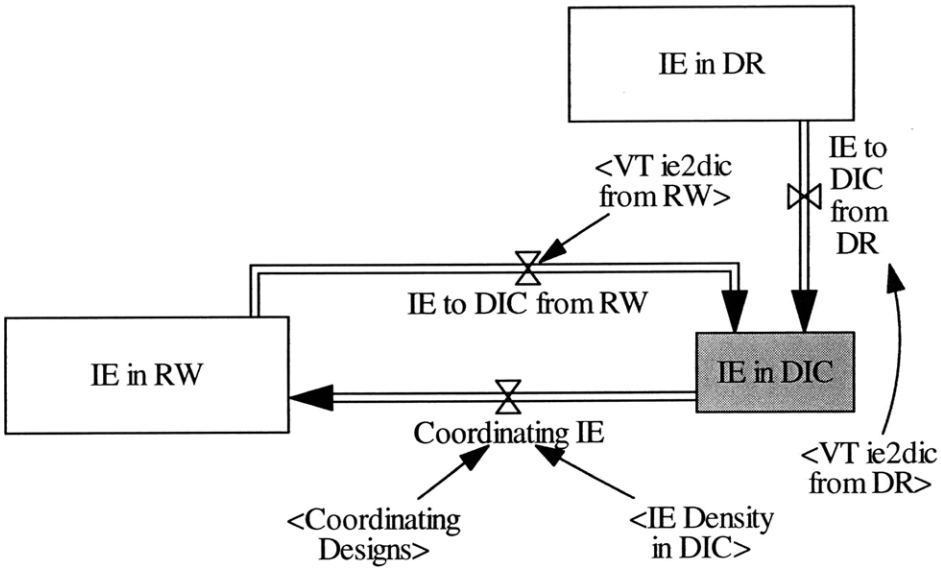


Figure 22 Integration Errors in Designs in Coordination (IE in DIC)

There are two inflows to the stock of *IE in DIC – IE to DIC from DR* and *IE to DIC from RW* – the first of which was discussed in the previous section. There is one outflow to the stock, *Coordinating IE*, which sends integration errors along with their associated designs as they are coordinated.

As mentioned, the formulation for *IE to DIC from DR* has already been discussed. The second inflow, *IE to DIC from RW*, is similar in that it too represents the inflow of integration errors to the stock of *IE in DIC* based on the discovery of integration errors in designs through vehicle testing and the subsequent decision to send these designs for coordination. The formulation for *VT ie2dic from RW* will be discussed in detail in the Vehicle Testing section of this chapter.

The only outflow to the stock of *IE in DIC* is *Coordinating IE* which represents the transfer of integration errors to the stock of *IE in RW* as their associated designs are coordinated. The

formulation ensures that integration errors are transferred along with the coordinated designs in proportion to the fraction of designs in coordination (DIC) that have integration errors (*IE Density in DIC*)[10].

It is important to remember that the process of coordinating designs, while required to fix an integration error, does not in and of itself fix the error. Coordinating designs simply involves two designers, one from each coupled component involved in the integration error, determining the root cause of the integration error and the necessary rework of one or both of the designs. By convention in the NPD model, the component design to which the integration error is attached will always require rework. With some fixed probability the coupled design will also require rework. It is not until the required rework (whether of one or both designs) is complete that the integration error is fixed. The formulation for ultimately fixing integration errors will be discussed in the next section.

Integration Errors in Designs in Rework

The final set of integration errors in the co-flow structure is the stock of integration errors that are associated with the designs in the queue for redesign (*Designs in Rework*). The stock of *IE in RW* and its associated flows is depicted in Figure 23.

There are four inflows to the stock of *IE in RW – IE to RW from DIT, IE to RW from DR, Coordinating IE*, and *Add External IE*. Each of these inflows, except the last (*Add External IE*) has a corresponding flow in the component error co-flow structure. There are also four outflows to

[10] Nominally, every design that is in coordination must have an integration error and this density should remain equal to 1. Indeed, in the simulation it does. The inclusion of this term in the formulation is done for both consistency and robustness.

this stock – *Fix IE, IE to DIC from RW*, C*arry IE to DIT,* and *Carry IE to DR* – each of which has a corresponding flow in the co-flow structure for component errors.

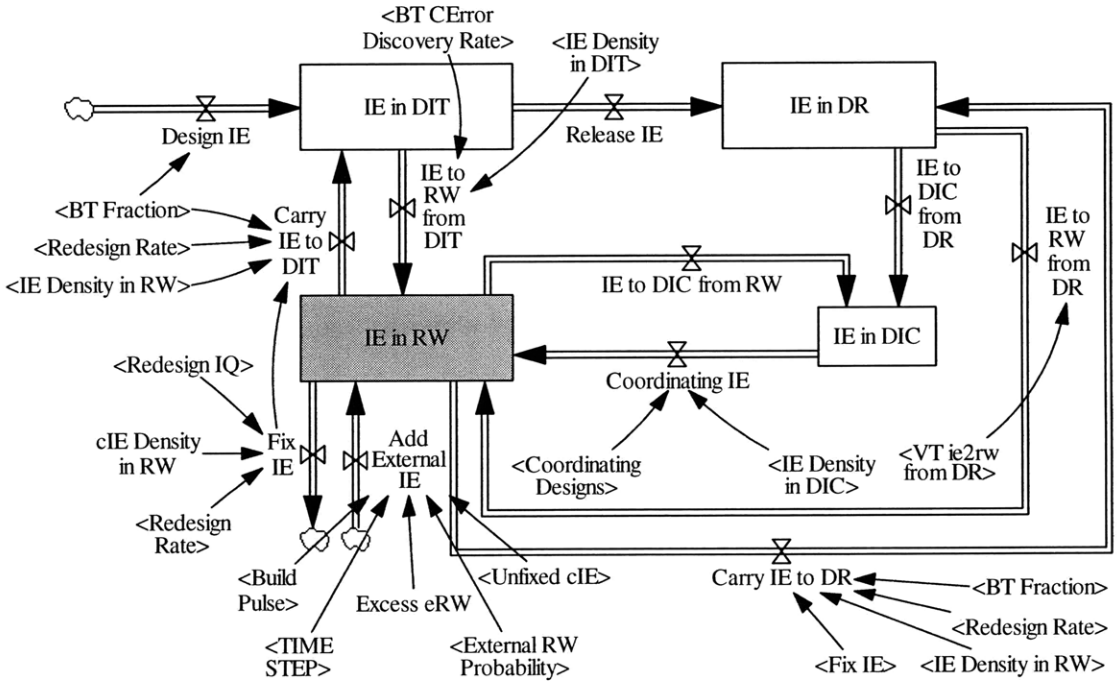


Figure 23 Integration Errors in Designs in Rework (IE in RW)

Of the four inflows to the stock of *IE in RW*, each flow except the last (*Add External IE*) is analogous to a corresponding flow in the component error co-flow. A discussion of each of these has been provided above. On the other hand, the last inflow – *Add External IE* – does not have an analogous flow in the component error co-flow and requires a fair bit of explanation.

The last flow, *Add External IE*, is the rate at which integration errors are added to the stock of *IE in RW* and represents the transfer of an integration error from one design in a coupled design pair to the other. This occurs when the coordination of an integration error requires both of the coupled designs to conduct rework and the design with the integration error attached to it completes its redesign and fixes its share of the of the integration error but the coupled design does not complete its rework. You'll recall that an integration error is not considered fixed until both designs have completed their rework. Therefore, if the design with the attached integration error completes its required rework the integration error will still exist if the coupled component design has not yet completed its rework - in which case, the model transfers the error to the incomplete design awaiting rework. For accounting purposes in the model, this transfer of integration errors between coupled designs occurs at the time of vehicle builds.

The variable *Excess eRW* represents the number of designs in external rework that are over and above that which would be expected based on the number designs the coupled component has left in rework with integration errors that have been coordinated. The indicated external rework (*Indicated eRW*) is the amount of external rework a given design should expect to have based on the number of coordinated designs that its coupled component still has in rework (*Coordinated IE in RW*) and the probability of a coordinated design requiring rework from both coupled designs (*External RW Probability*). In the NPD model, the *External RW Probability* is set at 0.5 meaning that half of the integration errors that are coordinated with require both designs from the coupled "design pair" to conduct rework to fix the integration error. The formulation for the stocks of *External RW* and *Coordinated IE in RW* and their associated flows will be discussed in more detail below.

A component that has more external rework than what is indicated has not completed its share of required redesigns based on previously coordinated integration errors. Therefore, the integration errors associated with the excess external rework will not have been fixed. In this case it is appropriate to attach these integration errors to the slower component (i.e. the component with excess external rework) and designate them as new uncoordinated integration errors in the stock of IE in RW. However, not all of the excess external rework will be transferred to the slower component as new integration errors because the faster component (the one that completed its share of required redesigns) may not have fixed the integration errors attached to it during redesign. Remember that the integration quality of redesign is not perfect and some integration errors, even those that have been coordinated, will not be fixed in redesign and thus will be carried over. In the event this happens, the integration error is still attached to the original component and need not be transferred to the slower component. These unfixed errors are referred to as *Unfixed cIE* because in this case we are only concerned about coordinated integration errors, which drive external rework, that weren't fixed during redesign. The formulation of *Unfixed cIE* will be discussed below. Not all of these unfixed errors are associated with external rework. Indeed we would expect only half of the unfixed errors to be associated with external rework based on an *External RW Probability* of 0.5. Therefore, we subtract half of the unfixed errors from the excess amount of external rework before adding them to the stock of *IE in RW*. Again the transfer of these integration errors between coupled components happens at the time of each build.

It should be noted that these "new" integration errors are added to the stock of *IE in RW* but they are not added to the stock of *Coordinated IE in RW* which will be discussed below. Because the NPD model assumes that integration errors must be coordinated in order to be fixed, by not adding these integration errors to the stock of *Coordinated IE in RW* effectively means that they

can not be fixed unless and until they are discovered in subsequent vehicle testing and coordinated again.

External Rework

The stock of external rework represents the set of designs in rework that require redesign because of an integration error that exists between the given designs and a corresponding set of designs from a coupled component. The rework is classified as external because it is identified in the process of coordinating the required rework to fix an integration error that is attached to a design from a coupled component and not the given component. The stock of External RW and its associated flows is depicted in Figure 24.

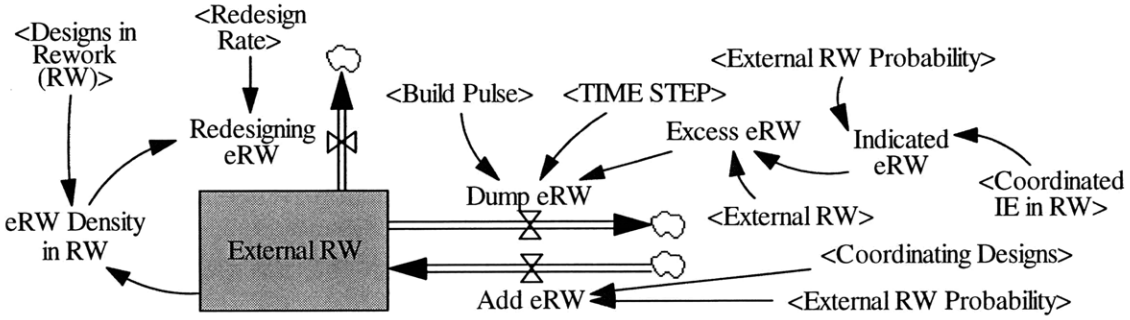


Figure 24 External Rework (External RW)

There is one flow into the stock of *External RW – Add eRW* and two outflows – *Redesigning eRW* and *Dump eRW*. The inflow, *Add eRW*, represents the addition of external rework to the stock of *External RW* for a given component as designs from the OTHER (i.e. coupled) component

that have integrations errors are coordinated. The formulation ensures that designs are added to the stock of *External RW* as the coupled designs from the other component are coordinated (*Coordinating Designs*) with the probability that both designs will require rework (*External RW Probability*).

The first outflow, *Redesigning eRW*, represents taking out designs considered to be external rework from the stock as they are redesigned in proportion to the fraction of designs in rework that can be considered as external rework (*eRW Density in RW*). The second outflow, *Dump eRW*, is part of the process of transferring integration errors between coupled components as outlined above. In addition to the excess external rework designs that are added as new integration errors to the stock of *IE in RW*, it is necessary to remove all of the excess external rework from the stock of external rework. This is true because the balance of the excess rework that was not added to the stock of *IE in RW* is associated with redesigned but unfixed integration errors in the coupled component. Therefore, the design's original designation as external rework affiliated with a coordinated integration error is no longer valid. Mind you, the design itself will remain in the stock of rework and thus may be redesigned somewhat unnecessarily. This notion of unnecessary rework is the basis for some of the phantom work captured by the model and will be discussed in more detail later in this chapter.

Unfixed Coordinated Integration Errors

The stock of unfixed coordinated integration errors (*Unfixed cIE*) represents the set of coordinated integration errors whose associated designs have been redesigned since the last build event where the integration error was not fixed. As such the integration error was carried over. In order to track these unfixed errors, the following stock and flow structure is used:

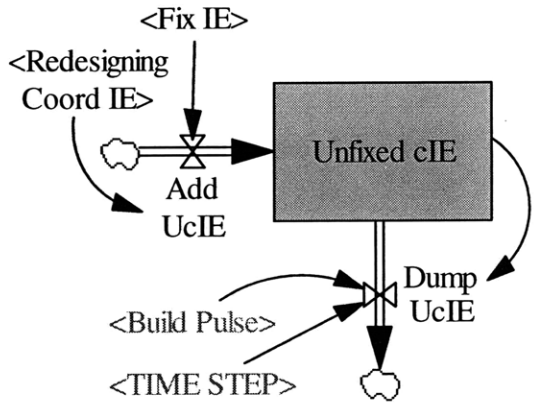


Figure 25 Unfixed Coordinated Integration Errors (Unfixed cIE)

There is a single stock representing coordinated integration errors which have been redesigned but not fixed since the last build event. There is one inflow which adds to the stock of *Unfixed cIE* as coordinated integration errors are redesigned but not fixed. There is one outflow that empties the stock of *Unfixed cIE* at the time of each build event.

Coordinated Integration Errors in Rework

The stock of coordinated integration errors in the designs in rework (*Coordinated IE in RW*) represents the set of designs awaiting rework that have associated integration errors that have been coordinated. Similar to the stock of known component errors in the stock of designs in rework (*Known CE in RW*), it is necessary to account for the stock of *Coordinated IE in RW* in order to determine the rate at which integration errors can be fixed through redesign. This is based on the assumption that integration errors that have not been coordinated can not be fixed. Additionally, it is necessary to keep track of these coordinated integration errors to determine the rate of transfer of integration errors between coupled designs as outlined above.

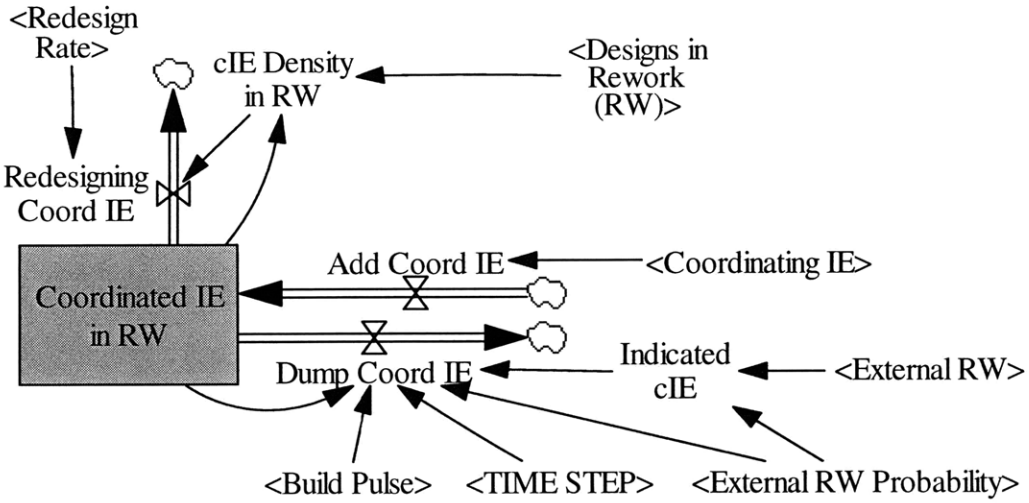


Figure 26 Coordinated IE in RW

There is one inflow to the stock of *Coordinated IE in RW – Add Coord IE* – and two outflows – *Redesigning Coord IE* and *Dump Coord IE*. The inflow, *Add Coord IE*, represents the addition of integration errors to the stock as designs with integration errors are coordinated and sent for rework. The rate at which integration errors are added to the stock is determined by the rate of *Coordinating IE* which was discussed previously.

The first outflow, *Redesigning Coord IE*, represents taking out integration errors from the stock of *Coordinated IE in RW* as designs with coordinated integration errors are redesigned. The second outflow, *Dump Coord IE*, is formulated in a manner similar to that of the flow *Dump eRW* discussed in the previous section. It compares the number of coordinated integration errors in rework for a given component to the indicated number of coordinated integration errors it should have based on the amount of external rework its coupled component has yet to complete.

BUILD SECTOR

Build Sector Overview

The build sector of the NPD model is structured to account for the current status of the set of build parts for a new product development project during vehicle testing. This sector includes the set of parts (also referred to as build parts) which correspond to the designs that make up each component of the new product under development. While the paper designs modeled in the design sector reflect the latest version (revolution) of the design for each given part, the parts in the build sector reflect the state of the designs at various points in time – i.e. at the time of each build event. While the set of designs continue to evolve, the parts for each build are like a snapshot in time – they don't change during vehicle testing. In addition to the set of parts, the build sector also accounts for the component errors and the integration errors associated with these parts using a co-flow structure as in the design sector. Figure 27 provides a high level depiction of the build sector of the NPD model.

The structure of the build sector is fairly straightforward. There is a single stock of *Build Parts* and its associated flows. There are two additional stocks, one for component errors (*Build CE*) and another for integration errors (*Build IE*) that are associated with the build parts, along with their associated flows. Just as is the case of the design sector, the build sector is subscripted for the different components, which means there is a different set of build parts and its associated errors for both component A and B. Additionally, since there may be multiple build events during the course of a new product development project, the build sector is also subscripted for the different builds. This means that there are different stocks of parts and errors not only for each component but for each build as well.

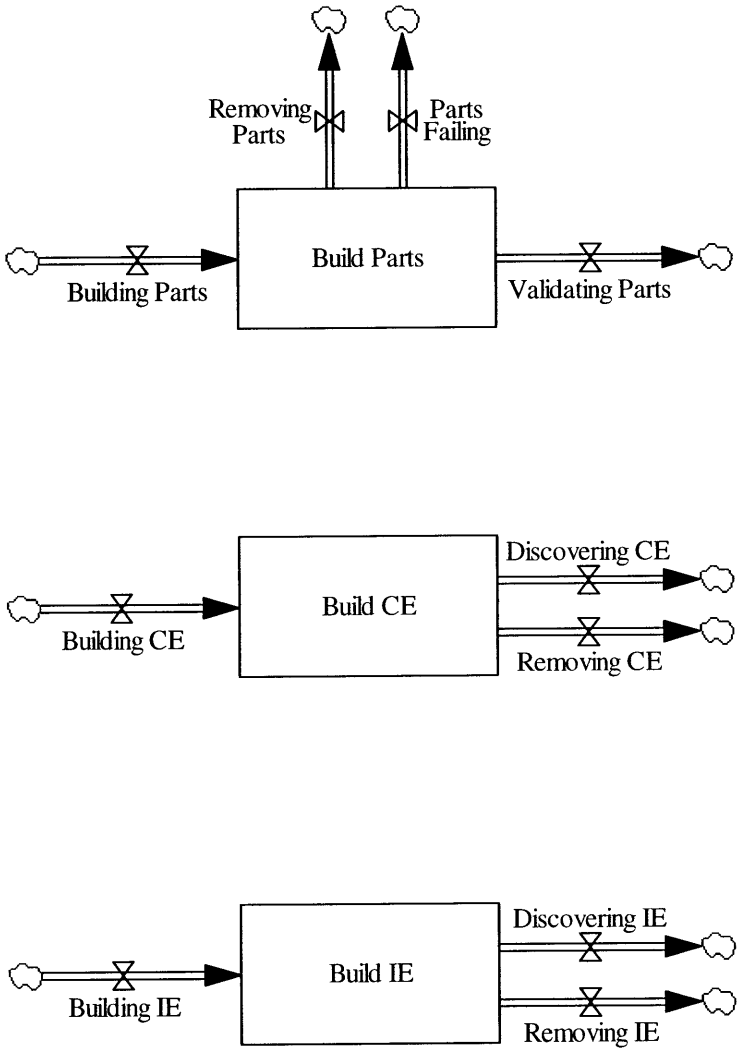


Figure 27 Build Sector

In the build sector, each stock has a single inflow that represents the building of parts and their corresponding errors. Each stock has an outflow that represents the discovery of errors through vehicle testing. In the stock of build parts this is referred to as *Parts Failing* while in the error stocks it is referred to as *Discovering CE/IE*. Additionally, there is an outflow to the stock of *Build*

Parts which validates parts as they are tested and found to be error free. There is no corresponding outflow to *Validating Parts* from the error stocks because vehicle testing is assumed to be perfect (i.e. it finds all true errors and no false ones) and therefore it wouldn't make sense to "validate" an error.

Finally, there is also an outflow from each of the stocks that represents the removal of build parts and errors. Parts and their associated errors are removed from the stocks for a given build when the same part with the same error(s) fails vehicle testing in another build. This can occur because it is possible (and indeed common in the client company) to overlap build events and subsequent vehicle testing. This means that the vehicle testing from a given build is still ongoing when the next build event occurs and the testing of the subsequent version of the vehicle begins. In some cases the errors from the original build designs and parts have not yet been discovered and/or fixed and therefore are replicated in a subsequent build. When this type of duplicated error is discovered in vehicle testing, it is discovered or "removed" from all build versions.

The NPD model by convention allows no more than two vehicle builds and testing to be ongoing simultaneously. This reflects the true state of vehicle builds and testing in the client company. While it is common to have two successive versions (revolutions) of a full vehicle in testing at the same time, historically the time between builds is sufficiently long enough that vehicle testing is completed for a given build before the second successive build occurs. For example, vehicle testing on Version 1 is normally completed before Version 3 is built. Additionally, on those rare occasions when vehicle testing is not completed as described, vehicle testing is cut off on the version of the vehicle from two generations ago. For example, when

Version 3 is built the vehicle testing on Version 1 would be cut off or any future test results (i.e. errors discovered) from Version 1 would be ignored.

As mentioned, throughout the course of a new product development project there may be multiple build events. The NPD model allows for this by using subscripting in the build sector. In addition to the stocks, flows and auxiliary variables in the build sector being subscripted for the different components (A and B), they are also subscripted for the different builds. In the NPD model, the basic builds include a Prototype build (referred to as *Proto*), a Design Intent Build (*DIB*), and a First Production Event build (*FPE*). While these are the pre-planned builds, the model also allows for additional builds to be scheduled as necessary. This often occurs in the client company if a given build does not go well (i.e. too many errors are discovered). These additional builds are referred to in the model as Proto2, Proto3...DIB2, DIB3...,and FPE2, FPE3...

While the simulation model accounts for the different builds using subscripting, it is necessary for the model to be able to distinguish between the builds in terms of their generation or vehicle version. As mentioned, only two vehicle builds and testing are allowed to be ongoing (or active) at any given time. Of these two vehicle builds, it is necessary to distinguish between the current build (i.e. the latest version of vehicle parts and errors) and the previous build (the prior generation or version of vehicle parts). While a description of how the model handles this distinction will be reserved for the technical documentation, the use of this convention will be necessary in the sections that follow.

Build Parts

The main stock of concern in the build sector is that of the *Build Parts*. These parts correspond to the set of designs in the main stock and flow chain in the design sector. At the time of a build event (for example, the Prototype build) a set of parts that reflect the state of the designs for each component is built. These parts are then assembled into a full vehicle and vehicle testing begins. While the build process includes assembly into a full vehicle, in the model each component maintains its own separate stock of *Build Parts* for each build event. Through the process of vehicle testing, component and/or integration errors are discovered in the set of build parts when they fail testing. Other parts are found to be without error and are thus validated. As parts fail in vehicle testing, their corresponding designs are sent for coordination and/or rework depending on their error profile.

The stock of *Build Parts* and its associate inflow and outflows is depicted in Figure 28. There is one inflow – *Building Parts* – and three outflows – *Parts Failing, Validating Parts,* and *Removing Parts* – to the stock of *Build Parts*.

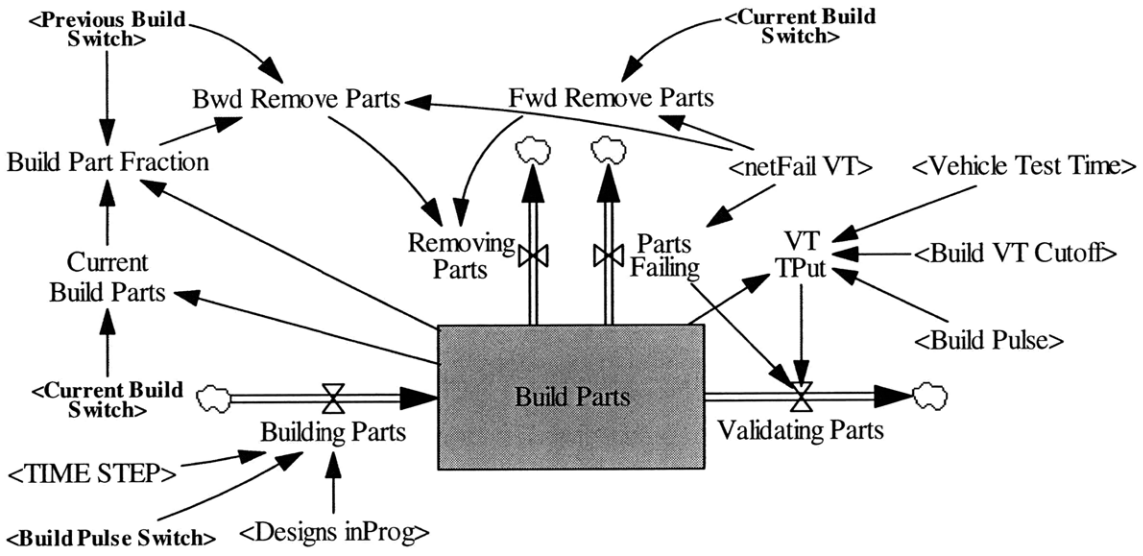


Figure 28 Build Parts

The stock of *Build Parts* at any given point in time is the integral of the inflows minus the outflows given an initial value of zero. Indeed, each of the stocks (*Build Parts*, *Build CE*, and *Build IE*) has an initial value of zero for each of the components and each of the builds.

Building Parts is the only inflow to the stock of build parts. It represents the process of converting designs into parts and assembling the parts into full vehicles in preparation for vehicle testing. For the purposes of this model, this process is greatly simplified and it is assumed that the parts and vehicles are built in a single simulation time step at the time of a build event.

As formulated, the flow of building parts takes the set of designs that are in progress for a given component (*Designs inProg*) and adds them all to the stock of *Build Parts* in a single time step. The set of parts corresponding to the designs in progress are built for all components at the

time of each build with each component maintaining a separate stock of build parts and the associated error co-flows.

The two major outflows for the stock of Build Parts are *Parts Failing* and *Validating Parts.* These outflows are the result of vehicle testing which is modeled as a simple first order delay where vehicle testing throughput (*VT TPut*) represents the rate at which build parts are tested.

The formulation for the flow of *Parts Failing* is based on the variable *netFail VT* which is an auxiliary variable representing the net rate at which build parts fail in vehicle testing because a component error, an integration error or both were discovered in a given part. The formulation for this variable will be discussed in detail in the vehicle testing section of this chapter.

The rate of validating parts is simply the difference between the rate at which build parts are tested (*VT TPut*) and the rate at which parts fail (*Parts Failing*). Again, this formulation is based on the assumption that vehicle testing is perfect. If a part does not fail then it is validated. Like bench testing in the design sector, vehicle testing is modeled as a simple first order delay where *Vehicle Test Time*, which may be different for the different components, represents the average time it takes to conduct vehicle testing on a part for a given component. Therefore, the rate at which build parts are tested (*VT TPut*) can be formulated by taking the number of build parts remaining and dividing it by the average test time. The variable *Build VT Cutoff* is a switch used to cut off vehicle testing when the stock of build parts falls below a minimum threshold. The cutoff switch also ensures that vehicle testing only occurs on active builds via the variable *Build Switch.* There will be a section included in the technical documentation that describes the formulation for these and the other switches used in the model. Just as with the rates of flow in the

design sector, the *Build Pulse* variable is used to pause vehicle testing at the instant in time when a build event occurs.

The third outflow to the stock of *Build Parts* is *Removing Parts.* When a part fails in vehicle testing during any build, if the same part with the same errors exists in another active build, it is removed from that build. Removing the failing parts (and their associated errors as you will see in the co-flow description later in this chapter) prevents the model from reacting twice to the same error. This is akin to a test engineer finding a duplicate error in testing and, recognizing it as such, not writing it up as a new error but a repeat of a previously identified error.

There are two mechanisms by which removing parts takes place in the NPD model. The first is called forward removing parts in the model (*Fwd Remove Parts*) and refers to the case where parts are removed from the stock of build parts for a given build, when it is considered to be the current build, because identical parts fail in a previously ongoing build. The second is called backward removing parts (*Bwd Remove Parts*) and refers to the case where parts are removed from the stock of build parts for a given build, when it is considered to be the previous build, because identical parts fail in a more recent (i.e. the current) build.

Additionally, as parts are redesigned in the design sector their associated errors are also removed from the pre-existing stocks of errors in build parts. While removing these errors means that they won't be discovered in the model as they would be on the actual vehicle, this is akin to a designer ignoring errors found on parts that have already been redesigned. Ultimately, the handling of errors in the co-flow structure ensures that every existing error in an ongoing previous

build also exists in the current build which allows failing parts (and newly discovered errors) from the previous build to be removed as duplicates from the current build.

Parts Error Co-Flow Overview

Much like the co-flow structure used in the design sector, the build sector uses two co-flows to track the errors (component and integration) associated with the set of build parts. These co-flow structures are depicted below the stock and flows for the set of build parts in Figure 27. There are two stocks, one each for the component and integration errors associated with the stock of build parts. There is a corresponding inflow in both of the co-flow structures to the one inflow of *Building Parts* to the main stock of *Build Parts*. There are outflows of discovering component and integration errors that correspond to failing parts. There are also outflows for removing component and integration errors that correspond to the removal of parts as described earlier. As mentioned, there are no outflows in the component and integration error co-flows that correspond to the outflow of *Validating Parts* from the main stock of Build Parts. The formulation for each of the corresponding stocks and flows in the co-flow structure is nearly identical to that of the main stock and flow chain for *Build Parts*. Where appropriate I will minimize the description of formulations that have been previously discussed.

Component Errors in Build Parts

The stock of component errors in build parts (*Build CE*) maintains the component errors that are associated with the stock of build parts discussed previously. There is one inflow – *Building CE* – which adds component errors to the stock at the time of a build event. There are two outflows – *Discovering CE* and *Removing CE* – that correspond to the outflows of *Parts Failing*

and *Removing Parts* from the stock of *Build Parts*. A depiction of the co-flow for component errors in build parts can be found in Figure 29.

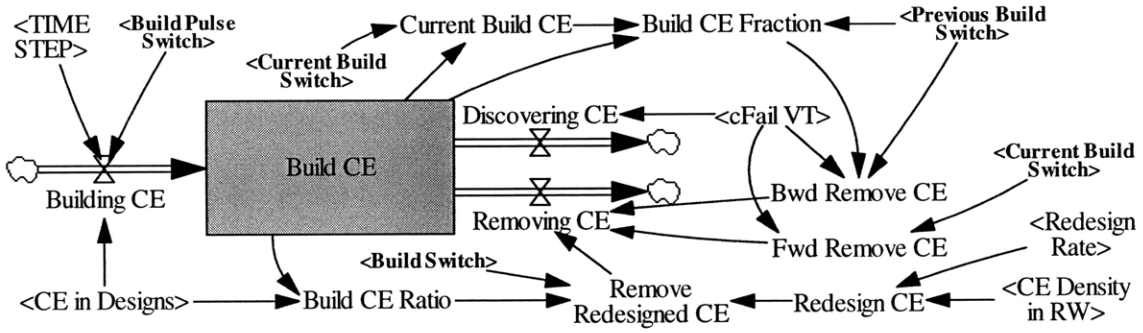


Figure 29 Component Errors in Build Parts (Build CE)

The formulation for the inflow *Building CE* is analogous to that of Building Parts except that instead of building the set of designs in progress the inflow adds the number of component errors that are associated with those designs (*CE in Designs*). Again, these errors are added in a single simulation time step at the time of the build event.

The outflow *Discovering CE* represents the rate at which component errors are drained from the stock of *Build CE* for a given build as they are identified in vehicle testing. The formulation is straightforward in that it relies on the rate of discovering component errors in vehicle testing (*cFail VT*) which will be discussed in further detail in the Vehicle Testing section of this chapter.

The outflow *Removing CE* represents the rate at which component errors are removed from the stock of *Build CE* in vehicle testing from a given build because they have been discovered in an ongoing previous build, subsequent build or the part with which the error is associated has been

redesigned. While the formulation for *Removing CE* is analogous to *Removing Parts* as it both forward removes and backward removes component errors, it varies in that it also removes component errors when the design for the associated part is redesigned.

Removing redesigned component errors (*Remove Redesigned CE*) represents the rate at which component errors are removed from the stock of *Build CE* during vehicle testing because the part with which the error is associated has been redesigned since the build event occurred. Because the design / part has been redesigned since the beginning of vehicle testing, the component error may or may not have been fixed during the course of its redesign. Thus, the component error associated with the previous version of the part will be ignored when it is discovered in vehicle testing. In order to account for this in the model, the original component errors associated with parts prior to their redesign parts are removed from the stock of *Build CE* and thus will not be discovered.

Because only component errors that are known by the designer can be fixed in redesign, only known component errors that are redesigned are removed from the stock of *Build CE*. However, not all of the component errors that are redesigned should be removed from the stock of *Build CE* because not all of the component errors that exist in the current set of designs were necessarily included in the build event. Remember, while the set of designs continues to evolve throughout the course of the new product development project the build parts and their associated errors represent a snapshot in time and do not change except from build to build. For example: One might imagine a part that was designed or redesigned after the build event occurred and vehicle testing began. In the course of designing or redesigning the part a component error may have been introduced that is subsequently discovered in bench testing and sent for rework. In this case, the component error would not have been included as part of the stock of component errors in build parts (*Build CE*)

since the error wasn't generated until after the build event occurred. Therefore, this error shouldn't be removed from the stock of *Build CE* as its associated design is redesigned. The formulation for removing redesigned component errors handles this likelihood using the variable *Build CE Ratio* which maintains the fraction of component errors in the current set of designs that exist in a given build (*Build CE*). As designs with component errors are redesigned, the probability that they are component errors that are also in the stock of *Build CE* is equal to this fraction and thus component errors are removed proportionally.

Integration Errors in Build Parts

The stock of integration errors in build parts (*Build IE*) maintains the set of integration errors that are associated with the stock of build parts discussed previously. Completely analogous to the stock of component errors in build parts (*Build CE*), there is one inflow – *Building IE* – which adds component errors to the stock at the time of a build event and there are two outflows – *Discovering IE* and *Removing IE* – that correspond to the outflows of *Parts Failing* and *Removing Parts* from the main stock of *Build Parts*. Figure 25 is a depiction of the co-flow structure for integration errors in build parts.

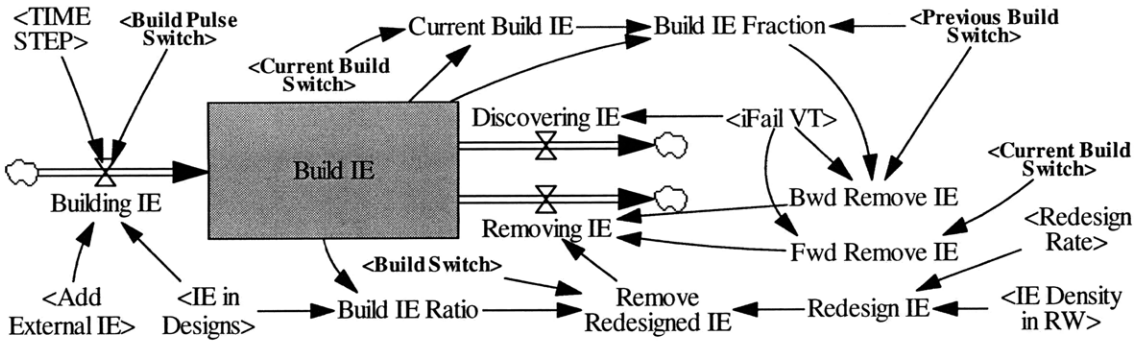


Figure 30 Integration Errors in Build Parts (Build IE)

With the exception of the formulation for the one inflow, *Building IE,* the structure and formulation for *Build CE* are completely analogous to those for *Build CE.* The formulation for Building IE is slightly different from that of *Building CE* in that in addition to building the integration errors that exist in the designs in progress (*IE in Designs*) at the time of the build event, in the case of integration errors it is also necessary to include the "new" integration errors that are created from the imbalance in external rework discussed previously (*Add External IE*). Doing so allows these new integration errors to be discovered in vehicle testing.

VEHICLE TESTING SECTOR

Vehicle Testing Overview

Vehicle testing is the process by which errors are discovered in full vehicles after they have been assembled as part of a build event. Vehicle testing typically refers to the process of rigging up a vehicle with the necessary test equipment and then operating it in a variety of operational conditions and recording any failures or concerns that might reflect an error in the design. Often

times the vehicle is ridden on a test track and miles are accumulated in an effort to determine where and when components may break down. This is generally referred to as high mileage testing. However, many errors are found up front in the process of building the vehicle itself. For example, it is not uncommon to find two components that don't quite fit together properly during the assembly process. This would be an example of an integration error as defined in the NPD model. For the purposes of this model, vehicle testing includes the testing done and errors discovered in both the build event and the subsequent road testing.

Unlike bench testing, which is assumed to only be able to find component errors, vehicle testing is formulated to be able to discover both component and integration errors. Additionally, for simplification purposes and without loss of generalization, vehicle testing is considered to be perfect in that as vehicle parts are tested any associated errors will be discovered.

The vehicle testing sector of the NPD model consists of the vehicle testing itself – which discovers the errors in build parts – and the necessary accounting to determine how to move the designs and their associated errors between the various stages of design (i.e. the stocks of *Designs in Test*, *Designs Released*, *Designs in Coordination*, and *Designs in Rework*). Each will be discussed in turn in the sections that follow and full documentation of the equations can be found in the appendices.

Discovering Errors

The main driver behind the rate at which errors are discovered in vehicle testing is the throughput rate at which build parts are tested. The variable *VT TPut* is modeled as a simple first order delay where *Vehicle Test Time*, represents the average time it takes to conduct vehicle testing

on a part for a given component. The average test times may be different for the different components which can result in one component identifying its errors sooner than the other giving its designers more time to do root cause analysis and redesign as necessary.

The use of a first order delay as the driver of vehicle testing results in a rate of vehicle testing for each component with profiles like those in Figure 31. In this simple case, both components begin testing at time 0 with 1000 build parts and component A has an average vehicle test time of 3 months while component B's test time is set at 6 months.

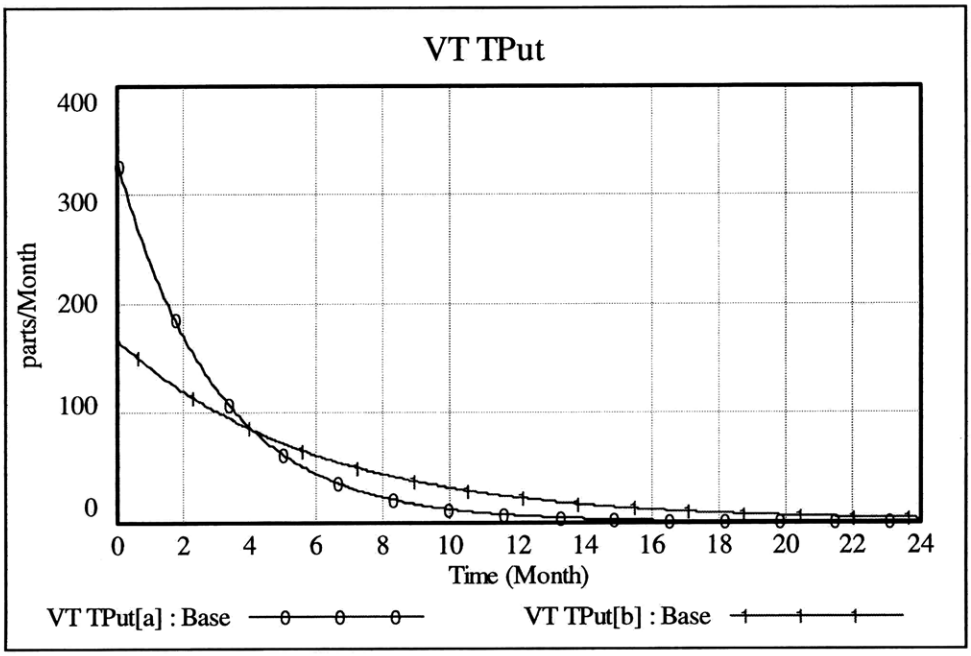


Figure 31 Vehicle Test Throughput

As you can see in the graph of *VT TPut*, the result for both components is a high throughput rate early in the vehicle testing that decays exponentially over time. The practical result is that many parts are tested early in vehicle testing and, as you will see, their associated errors are

discovered early as well. However, some parts aren't tested for a much longer time which delays discovery of their associated errors. This testing profile, in particular the resulting error discovery profile, reflects the testing results experienced in the client company.

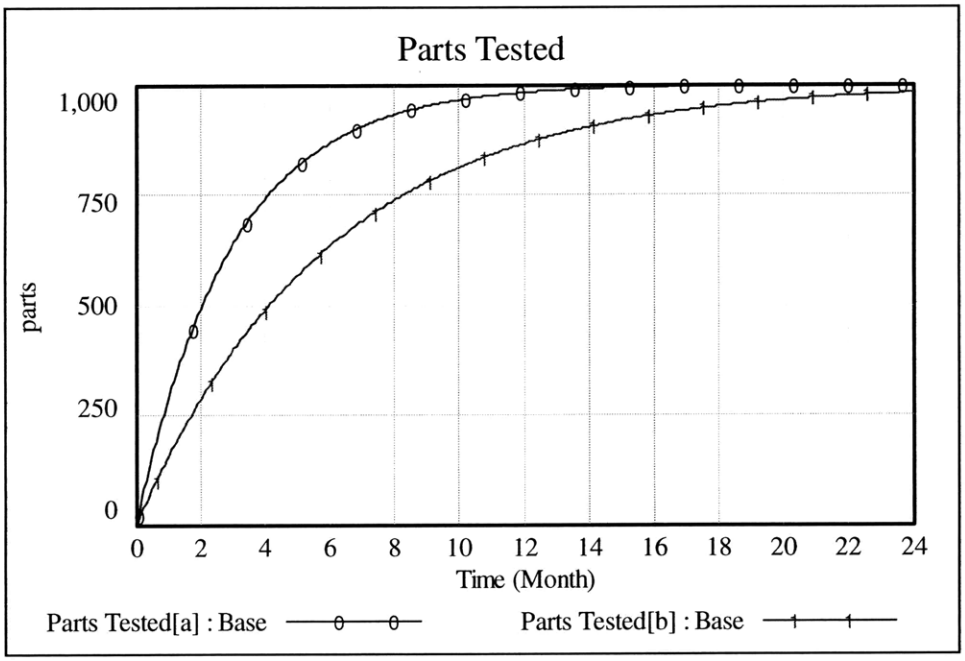


Figure 32 Parts Tested

Integrating the rate of vehicle testing (VT TPut) captures the number of parts tested cumulatively over time and perhaps gives a more intuitive sense for the rate at which parts are tested. Some parts only require a short time to be tested and their associated errors discovered. Others take a much longer time. Using a previous example, an integration error that causes two parts to not fit together properly will be discovered during the assembly process as part of the build event itself. A vibration problem that causes undue stress on the vehicle frame over time resulting in a fracture might not be discovered until thousands of miles have been accumulated on the test track.

The use of a first order delay structure for vehicle testing also assumes that the build parts and their associated errors are perfectly mixed. The likelihood of a given part being tested next is no more likely than any other. Also, the parts are assumed to have component and/or integration errors in proportion to the density of component and integration errors in the co-flow structure from the build sector. As a result, as a given part is randomly selected from the stock of build parts for testing it can have a component error, an integration error, or both. For accounting purposes in the model, it is necessary to be able to distinguish between each of these cases because designs that correspond to parts with these different error profiles will be handled differently by designers in real life and by the model as well. For example, a part that is found to have a component error only will immediately be sent for rework (i.e. redesign). However, a part that is found to have both a component error and an integration error will either be sent directly for rework or for coordination of the integration error first.

Figure 33 shows a schematic showing the relationship between the significant variables used in the model for the discovery of errors in vehicle testing.

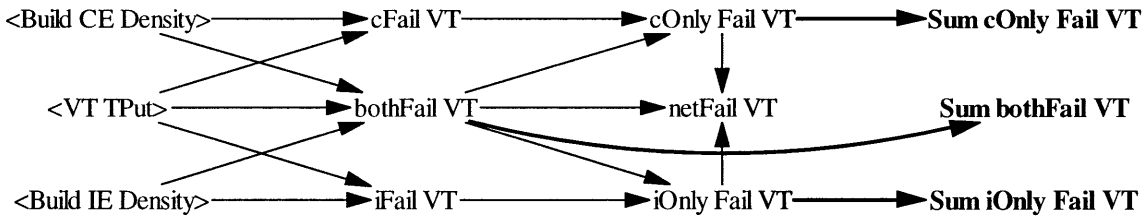


Figure 33 Error Discovery in Vehicle Testing

In addition to the rate at which build parts are being tested, the other main driver for discovering errors through vehicle testing in the NPD model is the density of the two error types in the build parts. *Build CE Density* and *Build IE Density*, respectively, represent the fraction of build parts with component and integration errors. The formulation for both of these densities has been discussed and their equations can be found in the technical documentation. As mentioned, the formulation for determining the rate at which parts fail for having a particular kind of error is simply to multiply the rate at which parts are being tested (*VT TPut*) by the density of errors in the stock of build parts. The formulation for determining the rate at which parts fail for having both types of errors (*bothFail VT*) is simply the rate at which they are tested multiplied by the joint probability of them having a component and an integration error. From these failure rates, we can determine the rate at which parts fail for having only a component error or only an integration error (*cOnly Fail VT* and *iOnly Fail VT*, respectively). Additionally, we can determine the net failure rate which represents the rate at which build parts fail for having a component error, an integration error, or both. This is done by adding the rate at which parts fail for having both types of error (*bothFail VT*) with the rates of a part failing for having only one type of error just discussed.

You'll note that error discovery is unique for the different components and the different builds as each component's parts are vehicle tested separately in the model and they are tested during multiple builds. However, the build events are common across the components.

These failure rates are what drive the outflows of parts and errors from their respective stocks in the build sector. However, it is also necessary to combine the vehicle testing results across the different builds. Because there can be as many as two different builds (generations or revolutions of the vehicle) and vehicle testing overlapping, it is necessary to sum up the failure rates from

vehicle testing from each build before taking appropriate action in the design sector (e.g. sending a design for coordination and/or rework).

In adding together the failure rates as outlined above, we can reasonably assume that the same part with the same errors will not be found at the same time in two different builds and thus added together as if they were different parts and/or different errors. We can assume this because of the forward and backward removing of parts and errors between builds as described in the build sector section of this chapter and because we use a sufficiently small time step (*dt*) in the simulation model. In the NPD model, the simulation *Time Step* is set at .0625 of the time unit which is months.

Vehicle Test Accounting

Once we have determined the failure rates for build parts, we must determine how to handle the designs and their newly discovered errors. There are two major components to the framework for handling failed parts in the NPD model. The first is to model the process used to decide what should be done with a design whose part fails in vehicle testing. The second component is determining where the designs corresponding to the failed parts reside (i.e. in which stock of designs from the design sector) at the time of the error discovery. Based on these two processes, the model can then move the designs and their associated errors to either the stocks for rework or coordination as appropriate.

The decision process used to decide what should be done with a design whose part fails in vehicle testing is fairly straightforward. If a part fails that has a component error only, then its corresponding design and the associated component error are sent directly to rework. If a part fails

that has an integration error only then its design and the error are sent for coordination prior to being sent for rework. A part that fails for having both a component error and an integration error will either have its design and associated errors sent directly to rework or they will be sent for coordination first. This decision is based on the designers' willingness to delay rework on a known component error until they have a chance to coordinate the required rework for a known integration error. This willingness to delay rework on a design until the integration error has been coordinated is reflected in the variable *Coordination Fraction* which will be discussed in more detail below. Essentially, this variable determines the fraction of designs with both types of errors that are sent for coordination first upon discovery.

The second component to the framework for handling designs with newly discovered errors is to determine where they reside at the time of error discovery. This is necessary because design and redesign work continue during vehicle testing. This means that designs and their associated errors can and do move between the various stocks in the design sector while a static representation, a prototyped part(s), of those designs are tested in the vehicle testing sector of the model. While there are 4 stocks of designs that have errors associated with them in the design sector – *Designs in Test*, *Designs Released*, *Designs in Rework* and *Designs in Coordination*, we know that errors found in parts in vehicle testing will not correspond to designs (and their associated errors) that reside in the stock of *Designs in Test (DIT)*. This is true for three reasons. First, at the time of a given build event all of the designs and their associated errors that are in the stock of DIT are dumped into the stock of Designs Released. Secondly, all designs that are redesigned and sent for bench testing (i.e. added to the stock of *Designs in Test*) have their associated errors removed from the stocks of *Build CE* and *Build IE* as described in the build sector. Therefore, none of these errors would be discovered in vehicle testing Lastly, the only other way for a design and its

associated errors to move into the stock of DIT is through new design work which would have occurred after the build event Therefore, none of the errors discovered in vehicle testing would correspond to these new designs.

The result of the previous discussion is that we know that errors discovered in vehicle testing must correspond to designs (and their associated errors) that are in the stocks of *Designs Released*, *Designs in Coordination* or *Designs in Rework*. As designs with certain error profiles fail vehicle testing, we pull the designs from the respective stocks just mentioned in proportion to the number of designs with the same error profile that reside in each stock. For example, as parts fail vehicle testing because they have a component error only (*Sum COnly Fail VT*), we would pull designs from the stock of *Designs Released* in proportion to the number of designs with component errors only that are in the stock released designs *(% CEo in DR)*.

Given the failure rates for parts, the location of designs corresponding to the failing parts, and the decision regarding where to send a design that has failed in vehicle testing, we can formulate the flows of designs and their associated errors between the various stages of development due to failures in vehicle testing. See the detailed model description and the technical documentation in Appendix A and B for a discussion of the use of the formulation above as a basis for discussing the movement of component errors, integration errors, and designs in conjunction with vehicle testing.

LABOR SECTOR

The labor sector of the NPD model accounts for the number of designers that are available to work on the different activities (design, redesign and coordination) conducted directly by designers in the detailed design phase of a new product development project. As in the execution of a real project, the NPD model provides for the ability to add designers to the project team as work pressure builds. The model also accounts for the effect of adding inexperienced designers to a project.

The basic framework for determining the number of designers available is to calculate the desired number of designers based on the amount of design, redesign, and coordination work to be done and the time remaining until the next deadline (e.g. the next build or product launch) and then, with some delay, add additional designers. In bringing in new designers, the NPD model distinguishes them as "inexperienced" designers with a lower productivity level than experienced designers. Over time, these new designers mature into experienced designers and increase their productivity accordingly. Figure 34 depicts the relationship between the key variables included in the Labor sector of the NPD model.

As in the other sectors of the model, the stocks, flows, and auxiliary variables used in the Labor Sector are subscripted to account for the different components (A and B) in the project. In the case of the Labor Sector, this allows the project team to set the number of designers available to work on each component based on the amount of work to be done and the productivity levels of the component designers. The deadlines, however, are project deadlines and are the same for both components.

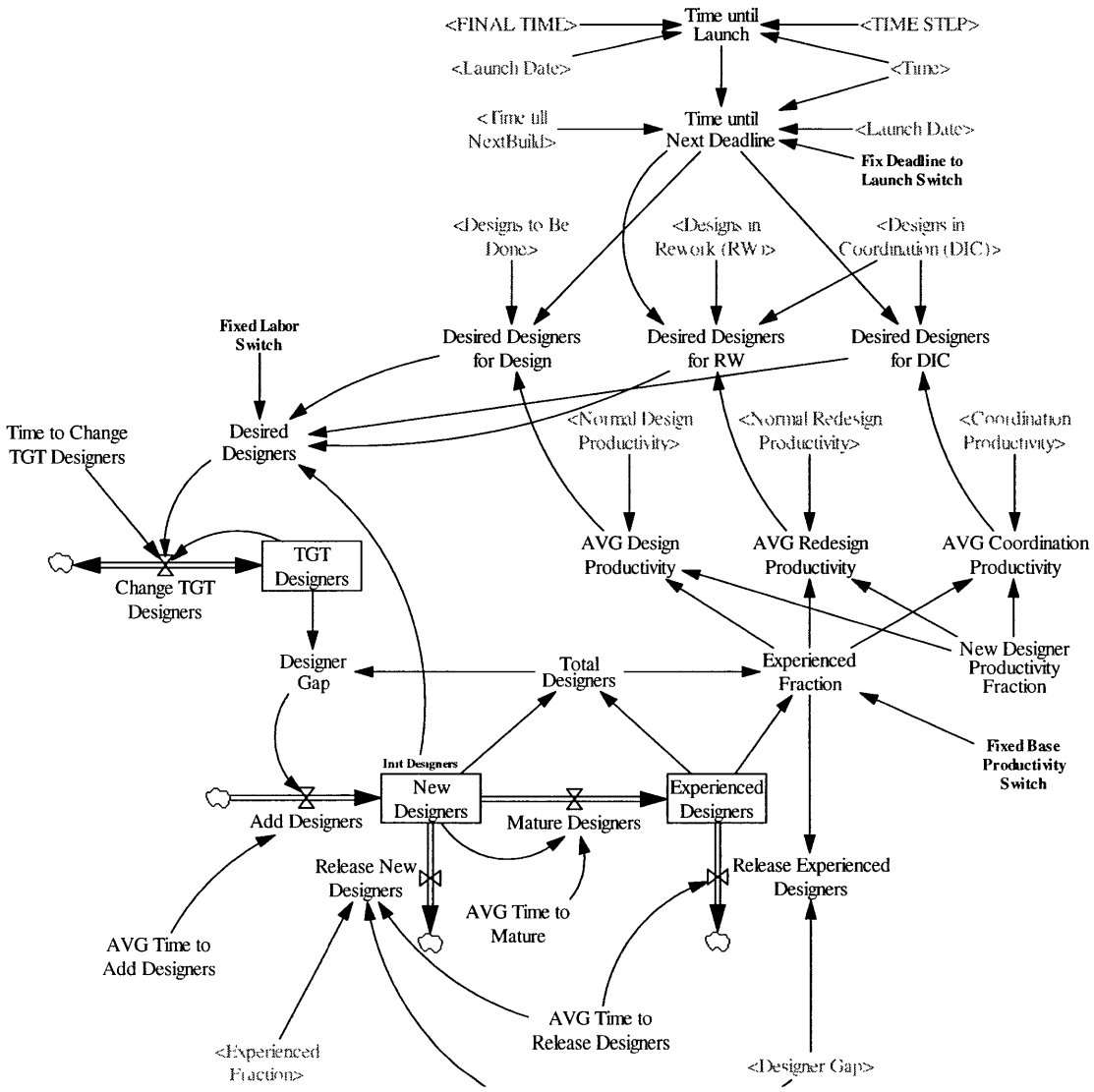


Figure 34. Labor Sector

The Labor Sector can be broken into three main sections. The first section accounts for the number of designers the project team needs (*Desired Designers*) based on the amount of work pressure the project is under. The second section sets the target number of designers that the project team is authorized to have (*TGT Designers*) based on the desired number of designers and some time to change the authorized number. The final section of the Labor Sector determines the total number of designers available to do project work (*Total Designers*) while accounting for the

experience factor of the designers on the project team by adding designers to the team as inexperienced designers (*New Designers*) with lower productivity and, over time, having them mature into *experienced designers*. Each of these sections will be discussed in some more detail below. Detailed discussion and technical documentation can be found in Appendix A and B respectively.

Desired Designers

The first section of the Labor Sector determines the number of designers the project team needs (*Desired Designers*) based on the amount of work pressure the project is under. The number of designers that are needed is based on the amount of design, redesign, and coordination work that needs to be done before the next perceived hard deadline. The model structure that is used to determine the number of desired designers is depicted in Figure 35.

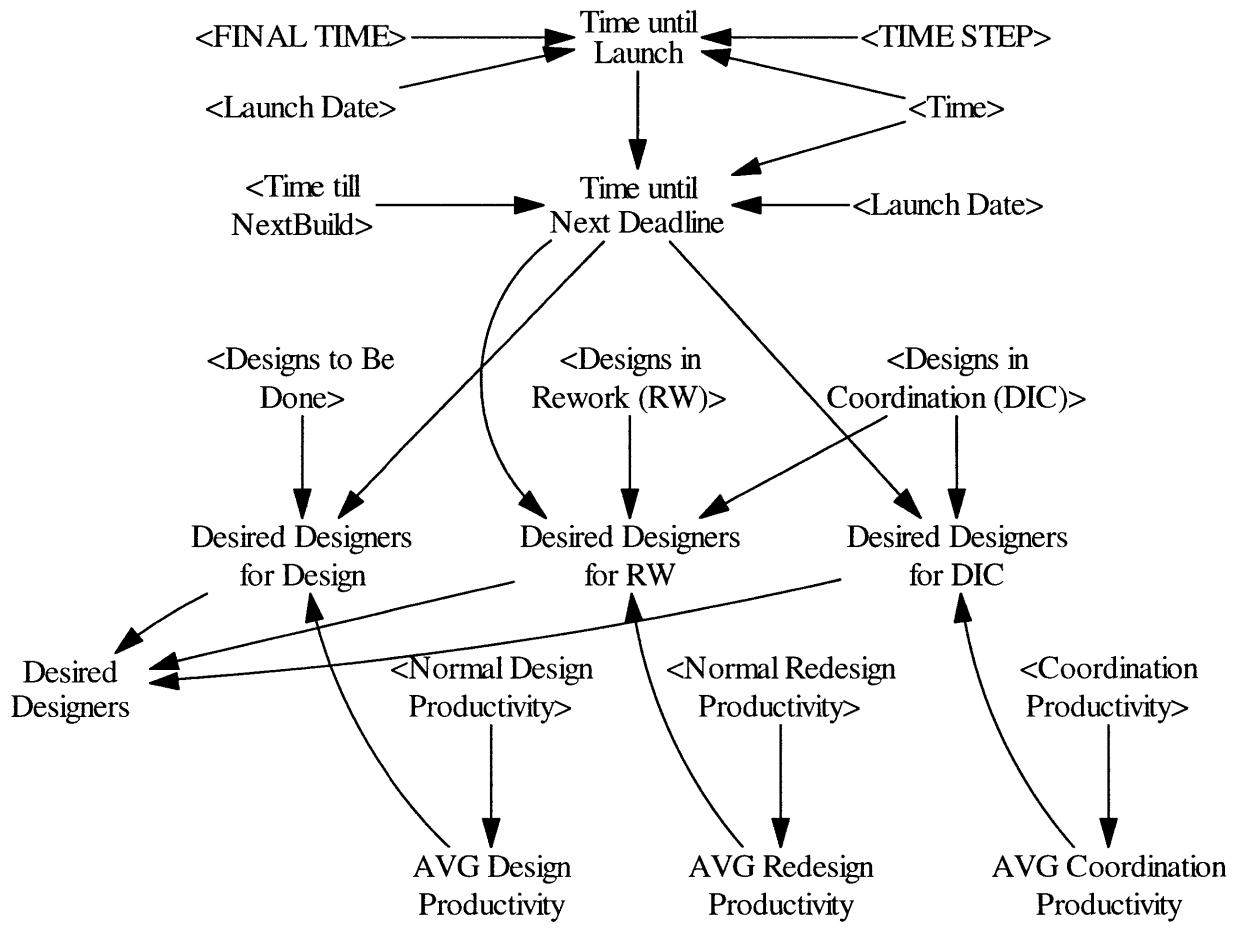


Figure 35 Desired Designers

In order to determine the total number of designers desired for the project team it is necessary to determine the number of designers desired for design, redesign, and coordination. Each of these is determined in the same way. For example, the Desired Designers for Design is determined by first dividing the number of designs in the stock of Designs to Be Done by the Time until Next Deadline. This determines the rate at which designs must be completed in order to be finished with the designs by the next deadline. By further dividing this rate by the average design productivity of our designers (AVG Design Productivity) we can determine the number of designers needed in order to complete the designs by the deadline.

The average productivity for designers in each of the activities (design, redesign, and coordination) is set at a percentage of the normal productivity based on the fraction of designers who are experienced versus new. This fraction will be discussed in further detail below.

In the base model, the hard deadline (*Time until Next Deadline*) is perceived by the project team to be the scheduled launch date of the product under design. However, in the policy analysis discussed in the next chapter the deadline is alternatively set to the next build event to explore the effects on project performance.

Once the number of designers desired for design (*Desired Designers for Design*), redesign (*Desired Designers for RW*), and coordination (*Desired Designers for DIC*) are determined the total number of *Desired Designers* is easily calculated by summing the three.

Target Designers

The second section of the Labor Sector sets the target number of designers for the project team (*TGT Designers*) for each component (A and B). The target number of designers is simply a smoothed forecast of the desired designers using first-ordered exponential smoothing which eliminates much of the short-term, high frequency "noise" that is inherent in the actual number of *Desired Designers*. The model structure for determining the target number of designers (TGT Designers) is shown in Figure 36.

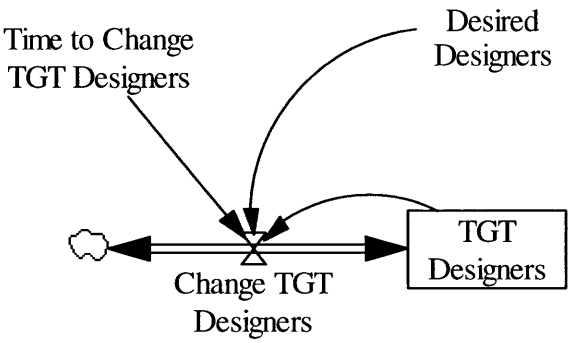


Figure 36 Target Designers

The target number of designers (*TGT Designers*) is represented by a stock which changes based on the difference between its current value and the current value of *Desired Designers*. The amount it changes, represented by the flow *Change TGT Designers*, is tempered by the *Time to Change TGT Designers*. This negative feedback structure, in effect, smoothes the forecast of the number of designers needed by eliminating the short-term variation in *Desired Designers* while still adjusting to the longer-term changes. Given that the actual number of desired designers can vary widely based on the amount of work to be done, the productivity of designers and the amount of time until the next deadline it is important "smooth" this number using adaptive expectations.

Total Designers

The last section of the Labor Sector adds new designers to the project team and accounts for the experience level of all the designers using a simple aging chain structure. This model structure is depicted in Figure 37. The result is a means for determining the number of designers (*Total Designers*) available to do project work and their experience level (*Experienced Fraction*) which drives the average productivity of the workforce.

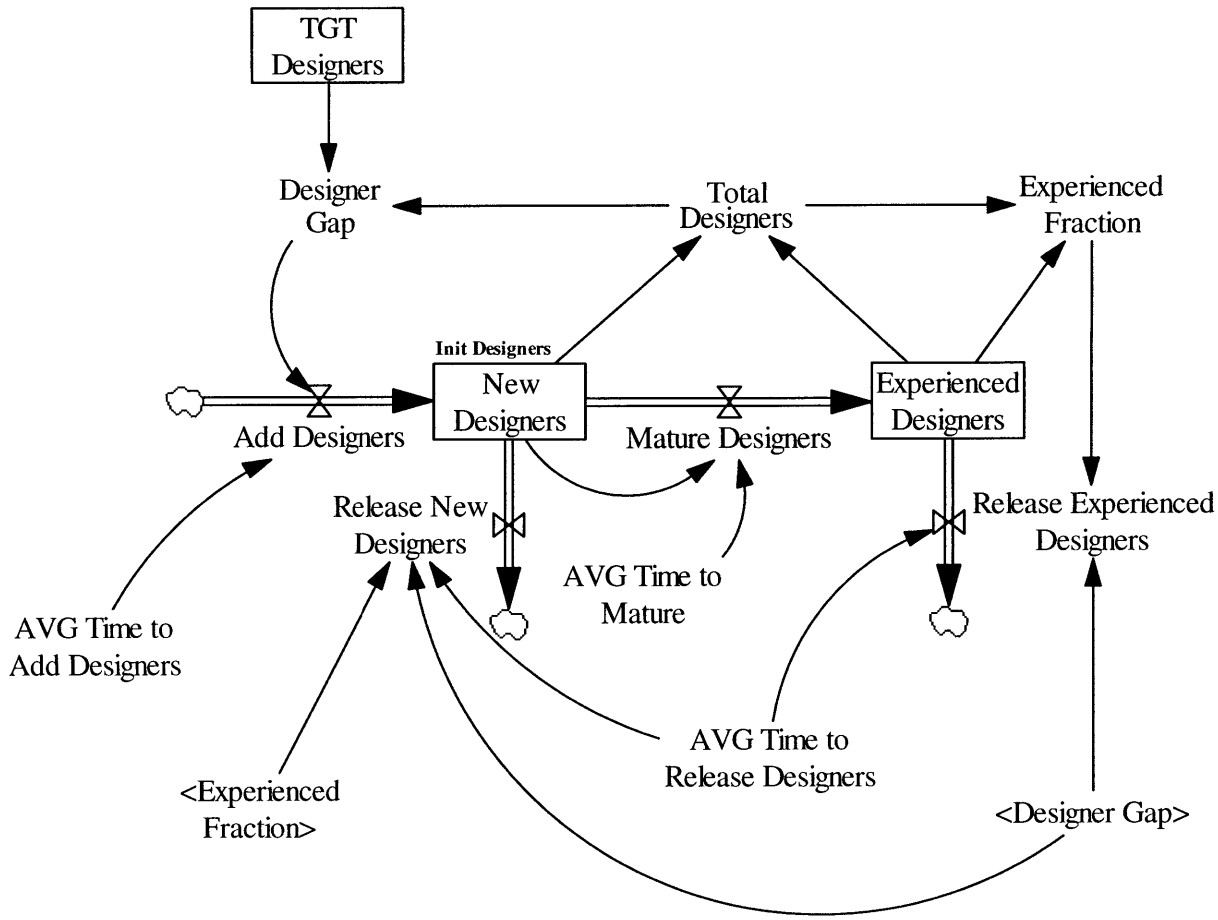


Figure 37 Experience Fraction

As you can see in Figure 37, this section of the Labor Sector consists of two main stocks - *New Designers* and *Experienced Designers* - which together represent the *Total Designers* available to the project team. Again, these stocks are subscripted by component and therefore they represent the designers who are assigned to work on the specific components (A or B) of the project. The third stock, *TGT Designers*, was previously discussed and represents the target number of designers the project team desires to have available for work.

The stock of *New Designers* is affected by adding new designers (*Add Designers*) to the team, releasing new designers (*Release New Designers*), and designers maturing to become experienced designers (*Mature Designers*). The model adds designers based on difference between the *Total Designers* and *TGT Designers* and adds them to the stock of *New Designers* through the inflow *Add Designers*. The rate at which it closes the gap is tempered by the *AVG Time to Add Designers* which represents the time it takes to find pull designers off of other projects or hire a new designer into the company. Similarly, the model releases designers when there are more designers available than are desired using the *AVG Time to Release Designers* as a means for tempering the outflows. It is assumed that the designers are released proportionally from both stocks (New and Experienced Designers).

New Designers mature into *Experienced Designers* using a simple first-order delay where the *Mature Designers* flow is set by simply dividing the number of designers in the stock of *New Designers* by the *AVG Time to Mature*, which is set at two months. Given the two stocks, it is easy to calculate the total number of designers available (*Total Designers*) and the fraction of the designers which are experienced (*Experienced Fraction*). These numbers are used to determine the average productivity (for design, redesign, and coordination) of the designers where it is assumed that a new designer works at 67% of the rate at which an experienced designer works.

DESIGNER ALLOCATION SECTOR

Given the number of designers that are available and their average productivity, this sector of the NPD model allocates designers among the different activities - design, redesign, and coordination – conducted directly by designers in the development stage of design work. While there are certainly other activities that compete for a designer's time, these three represent the major value added engineering activities most prominently discussed by designers in the client organization.

The basic framework for allocating designers is to always allocate the necessary designers to complete new design work first and then to allocate the remaining designers between the redesign and coordination work based on the relative demand for designers for each activity. Figure 38 depicts the relationship between the key variables included in the allocation of designers in the NPD model.

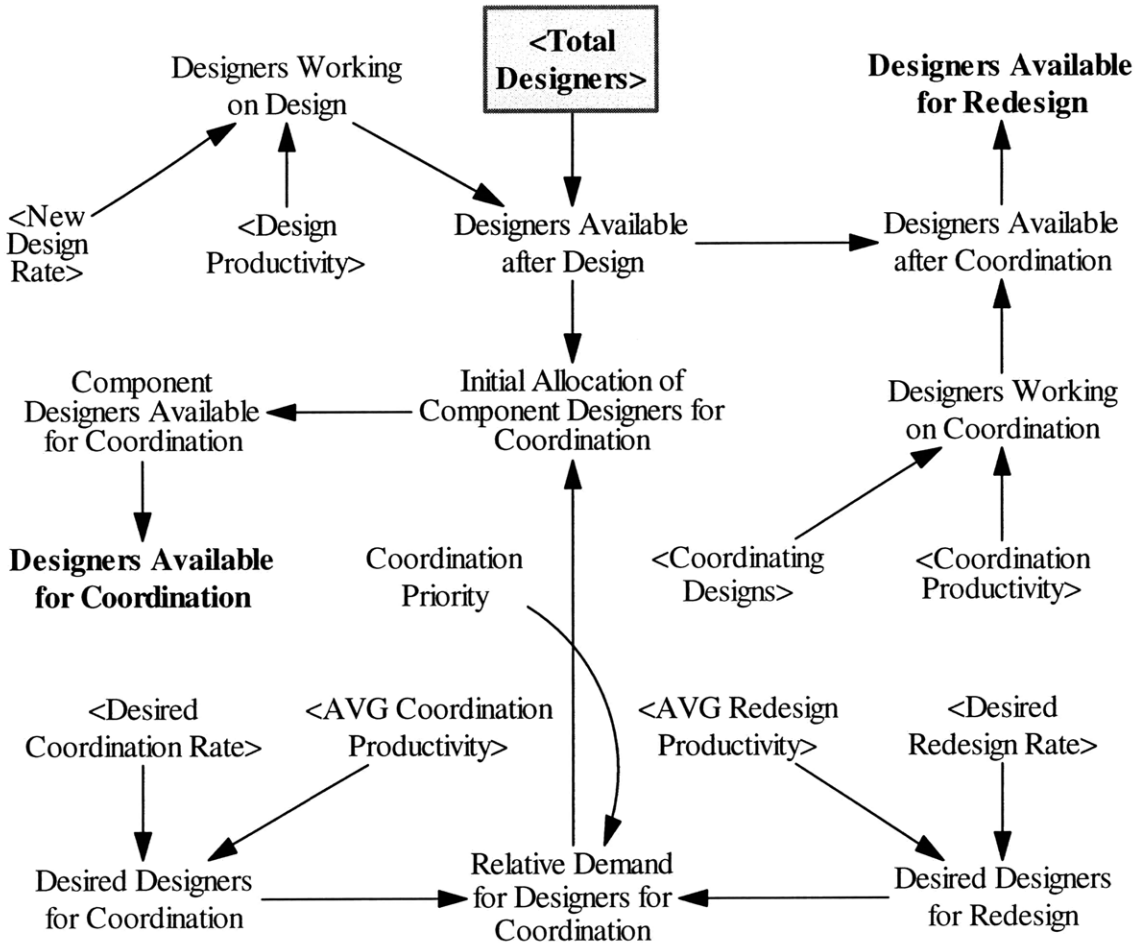


Figure 38 Designer Allocation

The allocation of designers for a given component in the NPD model begins with a given number of designers for each component based on the Labor sector described above. These designers are always allocated to complete new design work first. We can determine the number of designers who are actually working on new designs by dividing the *New Design Rate* by the *Design Productivity*. We can then subtract this number from the number of *Total Designers* to determine the number of designers available after new design work.

New Design Rate is the rate at which new designs are completed as formulated in the design sector of the model. *Design Productivity* is the rate at which a given designer is able to complete new designs. Its formulation will be discussed in the Productivity section of this chapter.

Given the number of *Designers Available after Design*, we need to allocate them between redesign work and coordination. The basic framework for doing so first allocates a number of designers for coordination work based on the relative demand for designers to do coordination versus redesign work. Then, once it is determined how many of these designers are actually working on coordination, the model allocates the remaining designers to do redesign work.

The number of component designers initially allocated for coordination work is calculated by taking the number of designers available after the new design work is accounted for and multiplying it by the relative demand for designers to do coordination.

The relative demand for designers to do coordination work is determined by comparing this demand to the total demand for designers to do coordination and redesign work using an US / (US+THEM) structure. Essentially, this formulation adds together the desired designers for coordination and redesign and then determines what fraction of the total desired designers are desired for coordination. The formulation also allows for a priority to be placed on coordination with respect to redesign. While the *Coordination Priority* is set at its default value coordination work and redesign work have equal priority. As the *Coordination Priority* increases, the weight placed on the demand for designers to do coordination increases resulting in a higher *Relative Demand for Designers for Coordination.* This in turn would result in more designers being

allocated to do coordination work. Conversely, as the *Coordination Priority* falls the relative demand for designers to do coordination falls resulting in fewer designers being allocated as such.

In order to determine the *Desired Designers for Redesign*, one simply divides the *Desired Redesign Rate* by the *Average Redesign Productivity.* However, to determine the *Desired Designers for Coordination* one must sum up the *Desired Coordination Rates* across components before dividing by the *Average Coordination Productivity.* This is necessary because an integration error attached to a design from the "other" component is still an error involving designs from both components and must be coordinated by designers from both in order to be fixed.

Once it is determined how many designers are available to do coordination work for a given component (*Component Designers Available for Coordination*), it is necessary to look across the components to determine the true number of designers available to do coordination. This is necessary because, as mentioned, coordination requires a designer from each component and one component may have fewer designers available than the other to conduct coordination. In this case, the number of designers that are truly available to do coordination is the minimum of the number available across the components.

The last piece of accounting in the allocation of designers is to determine the number of *Designers Available for Redesign.* We begin by determining the number of designers that are available after designers have been allocated for design work and coordination work. This number (*Designers Available after Coordination*) is calculated simply by taking the number of *Designers Available after Design* and subtracting the number of *Designers Working on Coordination.* Calculating the number of designers involved in coordination requires summing the rates at which

designs are being coordinated for both components since designers from each component are involved in the coordination of all of them. Dividing this summed rate of coordination by the *Average Coordination Productivity* determines the number of designers a given component has involved in coordination work at any point in time.

PRODUCTIVITY SECTOR

Given the allocation of designers among design, redesign and coordination work it is necessary to determine the productivity of the designers in completing each kind of work. The productivity sector of the NPD model is what determines the rate at which designers will design, redesign, and/or coordinate designs given the number and availability of designers and the amount of work to be done. At a basic level, it sets the various work rates in the model from a people perspective. The actual design, redesign, and coordination rates are determined in the design sector as the minimum of the work rate possible from people and the work rate possible from the work available. The reader can look at the formulation of *New Design Rate* as discussed in Appendix A as an example. In the sections that follow, each of the work rates (*Design Rate from People*, *Redesign Rate from People*, and *Coordination Rate from People*) and their formulations will be discussed.

Design Rate from People

The design rate from people represents the rate at which new designs can be completed based on the number of designers available, their productivity, the amount of design work to be done and the time remaining until the next build. This design rate is set at the minimum of the *Desired*

Design Rate and the *Capable Design Rate.* Figure 39 shows a schematic of the relationship between the key variables that determine the *Design Rate from People.*

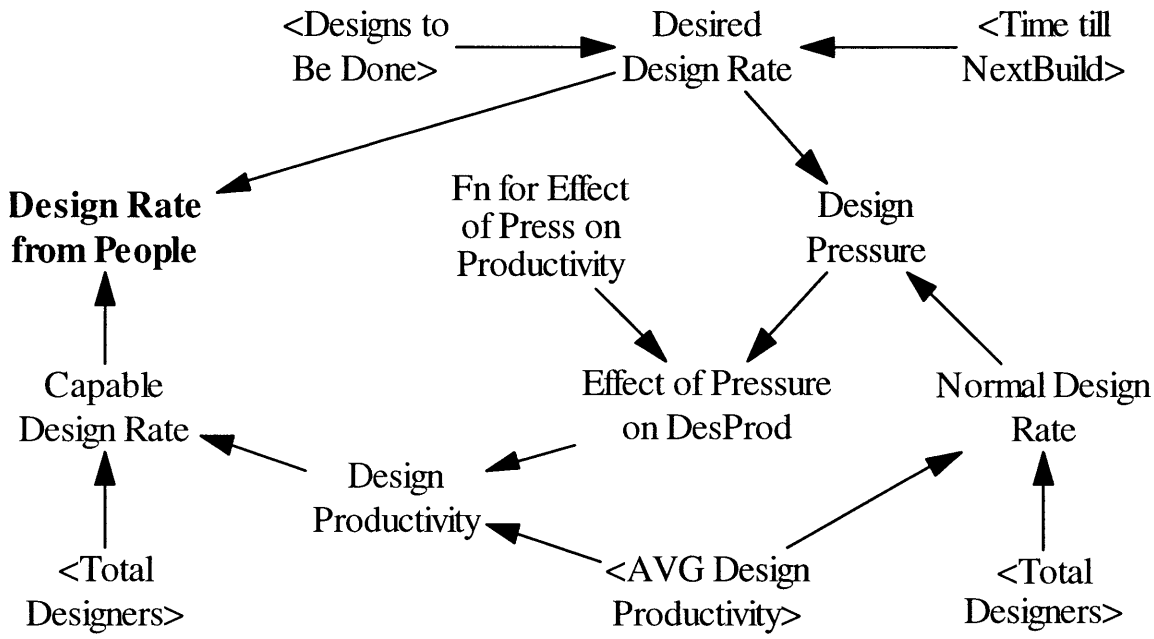


Figure 39 Design Rate from People

The *Desired Design Rate* is based on the number of designs to be completed and the time remaining until the next build event. It answers the question "How fast do we need to complete these designs in order to finish by the next build?" It can easily be calculated by dividing the number of designs still left in the stock of *Designs to Be Done* by the time left until the next build (*Time till NextBuild*). The *Capable Design Rate* is the rate at which new designs can be completed based on the number of designers available and their design productivity.

By setting the *Design Rate from People* to the minimum of these two variables the model recognizes two features of design work. First, designers aren't able to work any faster than they are capable of even if they have a strong desire to speed up. As you will see later in this section, designers are able to work faster than their normal productivity to an extent but at some point they can't work any faster. The second feature of design work captured is the fact that designers will work at a slower pace than what they are capable of if they have more than enough time available to get the required work done. In this case, the designers will use the extra time to work on other activities either related to the project at hand or to other projects to which they are assigned.

Design Productivity represents the rate at which an individual designer is able to complete new designs. It is based on the average design productivity and the effect that pressure has on that rate (*Effect of Pressure on DesProd*). Presumably as designers feel pressure to complete their designs prior to an upcoming build event they will work harder and faster, in some cases cutting corners, in order to get done on time.

The *Average Design Productivity* is based on the *Experienced Fraction* and the *Normal Design Productivity* which is set individually for the different components as the normal speed at which the components can be designed may differ between components based on the complexity of the component and the number and experience level of its available designers. For example, it might be easier (and therefore faster) to design the parts for a vehicle frame than it is to design the parts for the transmission. In the case of the NPD model, it is presumed that designers working on component A, on average, are able to design faster than those working on component B. The formulation for the differences in both work and test rates between the two components will be discussed in the Speed Factor section of this chapter.

The effect that pressure has on *Average Design Productivity* is to increase the rate at which new designs are completed as the design pressure increases. This effect is non-linear and reaches a maximum value of 1.667 times the normal productivity as depicted in Figure 40.

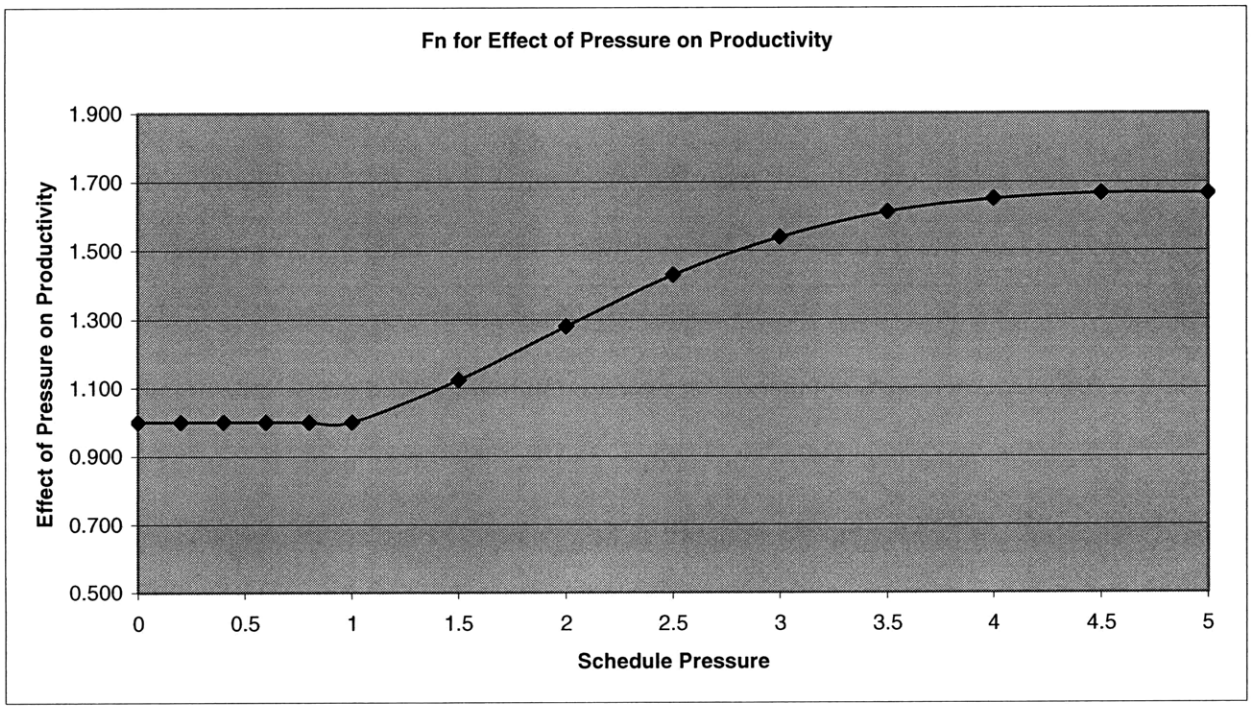


Figure 40 Effect of Pressure on Productivity

Design Pressure is a dimensionless variable used to represent the pressure felt by designers to complete design work as the *Desired Design Rate* increases relative to the *Normal Design Rate* which is the rate at which designs can be completed based on the number of designers available for new design work and their average design productivity.

Redesign Rate from People

The redesign rate from people represents the rate at which designs in the stock of rework can be redesigned based on the number of designers available to do rework, their average productivity,

the amount of redesign work to be done and the time remaining until the next build. Figure 41 provides a schematic of the relationship between the key variables used in determining the *Redesign Rate from People.*

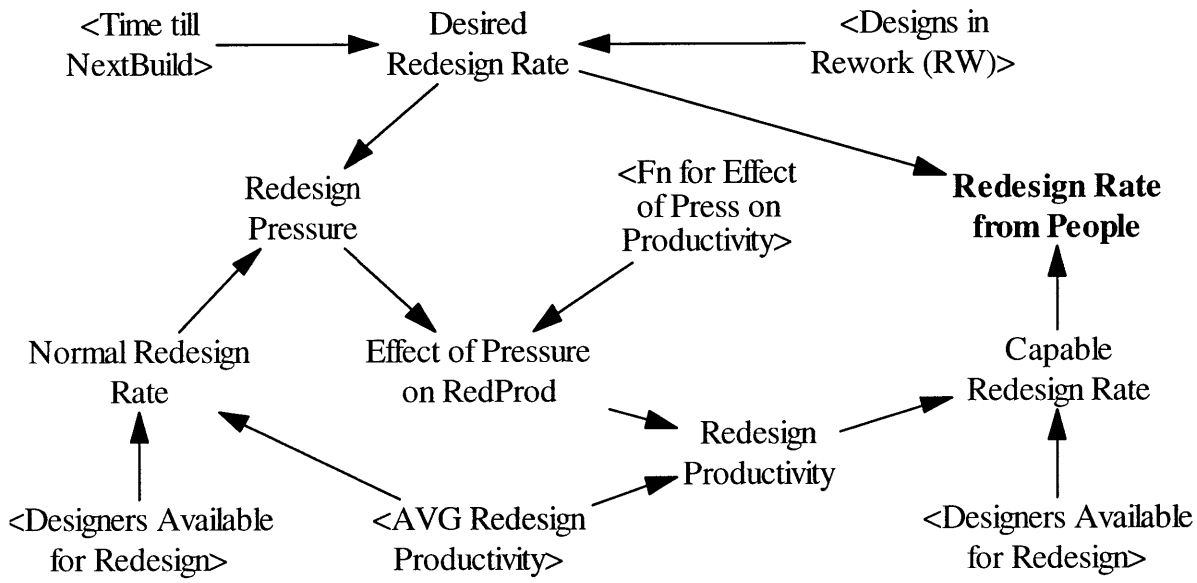


Figure 41 Redesign Rate from People

The formulation for *Redesign Rate from People* is completely analogous to the one described for *Design Rate from People* in the previous section. From the previous section we learned that the number of designers available for redesign work (*Designers Available for Redesign*) is determined based on a priority to always do new design work first and then to allocate the remaining designers between redesign work and coordination. You'll note that the same non-linear function used to determine the effect of pressure on *Design Productivity* (shown in Figure 40) is used to determine the effect of pressure on *Redesign Productivity.* However, in this case *Redesign Pressure* is entered into the function to determine the effect.

Coordination Rate from People

The coordination rate from people represents the rate at which designs with integration errors can be coordinated based on the number of designers available to do coordination, their productivity, the amount of coordination work to be done and the time remaining until the next build. Figure 42 provides a schematic of the relevant variables used in determining the *Coordination Rate from People.*

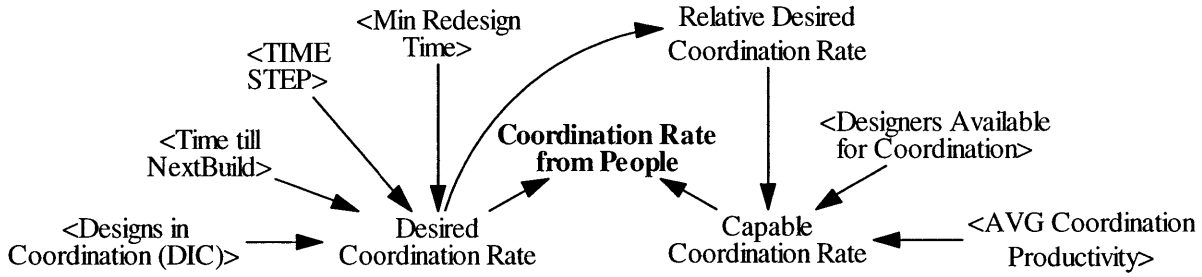


Figure 42 Coordination Rate from People

The basic formulation for the *Coordination Rate from People* is the same as the formulations used for the design and redesign rates above. The coordination rate from people is determined by taking the minimum of the *Desired Coordination Rate* and the *Capable Coordination Rate.*

However, in determining the *Desired Coordination Rate*, the model factors in an estimate of the time it will take to complete a redesign after a given design is coordinated. The formulation for *Capable Coordination Rate* also differs from the analogous rates in design and redesign. Because coordination requires designers from both components but only one component design is being

coordinated, it is necessary to allocate the coordination work between the two components. This is done using the variable *Relative Desired Coordination Rate* which prioritizes the allocation of designers available for coordination using an US / (US+THEM) structure. Essentially, this formulation adds together the desired coordination rates from both components and then determines what fraction of the total each component comprises and then allocates the available designers proportionally.

Lastly, while the *Average Coordination Productivity* may be different for the designers from the two components (A and B), it is not affected by pressure as was the case with design and redesign productivity. This formulation represents the reality that coordination involves two designers working closely together – not one faster than the other. Also, although one or both designers may feel pressure from a backlog of work and an impending build event it will certainly be different for both designers and thus when they come together to coordinate a design with an integration error they will maintain their normal coordination productivity and relieve their pressure elsewhere.

QUALITY SECTOR

The Quality sector of the NPD model establishes the quality of design and redesign work completed by designers both in terms of component quality and integration quality. The quality of the work done is what determines the rate at which errors are introduced into the designs and corresponding parts as outlined in the Design and Build sectors. The basic framework for setting the level of quality recognizes four separate quality inputs: the component quality of design

(*Design CQ*), the component quality of redesign (*Redesign CQ*), the integration quality of design (*Design IQ*) and the integration quality of redesign (*Redesign IQ*). A schematic of the relationship between these variables and those necessary to determine their values is included in Figure 43 .

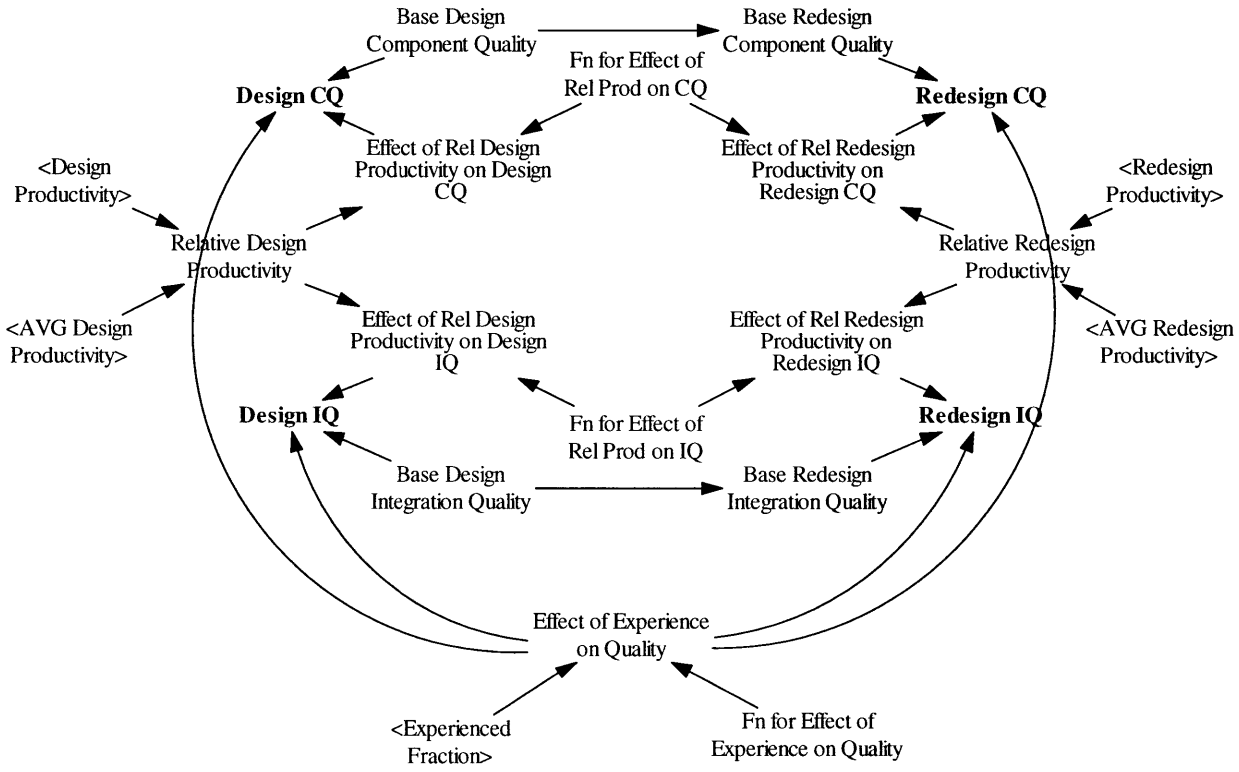


Figure 43 Quality

The formulation of the various values for quality is based on the premise that there is a base or normal quality of design (or redesign) that is affected by rate at which designers work relative to the normal work rate given the pressure of a work backlog and an impending build event. The result is that the actual quality of design (or redesign) can be considerably less than the normal quality when a designer speeds up their work rate, perhaps cutting corners, in an effort to clear their backlog before the next build. The formulation is quite similar to the one used to capture the

effect of work pressure on productivity. Secondly, the quality of design and redesign is affected by the experience level of the designers assigned to the project.

You'll recall that an increase in work pressure results in an increase in the productivity of individual designers with non-linear and ultimately diminishing returns. Designers work faster and are able to get more designs or redesigns done as there work backlog increases and/or the next build event approaches. However, this working faster comes at the expense of quality as designers will often cut corners in order to get things done more quickly. While the impact on quality affects both component quality and integration quality, the drop-off in integration quality is more severe as designers tend to become more isolationist in their design work as the build approaches. Their chief concern is to make sure they have their part ready for the build and care less about making sure it fits with the other parts. Similarly, a decrease in the experience level of the designers causes a corresponding decrease in the design and redesign quality.

Component Quality of Design

The component quality of design work (*Design CQ*) determines what fraction of new designs are completed without introducing a component error as discussed in the Design sector of the NPD model. *Design CQ* is calculated by taking the normal component quality of design work (*Base Design Component Quality*) and multiplying it by the effect of relative design productivity on design component quality (*Effect of Rel Design Productivity on Design CQ*). The base component quality for design is set at 0.85 for both components. This base quality means that when designers are working at their normal productivity pace 85% of their new designs will be completed without introducing a component error.

The effect of increasing the *Relative Design Productivity* (i.e. speeding up the rate at which designers complete designs) is a corresponding decrease in the design component quality and thus a decrease in the percent of new designs completed without a component error. The effect of the relative design productivity on component quality is determined using a table function which takes *Relative Design Productivity* as an input and returns a value between 0 and 1 as depicted graphically in Figure 44. *Relative Design Productivity* is calculated simply by dividing the actual *Design Productivity* by the *Average Redesign Productivity.*

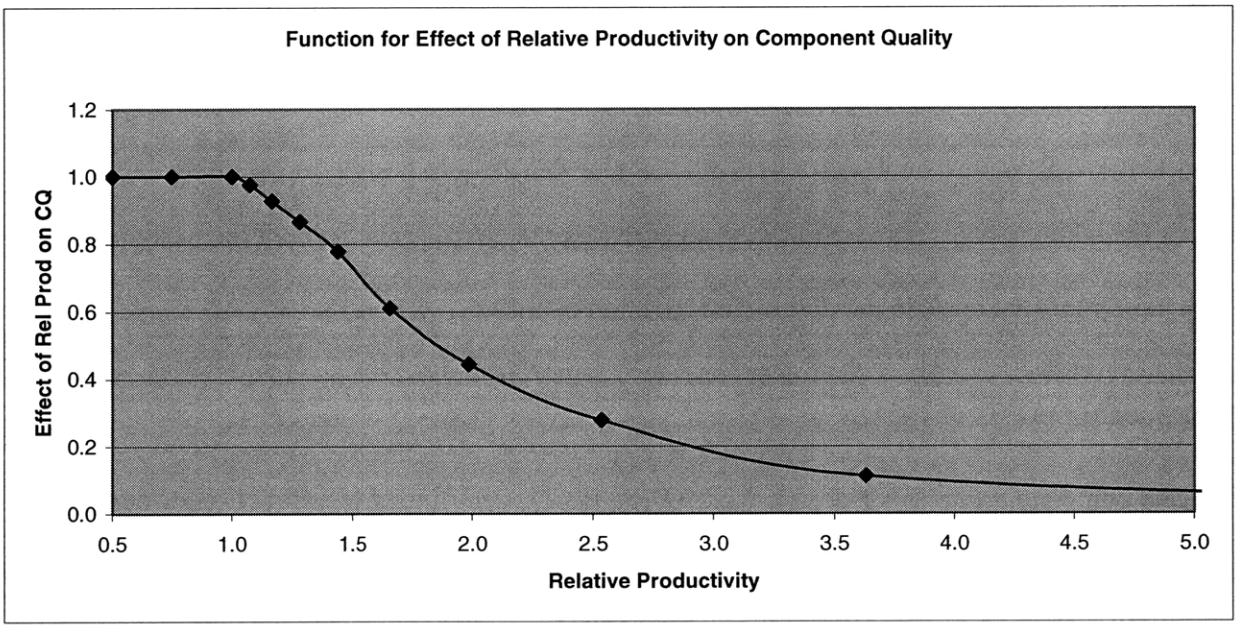


Figure 44 Effect of Relative Productivity on Component Quality

As you can see, the function is decreasing and S-shaped with the drop-off fairly severe as the relative design productivity increases from 1 to 2.5 with the deleterious effect saturating at a minimum value of 0 at a *Relative Design Productivity* of 12. As an example, the effect of speeding up the design productivity to three times the normal rate (*Relative Productivity* = 3) will cause the component quality to drop to less than 20% of the base component quality.

You'll note that even in the region of a low relative design productivity (<1) the effect on component quality never goes above 1; thus, *Design CQ* can never be higher than the *Base Design Component Quality*. In other words, reducing the rate at which designs are completed below the normal productivity level does not have a positive impact on component quality. This recognizes that designers with low *Design Pressure* will not slow down their individual design work (thus reducing their productivity below normal) but rather will work on other requirements within the new product development project (perhaps redesign or coordination) or on requirements from other projects to which they are assigned.

Similarly, the component quality of design is affected by the experience level of the designers. Just like the effect of relative productivity, the effect of experience is a fraction between 0 and 1 that is multiplied by the base quality to determine the effective quality. In this case, as the fraction of designers that are experienced drops the effect is to reduce the component and integration quality of design and redesign work. As depicted in Figure 45, as the fraction of experienced designers drops all the way to zero the effect on quality would be to decrease the quality to two-thirds (67%) of what it would be otherwise. On the other hand, even if all of the designers were experienced (i.e. Experience Fraction = 1) the quality would not be any higher than it would be otherwise based on the normal quality and any effects of relative productivity. The effect of experience as outlined above affects both component and integration, design, and redesign quality in the same way using the same function depicted in Figure 45.

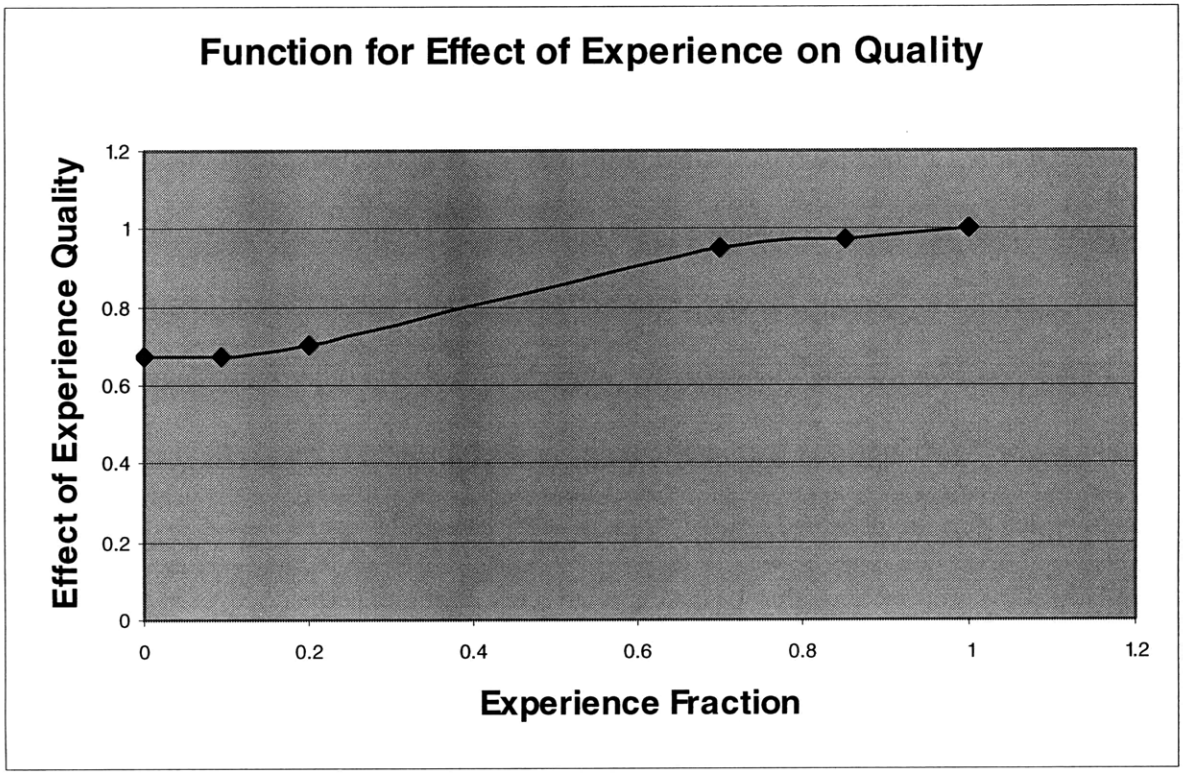


Figure 45 Effect of Experience on Quality

Component Quality of Redesign

The component quality of redesign (*Redesign CQ*) together with the *Redesign Rate* determines the rate at which known component errors are fixed in redesign and the rate at which new component errors are introduced during redesign work. It is formulated in the same manner as *Design CQ* except it uses a different base component quality for redesign *Base Redesign CQ* and uses *Relative Redesign Productivity* as the input to the table function that determines the effect of relative productivity on component quality. The same table function, depicted graphically in Figure 44, that was used for *Design CQ* is used to determine the effect of relative productivity on *Redesign CQ* as well.

The *Base Redesign Component Quality* is a function of the base design CQ. The formulation sets the value of *Base Redesign Component Quality* halfway between the *Base Design Component Quality* and 1. Given the model's default value of 0.85 for the *Base Design Component Quality*, the base value for redesign CQ is 0.925.

Integration Quality of Design

The integration quality of design (*Design IQ*) determines the fraction of new designs that are completed without introducing an integration error as outlined in the Design sector of the NPD model. The formulation for *Design IQ* is completely analogous to that used for *Design CQ* except that it uses a different *Base Design Integration Quality* and a different table function for determining the effect of pressure on integration quality.

Again, the *Base Design Integration Quality* is the same for both components (A and B) and is set in this case to 0.5. This means that when designers are working at their base rate of productivity they will introduce integration errors into 50% of their new designs. The table function used to determine the effect of relative design productivity is somewhat different than the one used for component quality. The table function used for integration quality is depicted graphically in Figure 46.

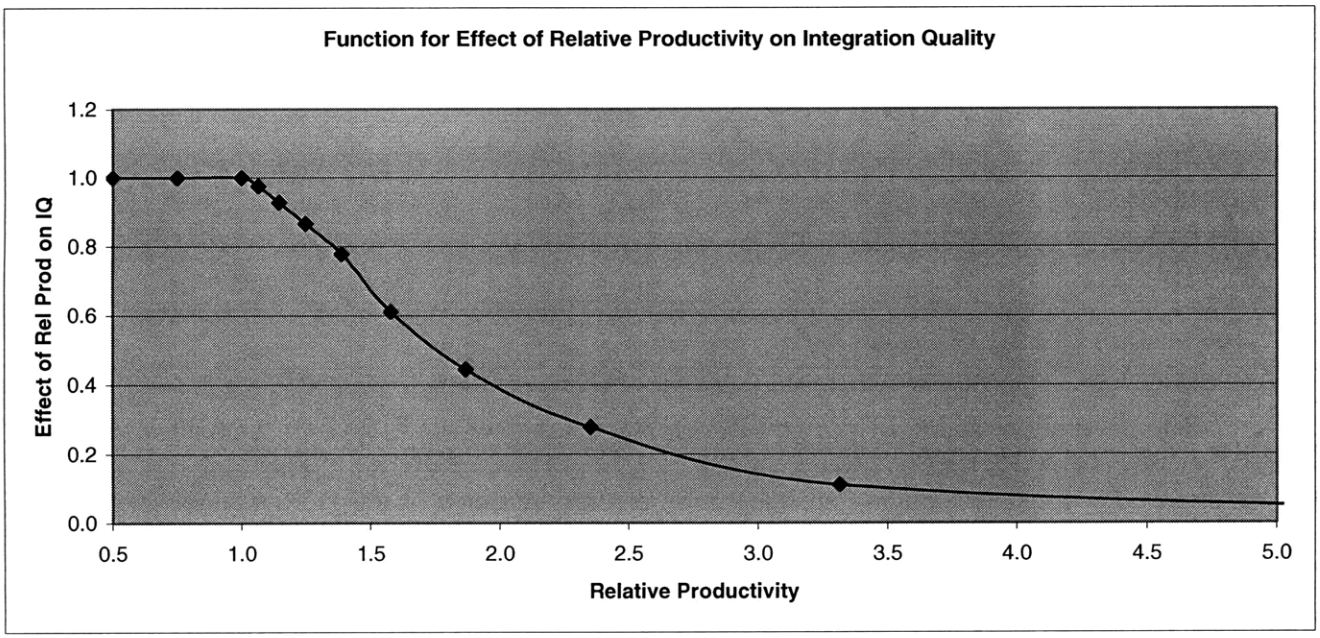


Figure 46 Effect of Relative Design Productivity on Integration Quality

You'll note that the general shape of the curve is the same as that for component quality. However, the drop-off is slightly more severe and sudden in the case of integration quality as one of the first places that designers cut corners when reacting to design pressure is to become more isolated in their work. The result is more of a concern to get their "own work" done and spend less time worrying about integrating their work (i.e. their designs) with other designers. As a result, the *Effect of Relative Design Productivity on Design IQ* falls below 0.4 when a designer is working at twice the normal productivity rate. As in the case of component quality, reducing the *Relative Design Productivity* below 1 does not result in an increase in the integration quality above its base value.

Integration Quality of Redesign

The integration quality of redesign (*Redesign IQ*) together with the *Redesign Rate* determines the rate at which coordinated integration errors are fixed in redesign and the rate at which new integration errors are generated during redesign work. It is formulated in the same manner as *Design IQ* except it uses a different base integration quality for redesign – *Base Redesign IQ* and uses *Relative Redesign Productivity* as the input to the table function that determines the effect of redesign productivity on quality. The same table function, depicted graphically in Figure 46, that was used for *Design IQ* is used to determine the effect of productivity on *Redesign IQ* as well.

Like the base component quality of redesign, the *Base Redesign Integration Quality* is set halfway between the *Base Design Integration Quality* and 1. At the default value for the NPD model, this sets the base value for redesign IQ at 0.75.

BENCH TEST FRACTION SECTOR

The *Bench Test Fraction* determines the fraction of designs and redesigns that are sent for bench testing at a given point in time. It represents the designers' willingness to send a design or redesign for bench testing prior to the next build event. There are two prevailing thought processes that play into the decision as to whether or not to bench test a design. First, because vehicle testing is presumed to be a more comprehensive, realistic and accurate test of a design's suitability some designers choose to wait until the next build event and vehicle testing rather than bench test their design. Secondly, as the next build event draws near some designers fear that there will be insufficient time to get bench test results and/or make the necessary corrections to their design

prior to the next build event. This fear results in fewer designs and redesigns being sent for bench testing as a build event approaches. As such, the bench test fraction (*BT Fraction*) is based on both the designers' intentions toward bench testing (*Intended Bench Test Fraction*) and the effect of redesign pressure (*Effect of Redesign Pressure on BT Fraction*) as depicted in Figure 47.

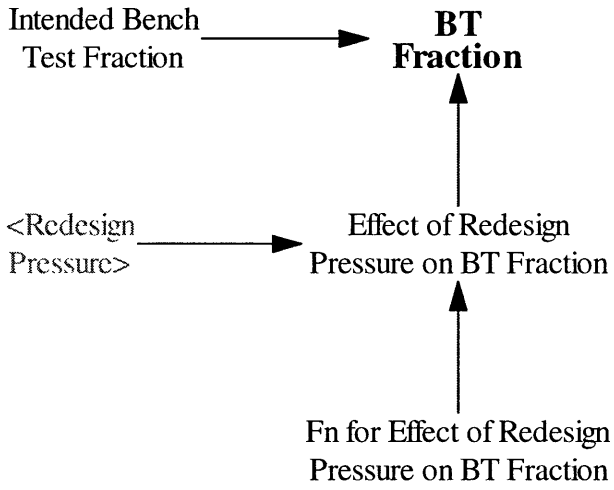


Figure 47 Bench Test Fraction

The bench test fraction is calculated simply by taking the *Intended Bench Test Fraction* and multiplying it by the effect of redesign time remaining prior to the next build (*Effect of Rd Time aBT on BT Fraction*). In the NPD model, the *Intended Bench Test Fraction* is set at 0.5 meaning that given ample time prior to the next build designers would send only half of their designs or redesigns for bench testing. The *Intended Bench Test Fraction* reflects the fact that designers are somewhat leery of the results of bench testing. Some believe that the bench test conditions are unrealistic and ignore errors found in bench testing until they are "verified" in vehicle testing..

Other designers fear that bench testing "over tests" their design and discover "false" errors and thus they avoid bench testing altogether. The *Effect of Redesign Pressure on BT Fraction* is a fraction between 0 and 1 which decreases non-linearly as *Redesign Pressure* grows. The effect represents the designers' unwillingness to send designs for bench testing when they are concerned that they might not have time to rework the design if it is found to have an error. Designers are very concerned about showing up to a build event with a part that has a known error in its design. The effect is captured using a table function depicted graphically in Figure 48.

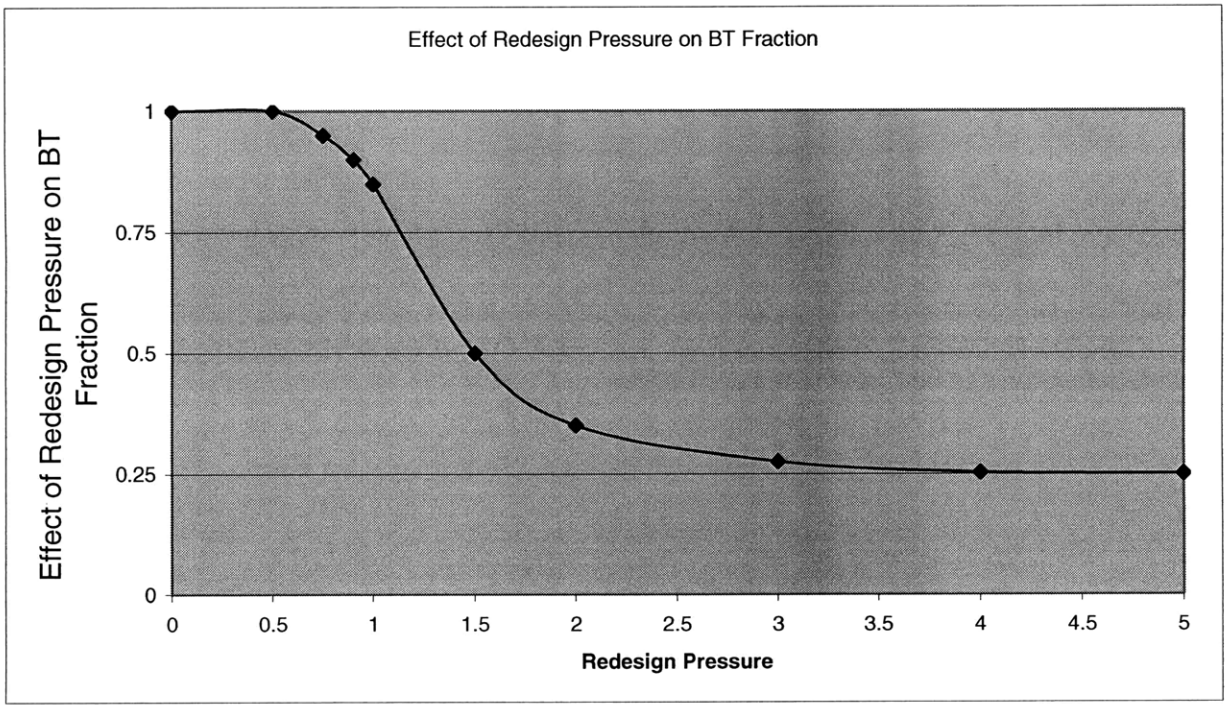


Figure 48 Effect of Redesign Pressure on Bench Test Fraction

When *Redesign Pressure* is equal to 1 (i.e. *Desired Redesign Rate* is equal to the *Normal Redesign Rate*) – the effect of on the bench test fraction is equal to 0.85 which means that the bench test fraction will be reduced to 85% of the *Intended Bench Test Fraction*. This means that

when designers are completing redesign work at a normal pace and meeting the demand (i.e. the Desired Redesign Rate) that there will still be some reluctance by designers to send their designs for bench testing out of fear that there will be insufficient time to rework a design found to have an error. It is only when the *Redesign Pressure* falls to 0.5 or below (when the desired redesign rate is only half of the normal redesign rate given available designers) that the *Effect of Redesign Pressure on BT Fraction* equals one meaning that designers will send the full *Intended Bench Test Fraction* of designs for bench testing. From the graph in Figure 48, it becomes clear that the effect of increasing redesign pressure becomes more severe in depressing the bench test fraction. However, the effect saturates at a value of 0.25 to reflect the fact that no matter how severe the redesign pressure some designers will send a portion of their designs to bench testing anyhow.

The table function for the *Effect of Redesign Pressure on BT Fraction* is included in Figure 49. This function is the same one that is graphed in Figure 48 above. As with all model variables, the full documentation can be found in the appendices.

Redesign Pressure	Effect of Redesign Pressure on BT Fraction
0	1
0.5	1
0.75	0.95
0.9	0.9
1	0.85
1.5	0.5
2	0.35
3	0.275
4	0.2525
5	0.25
10	0.25

Figure 49 Table Function for Effect of Redesign Pressure on BT Fraction

COORDINATION FRACTION SECTOR

The *Coordination Fraction* determines the fraction of designs found to have both a component and an integration error that are sent for coordination prior to being redesigned. The act of coordinating a design with an integration error requires a designer from each of the coupled components (A and B). Together the designers will coordinate the rework required in order to fix the integration error. The outcome of coordinating a design with an integration error is to send either just the design with the attached integration error or both designs from the coupled "design pair" to the stock of *Designs in Rework (RW).*

The *Coordination Fraction* represents the willingness of a designer to delay rework on a known component error until he or she is able to link up with a designer from the coupled component to coordinate the integration error. Similar to the bench test fraction, there are two

factors that determine the *Coordination Fraction.* The first factor is the fraction of designs with both errors that designers would send for coordination if they had ample time until the next build to both coordinate and redesign the given design. This is represented in the NPD model by the variable *Intended Coordination Fraction.* The second factor is the amount of coordination pressure that exists at the time the design is found to have both an integration error and a component error. This represents the designers' unwillingness to put off reworking a design to fix a component error for too long while waiting to coordinate an integration error. A schematic of the relationship between the variables relevant in determining the *Coordination Fraction* is included in Figure 50.

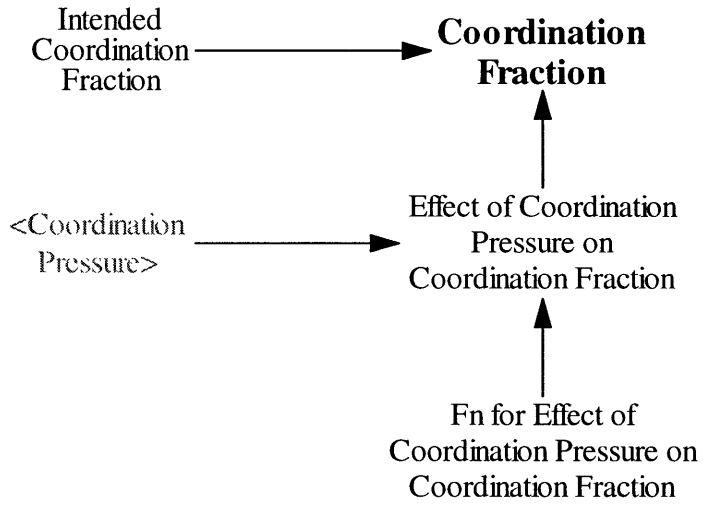


Figure 50 Coordination Fraction

The basic formulation for *Coordination Fraction* is identical to that of the bench test fraction described above. However, in this case the *Intended Coordination Fraction* is set at 1. This is considerably higher than the *Intended BT Fraction* set at 0.5 because in this case there is a known integration error that must be fixed and in order to fix it the design must be coordinated. You'll

recall that in the case of bench testing, designers do not have a known error prior to testing and are leery of the results of bench testing.

In addition to a higher *Intended Coordination Fraction*, the function used to determine the effect of redesign time available after coordination on the *Coordination Fraction* is different than the one used for the *BT Fraction*. It, too, decreases non-linearly as the Coordination Pressure increases. However, as you can see in Figure 51 the rate at which the effect decreases the *Coordination Fraction* as the pressure increases is more severe than that for *BT Fraction* in Figure 48. The effect decreases the *Coordination Fraction* both sooner and more dramatically as *Coordination Pressure* increases.

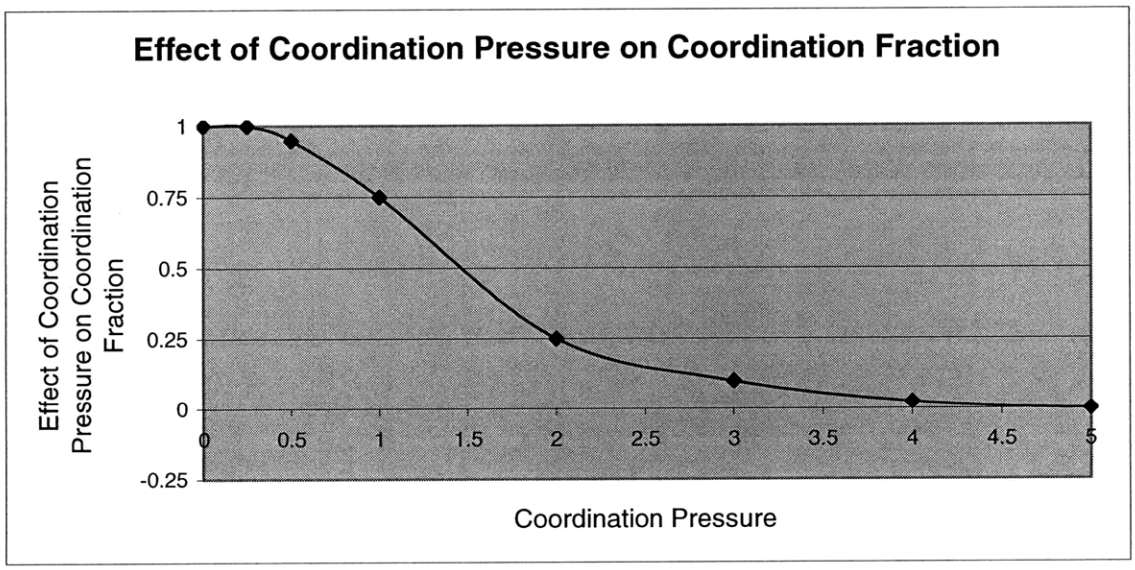


Figure 51 Effect of Coordination Pressure on Coordination Fraction

The effect of pressure is more severe in the case of the *Coordination Fraction* because in addition to the integration error associated with a given design the designer has a known

component error that he must also try to fix prior to the next build. Thus, as the coordination pressure increases, the likelihood of getting a design with an integration error coordinated with sufficient time to fix it and its known component error prior to the next build goes down; and with it, the fraction of designs with both types of errors sent for coordination goes down as well. This reflects a feeling among designers that a designer who brings a part to a build with a known component error has no one else to blame but himself while the blame for a build part with a known integration error can at least be divided between the designers from the coupled components.

When *Coordination Pressure* is equal to 1 (i.e. *Desired Coordination Rate* is equal to the *Capable Coordination Rate*) the effect on the *Coordination Fraction* is equal to 0.75 which means that the fraction of designs with both types of errors sent for coordination will be reduced to 75% of the *Intended Coordination Fraction*. When the *Coordination Pressure* falls to 0.5 (when the *Desired Coordination Rate* is half the *Capable Coordination Rate* given the available designers) the effect is equal to 0.95 meaning that the *Coordination Fraction* will be at 95% of its intended fraction. It isn't until the *Coordination Pressure* falls to 0.25 that the *Coordination Fraction* equals the *Intended Coordination Fraction*. You'll note that the effect falls all the way to zero at a *Coordination Pressure* of 5 which means that when the *Desired Coordination Rate* is five times the *Capable Coordination Rate* the *Coordination Fraction* will fall to zero meaning no designs with both types of errors will be sent for coordination prior to redesign.

The table function for the *Effect of Coordination Pressure on Coordination Fraction* is included in Figure 52.

Coordination Pressure	Effect of Coordination Pressure on Coordination Fraction
0	1
0.25	1
0.5	0.95
1	0.75
2	0.25
3	0.1
4	0.025
5	0
10	0

Figure 52 Table Function for Effect of Coordination Pressure on Coordination Fraction

The last piece of model formulation that is relevant to the Coordination Fraction is the computation for Coordination Pressure. In much the same manner as Redesign Pressure, it is calculated by dividing the Desired Coordination Rate by the Capable Coordination Rate.

EXTERNAL REWORK SECTOR

External rework is the rework that is generated for a given component when it "shares" an integration error with a design in a coupled component. By convention in the NPD model, the integration error is attached to only one of the designs in coupled "design pair." During the process of coordinating the required rework necessary to fix the integration error, the two designers (one from each component in the "design pair") determine if one or both of the designs must be redesigned. By convention, the design with the attached integration error will always require

rework. With some probability the second design in the "design pair" will require rework as well. External rework refers to the redesign work required of this second design.

Accounting for external rework involves two processes. The first is to determine how much external rework to assign to a given component. The second is to determine which designs to move into the stock of *Designs in Rework (RW).* Figure 53 shows a schematic of the relationship between the key variables necessary to account for external rework.

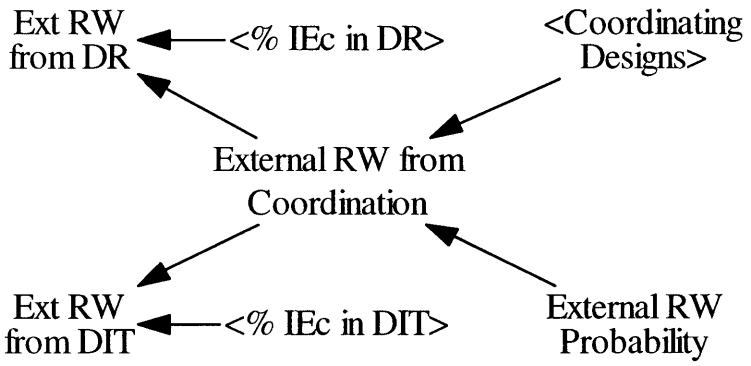


Figure 53 External Rework

Determining the amount of external rework to assign to a given component is fairly straightforward. In the NPD model it is assumed that as designs with integration errors are coordinated that external rework will be generated with a fixed probability. Because the external rework is assigned to the "other" component it is necessary to use two separate equations to determine the rate at which external rework is assigned to the respective components. By simply multiplying the rate at which designs from the coupled component are coordinated (*Coordinating Designs*) by the probability that both designs will require rework (*External RW Probability*), we

can calculate the rate at which external rework is assigned to a given component (*External RW from Coordination*).

Once we know the rate at which external rework is assigned to a given component, we need to determine which designs will be sent for rework. The first step in doing so is to recognize that the designs we send for external rework, by definition, can not have integration errors. This is true because the integration error in question is attached to the other design in the coupled design pair. However, the designs we send for external rework may (or may not) have a component error, known or unknown, at the time we send it for rework. As such, we need to determine the relative densities of designs without integration errors in the stocks of designs in progress. We refer to these designs as being "IE Clean." Since by definition designs in coordination (*DIC*) must have an integration error we do not need to consider this stock. Also, since the IE Clean designs that are already in the stock of rework do not need to be moved if we assign them external rework we do not need to explicitly determine the rate at which external rework is assigned to these designs. That leaves us with assigning external rework to designs in the stocks of designs released (*DR*) and designs in test (*DIT*). The formulation takes the rate at which external rework is assigned to a given component and multiplies it by the percentage of IE Clean designs for that component that reside in either designs released (DR) or designs in test (DIT). These rates, *Ext RW from DR* and *Ext RW from DIT*, are used to move designs and their associated component errors to the stock of rework as outlined in the design sector. The formulation for *% IEc in DR* and *% IEc in DIT* are included in the technical documentation.

PHANTOM WORK SECTOR

Phantom work is the term used to represent potentially unnecessary work that is done during the course of a development project. This phantom work is created from a mismatch in the speeds at which components can iterate their design work. The amount of phantom work can vary greatly depending on the degree to which the components design iteration speeds are mismatched, the degree to which the components are coupled, and the build event timing.

There are two types of phantom work that we characterize in the NPD model. The first is the creation of *phantom integration errors*. These represent integration errors that are created from previous integration errors that have been discovered, coordinated, found to require rework of both designs in the coupled "design pair", but only one of the two redesigns is completed by the next build event. The second type of phantom work is called *phantom rework*. It represents the rework done primarily on designs with phantom integration errors after a build event has occurred but before the new "phantom" error is discovered. Each of these types of phantom work will be discussed in more detail below and the model structure used to account for each will be described.

Phantom Integration Errors

Phantom integration errors are "new" integration errors created from old ones because one of the two coupled designs requiring rework did not get redesigned in time for the next round of vehicle testing (i.e. the next build). These are not the same as old integration errors that "carry over" to the next build because they were redesigned but not fixed. Nor are they the same as integration errors that reoccur on the next build because none of the required rework was completed in time.

Phantom integration errors occur when the coordination of an integration error results in a requirement for both designs from the coupled component "design pair" to be redesigned and one of the two redesigns is completed before the next build event and the other is not. When this happens, a new integration error will be discovered in vehicle testing. This integration error exists between the new version of the redesigned part and the old version of the coupled part that was not redesigned. Because this "design pair" is different and indeed more current than the old "design pair" that had the original integration error, the integration error discovered in the subsequent build will be considered a new integration error.

At a minimum the new (or phantom) integration error will need to be coordinated – requiring designers from both components – in order to determine the root cause of the new integration error and what rework will be required to fix it. In the worst case, the designers may determine that both designs will need to be reworked in order to fix the new integration error.

We refer to these new integration errors as *phantom* errors because if the slow iterating component had had enough time to complete its redesign prior to the second build and it did so correctly, the new integration error would not have occurred.

Tracking how many phantom integration errors occur during the course of a development project follows from the integration error accounting done in the Design Sector. Accordingly, discussion of the formulation of variables used from the Design Sector used will be limited and the reader can refer to the previous description in the Design Sector. The stock and flow structure used to account for phantom integration errors (*Phantom IE*) is depicted in Figure 54.

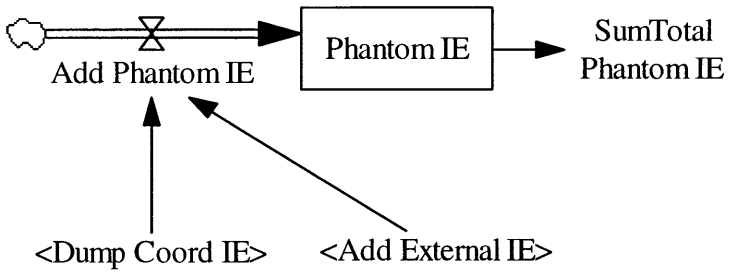


Figure 54 Phantom Integration Errors

The stock of *Phantom IE* accumulates the phantom integration errors as defined above for a given component over the course of the new product development project. It has a single inflow – *Add Phantom IE* – and no outflows. The equation for *Phantom IE* and its inflow are included below. You'll note that both *Dump Coord IE* and *Add External IE*, which serve as inputs to the inflow, are flows from the integration error co-flow in the Design Sector. As expected, the initial value of the stock of *Phantom IE* is zero. Lastly, the variable *SumTotal Phantom IE* represents the total number of accumulated phantom integration errors across all components.

Phantom Rework

Phantom rework refers to the potentially unnecessary redesign work that is done during the course of a development project. There are two main sources of phantom rework. The first source of phantom rework is the redesign work done after the build event has occurred and vehicle testing has started on designs with coordinated integration errors that required both designs from the coupled design pair to be reworked and only one of the redesigns was done at the time of the build. This phantom rework can be done on designs that had the integration error attached to them or on

designs that had been designated for external rework. Essentially, this phantom rework is the redesign work done on designs with phantom integration errors, as defined above, before the phantom integration errors are discovered. As such, the phantom integration errors cannot be fixed in redesign and will need to be coordinated and redesigned again upon discovery. Thus any rework done on these designs after the build but prior to discovering the phantom IE is potentially unnecessary since it will have to be done again. The second source of phantom rework is the external rework that is required of a component when a phantom integration error attached to a design from a coupled component is coordinated. In this case, because of the way that phantom integration errors are defined and generated in the model, we know that external rework assigned to a component from the coordination of a phantom integration error represents rework that it being done for the second time on the same design and could have been avoided had the phantom integration error not been generated.

It is necessary to create some new structure to account for the phantom rework (*Phantom RW*) as defined above. However, as in the case of accounting for *Phantom IE* we will be able to make use of existing model structure from the integration error co-flow in the Design Sector. Figure 55 shows the model structure and variables needed to track the *Phantom RW* as it is generated during the course of a project in the simulation.

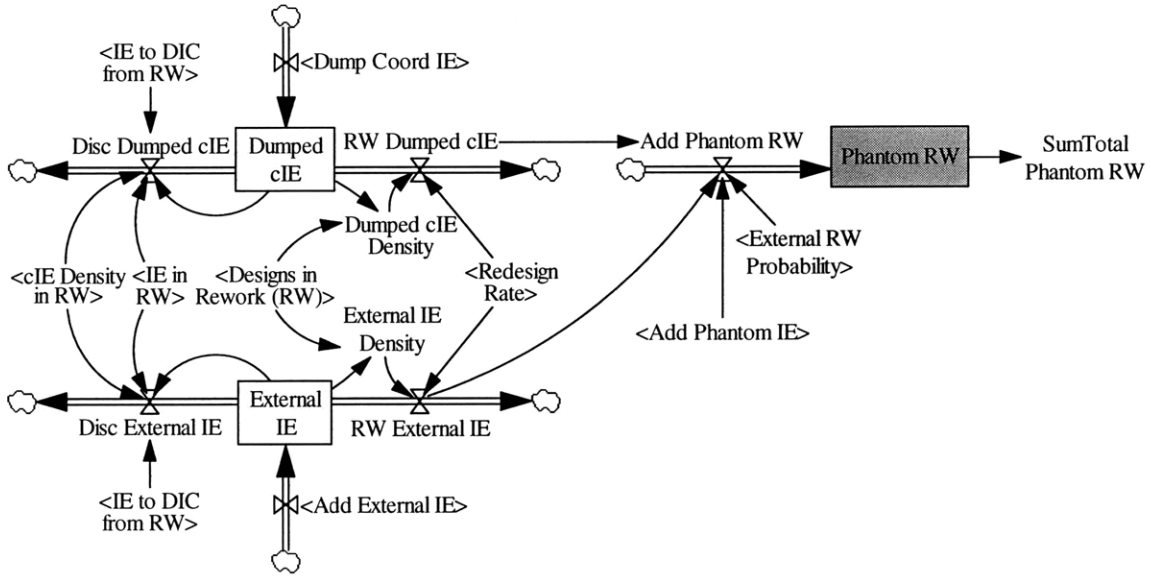


Figure 55 Phantom Rework

As you can see in Figure 55, the stock of *Phantom RW* has only one inflow which is *Add Phantom RW*. However, as mentioned above there are two primary sources for generating phantom rework. The first is reworking designs with outdated integration errors. This type of phantom rework is represented by the flows *RW Dumped cIE* and *RW External IE* which are both inputs to the inflow *Add Phantom RW*. The second source of phantom rework is the external rework that is generated in coordinating phantom integration errors. While this phantom rework is not done until some time after the phantom integration errors are discovered and coordinated, it is added to the stock of *Phantom RW* at the time the phantom integration errors are added to the stock of *Phantom IE*. This process is represented by the use of *Add Phantom IE* and External *RW Probability* as inputs to the inflow *Add Phantom RW*.

The stock of *Phantom RW* begins with an initial value of zero and accumulates the inflow *Add Phantom RW*. There are no outflows. Adding phantom rework to the stock requires a different

equation for each component (A and B) as the inflows are dependent on the number of phantom integration errors being added to the coupled component. *RW Dumped cIE* represents the rate at which designs with coordinated integration errors that have been dumped are redesigned. *RW External IE* represents the rate at which designs with external integration errors are redesigned. A description of how these rates are determined is included below. The last input, Add *Phantom IE * External RW Probability*, represents the additional external rework that will be required based on the coordination of designs with phantom integration errors. The sum of these three determines the rate at which designs are added to the stock of *Phantom RW*. See the technical documentation for a full description of how these rates are formulated in the model.

Phantom Work Accounting

Given the amount of phantom work that is done in terms of the number of designs that have been reworked unnecessarily (phantom rework) and the number of phantom integration errors, we can calculate the number of months of designer work that are spent during the course of a development project on phantom work. This represents real work done by designers that could potentially have been avoided if the mismatch in design iteration speeds between components was eliminated and/or the timing of the build events was matched more closely to the slow iterating component.

To calculate the amount of phantom work done by designers in terms of time spent on unnecessary rework (*Phantom RW ManMonths*) we simply take the cumulative number of designs in the stock of *Phantom RW* and divide it by the *AVG Redesign Productivity.*

To calculate the amount of phantom work done in terms of the designer time spent on coordinating phantom integration errors (*Phantom IE ManMonths*) we take the cumulative number of errors in the stock of *Phantom IE* and divide it by the *AVG Coordination Productivity*. Since coordinating an integration error requires two designers – one from each of the coupled components – we must multiply this quotient by two.

In order to calculate the total amount of designer time spent on phantom rework (*SumTotal Phantom ManMonths*) we must first sum the designer time spent on phantom rework and phantom integration errors across the components and then simply add these two sums together to determine the total amount of designer time spent on phantom rework.

BUILD SWITCHES

This sector of the model handles two important structural issues in the NPD model. First, it establishes the dates of scheduled builds and turns them on and off as appropriate. Secondly, it ensures that no more than two builds are active at any point in time and provides the means for distinguishing between the two active builds - the "current build" versus the "previous build" as described in the Build Sector. The model structure and formulation for each will be discussed below.

Build Scheduling

Build scheduling determines how many builds will occur during the course of a given simulation of the new product development project and establishes the dates of those builds. During the course of the simulation, the build scheduling structure turns builds on and off as

appropriate based on the dates of the scheduled builds and the current simulation time. The build scheduling structure also turns off builds and the corresponding vehicle testing when the number parts remaining for testing falls below a given threshold. Lastly, the build scheduling structure provides accounting for some of the necessary inputs to decision processes and model structure outlined previously. For example, it tracks the amount of time until the next scheduled build which is used to calculate the Desired Design Rate. A schematic of the relationship between the relevant variables needed for build scheduling is included below in Figure 56.

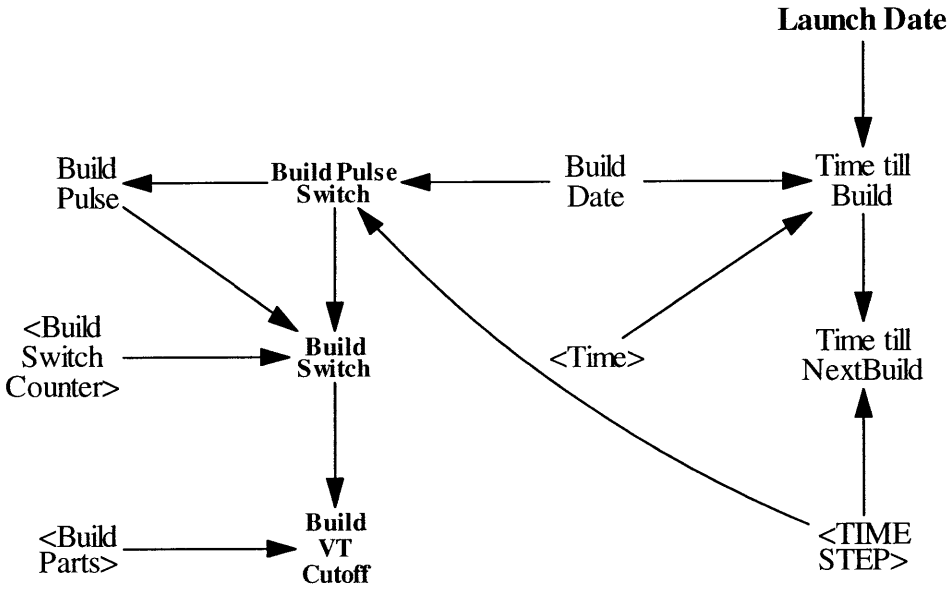


Figure 56 Build Scheduling

The most important variable in build scheduling is obviously the Build Date. It establishes when the builds will occur. In the NPD model, the variable Build Date is subscripted for the different builds allowing each build to have its own build date. There are ten (10) build dates corresponding to the possible vehicle builds built into the model: Proto, Proto2, Proto3, Proto4, DIB, DIB2, DIB3, FPE, and FPE2. Given that the normal simulation run time is 50 months, by

setting the start dates for certain builds at 1e+006 (month 1,000,000) in the base case we essentially remove these builds from the simulation and leave Proto, DIB and FPE as the three active builds in the base scenario. However, in model testing we are able to add builds to the simulation simply by resetting their build date to a time within the simulation time bounds.

The *Build Pulse Switch* pulses to a value of 1 at the time of given build's *Build Date* for a single time step. The variable *Build Pulse* then sums these values across all builds resulting in a variable that pulses to a value of 1 for a single time step when any build event occurs. The variable Build Pulse is used in a number of places in the model to stop the rates of flow for stocks in both the design and build sector as previously described.

Different than the *Build Pulse Switch* which is turned on only at the instant of a build event, the *Build Switch* is turned on (i.e. equals 1) and stays on throughout the time a given build is active (i.e. while vehicle testing is ongoing); else it is equal to zero. By definition, a build is active if it is either the current build or the previous build. Thus, a simple formulation for *Build Switch* is to take the maximum value of the *Current Build Switch* and the *Previous Build Switch*. As the names imply, these switches are turned on (equal to 1) when a given build is active and is the current or previous build, respectively. These switches will be discussed in detail in the Active Builds section below.

A switch is used to cutoff vehicle testing when the number of build parts remaining to be tested falls below a certain threshold. This is done to reflect the fact that vehicle testing is shut down in the client company when further vehicle testing yields little results. This is accomplished in the model by using the IF THEN ELSE function in VENSIM to cut off vehicle testing by setting the

value of *Build VT Cutoff* to zero when the number of build parts remaining to be tested falls below 0.005 as outlined in equation.

The use of *Build Switch* as a multiplier ensures that vehicle testing is shutoff for a given build whenever it is not an active build. The *Build VT Cutoff* variable is subscripted for both the different components and the different builds. This is because vehicle testing on a given build can be shut down for the different components separately.

The next piece of accounting handled by the Build Scheduling sector of the model is calculating the time remaining until a given build (*Time till Build*) and, from this, the time until the next build (*Time till NextBuild*).

The formulation for Time till Build handles a couple of scenarios. First, while the current time (i.e. simulation time) is less than the build date for a given build the time until the build is calculated simply by subtracting the current time (Time) from the Build Date. When the build date for a given build has passed (Time> Build Date), then the Time till Build is set to be the time remaining until product launch (Launch Date – Time). The launch date of the product is set at a default value of 27 months in the simulation to reflect the new product development timeline of the client company. However, when the launch date has also passed, resulting in a negative amount of time remaining, the model sets the value of *Time till Build* at 0 using the MAX function.

The variable *Time till NextBuild* represents the time remaining until the next scheduled build event. It is the variable used in setting the desired completion rates for design, redesign, and coordination rates as discussed previously. It is also used in determining the fraction of designs

and redesigns that are sent for bench testing as well as the fraction of designs with both component and integration errors that are sent for coordination. Calculating the time until the next build is simple in that it is merely the minimum value of the time remaining until each of the remaining builds.

Active Builds

In order to account for the active builds and to distinguish between the current and previous builds, the NPD model uses a separate stock which serves as a build counter. A depiction of this stock used as a counter and its associated inflow and outflow are included in Figure 57.

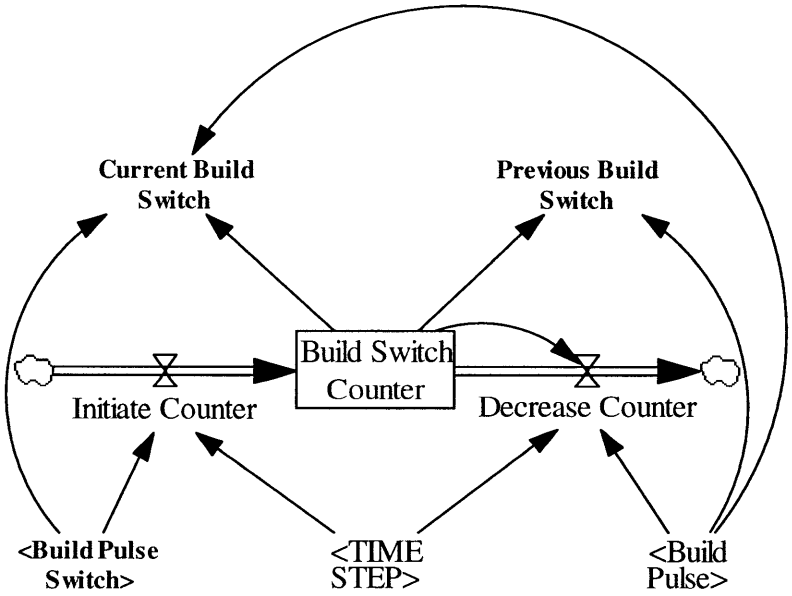


Figure 57 Build Switch Counter

There is a build counter for each build which is initiated at a value of 2 at the build date. This is done via the inflow *Initiate Counter*. The stock is depleted via the outflow *Decrease Counter* which decreases the counter by 1 at each subsequent build event (Build Pulse/TIME STEP) while

the *Build Switch Counter* is greater than zero. This formulation results in a build switch counter which initiates at a value of 2 at the start time of the given build, decreases to 1 at the time of the next build, and finally decreases to 0 at the second subsequent build.

Given the formulation of the *Build Switch Counter*, it isn't difficult to distinguish between the current and the previous builds. Two switches are used in the NPD model that turn on when a given build is considered the current build (*Current Build Switch*) or when it is considered the previous build (*Previous Build Switch*). A given build is considered the current build from the time of its build date until the date of the next build event. A given build is considered the previous build beginning on the date of the build that follows it up until the date of the subsequent build. In other words, Build 1 is considered the previous build beginning on the date of Build 2 and up until the date of Build 3 at which time it is no longer considered an active build.

SPEED FACTOR

The Speed Factor sector of the NPD model sets the relative speed at which the different components (A and B) are able to complete their activities. This includes design and redesign work as well as bench testing and vehicle testing. Figure 58 provides a schematic of the relationship between the variables used to set the relative speeds of the two components' activities.

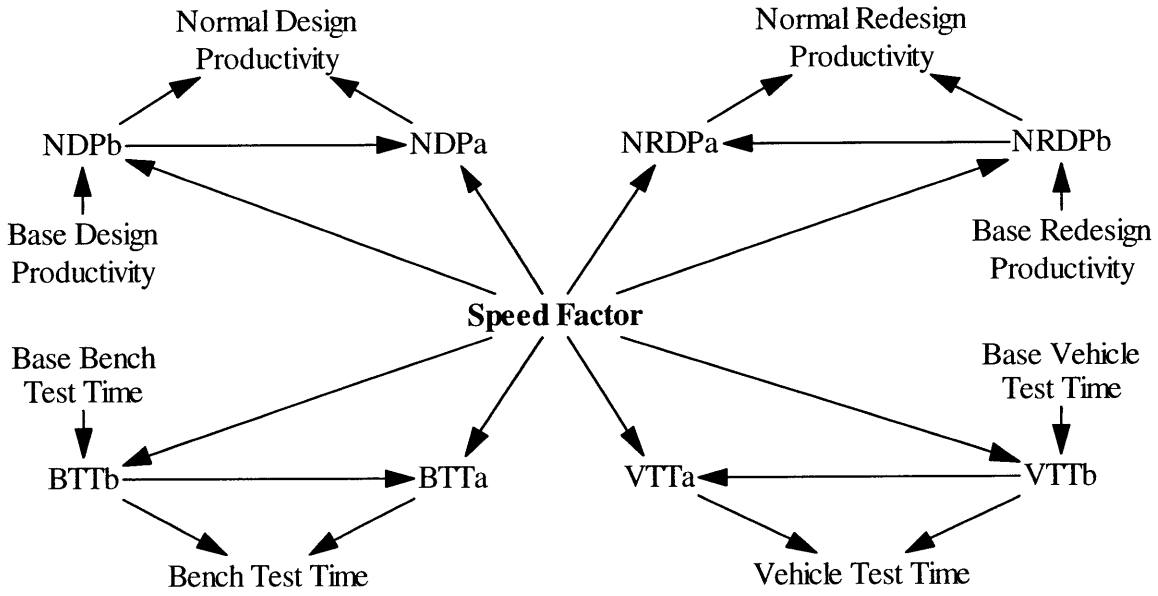


Figure 58 Speed Factor

Generally speaking, when there is a difference in speed between the two components it is assumed that component A is faster than component B. However, as formulated the NPD model can accommodate either component being faster than the other. The model variable *Speed Factor* sets the relative speed imbalance between the components. For example, the model's base speed factor of 2 indicates that component A is able to operate twice as fast as component B. This means that component A's normal productivity rates are twice as fast as those for component B and its test times are only half as long. Setting the *Normal Redesign Productivity* for the two components is completely analogous to the design productivity just discussed. However, the *Base Redesign Productivity* is set lower than the *Base Design Productivity* to reflect the fact that it takes more time to conduct fault analysis to establish the root cause of the error(s) in a design prior to redesigning it. See the technical documentation in the appendices for a description of the equations for these rates.

The formulation required for setting the *Bench Test Time* for the two components is similar to the formulation used above for design and redesign. It uses a *Base Bench Test Time* set in the model to be 1 month. This means that on average it takes one month to prototype a design, outfit it with any special test gauges, run the bench test, receive and analyze the results. The formulation for setting the *Vehicle Test Time* for the two components is exactly the same as setting the times for bench testing. However, the *Base Vehicle Test Time* is set at 3 months, considerably longer than the *Base Bench Test Time*. This longer test time reflects the time it takes to accumulate miles on the test vehicle and the time required to outfit the vehicle with various test gauges in order to collect different data. See the technical documentation in the appendices for a description of the equations used to set these test times for the two components.

Chapter 5:
Analysis

> ***"A successful model is a model that tells you something that you didn't tell it to tell you."***
>
> -- Professor Sam Savage, Stanford University

MODEL ANALYSIS

Perhaps the greatest benefit of using simulation is the ability to run the simulation model many times using different parameters in a short period of time at little to no cost beyond the initial cost of developing the simulation. This allows us to explore the model's behavior under a variety of conditions. By doing so, we can come to understand how the structure of the model, indeed the structure of the system we are modeling, affects its behavior and how we'd expect it to behave under a given set of conditions. This understanding aids the modeler in developing a set of policy alternatives that can be tested using the simulation. Of course, this is all aimed at gaining an improved understanding of our true system and the ability to develop robust policies that will improve our system's performance.

In this chapter, we will look at the behavior of the NPD model. We will look at the sensitivity of the model's behavior to the parameter values that we have selected. We will explore the model's behavior to identify the key pieces of model structure that drives its behavior. Finally, we will develop and test a set of policies that address the questions surrounding the number and timing of vehicle builds during a new product development project.

Model Behavior

We will begin by looking at the typical behavior of the NPD Model under a base set of conditions. The base model consists of two components each consisting of 1,000 parts that need to be designed, built, tested, and redesigned as necessary. Paper designs can be rapid-prototyped and bench tested where component errors can be discovered. Designs are prototyped as parts during each build event and subsequently tested where both component and integration errors can be discovered. Integration errors that are discovered in vehicle testing require coordination to be conducted between engineers from both components before it can be fixed. Component errors do not require coordination in order to be fixed. Each component begins the simulation with 10 engineers assigned to complete the engineering design, redesign, and coordination work. The base model assumes that component A is twice as fast as component B in completing its work (design, redesign, bench testing, and vehicle testing). Each simulation is run for 50 months. However, product launch is scheduled for month 36 therefore project performance is based on various measures of effectiveness at month 36. The base model assumes that 3 vehicle builds will be executed on the normal product development timeline at months 13 (Prototype Build - Proto), 25 (Design Intent Build - DIB) and 31 (First Production Event – FPE) in the simulation. See the full equation listing in Appendix B to find the values for each of the model parameters.

We will first look at the behavior of the model under the base set of conditions. Figure 59 shows a graph of the stock of designs released for both component A and component B. As you can see in the graph, the stock of designs released increases over time as designs are completed. As expected, the rate at which this stock increases is greater for component A since it can be designed, redesigned, and tested twice as fast as component B. However, you should also note that the stock of designs released is not strictly increasing. In fact, after each build event (at times 13,

25 and 31 months) the stock of designs released actually decreases as errors are found through vehicle testing in parts that correspond to the designs in the stock of designs released. As these errors are discovered, the corresponding paper designs are moved to either the stock of designs in coordination or the stock of designs in rework. Overall, the stock of designs released increases over the course of the new development project and by the time the product is scheduled for launch (month 36) almost 1,900 of the 2,000 initial designs are found in the stock of designs released with component A contributing slightly more than component B to this total.

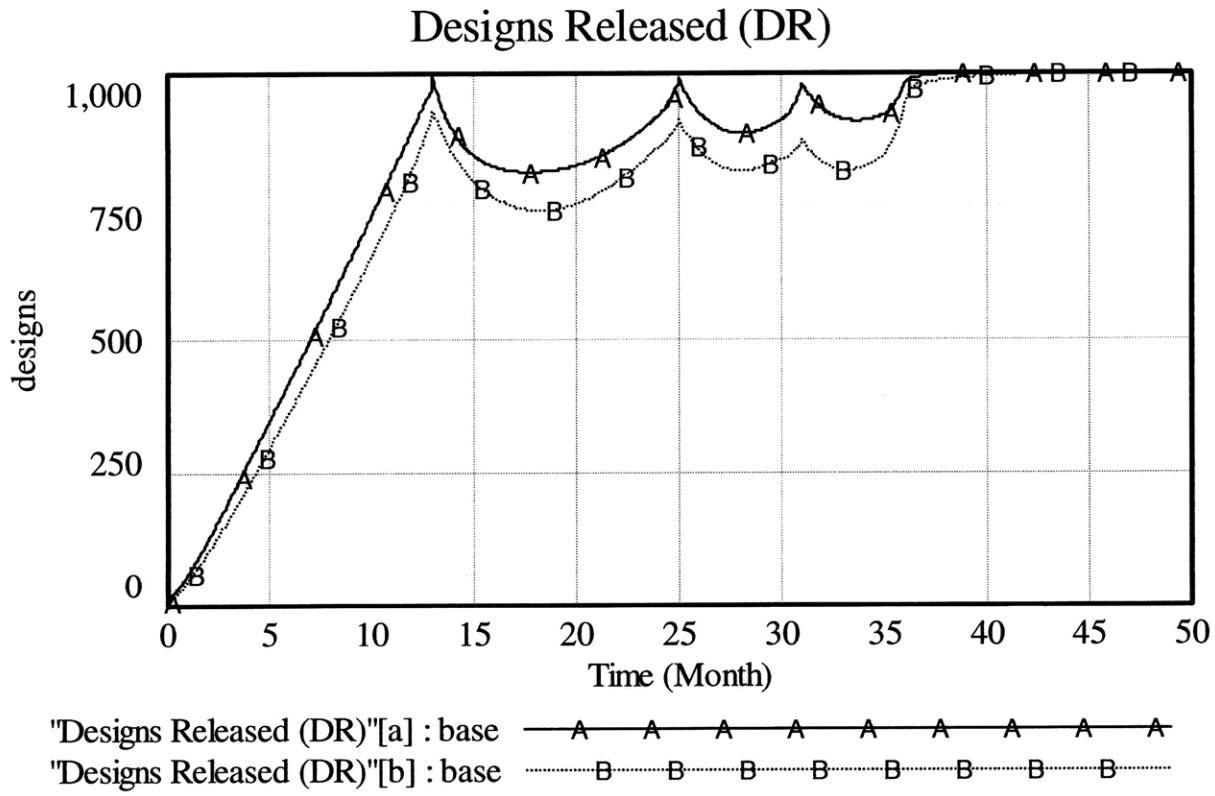


Figure 59 Base Model Designs Released

While the stock of designs released is an indication of the amount of work that has been accomplished it certainly does not tell the whole story. As noted, designs in this stock can, and

very often do, have errors associated with them. A better indication of the state of the project is the number of "clean" designs – those without component and/or integration errors and not coupled with a design with an integration error that might generate rework. A graph of the number of clean designs is included in Figure 60.

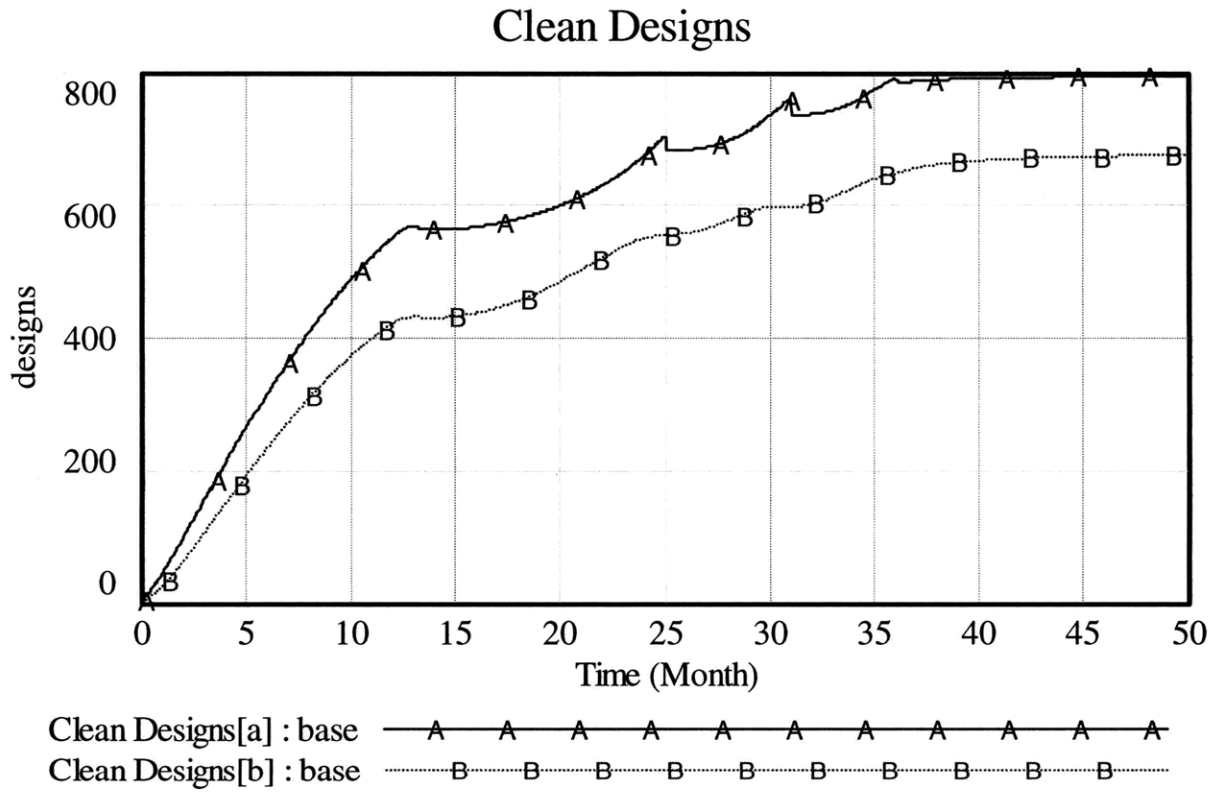


Figure 60 Base Model Clean Designs

As you can see from the graph above, the number of *Clean Designs* is significantly less than the number of *Designs Released* painting a different picture of project performance. In fact, after 36 months and 3 builds components A and B complete less than 80% and 70% of their required designs correctly. Again, you'll note that the number of clean designs goes down slightly after each vehicle build. This occurs as integration errors are discovered in vehicle testing which, when coordinated, require rework from coupled components. It is important to understand that the

number of clean designs accounts for errors in designs whether or not they have been discovered. This is a capability of the simulation model which is unrealistic in the real world. It is only through testing that one can hope to identify errors and "validate" those parts that do not incur any problems in testing.

Another measure of project performance is the total number of designs and redesigns that are executed during the course of the NPD project. This represents the amount of "design work" it costs the company to complete a project. A graph of the cumulative number of designs (both designs and redesigns) completed during the course of the NPD project is included below. As you can see from the graph, initially the cumulative number of designs is higher for component A. This is true because, in the base case, component A is able to complete designs, redesigns, and testing faster than component B. However, eventually component B executes more cumulative designs in an effort to complete its work overtaking component A in month 34.

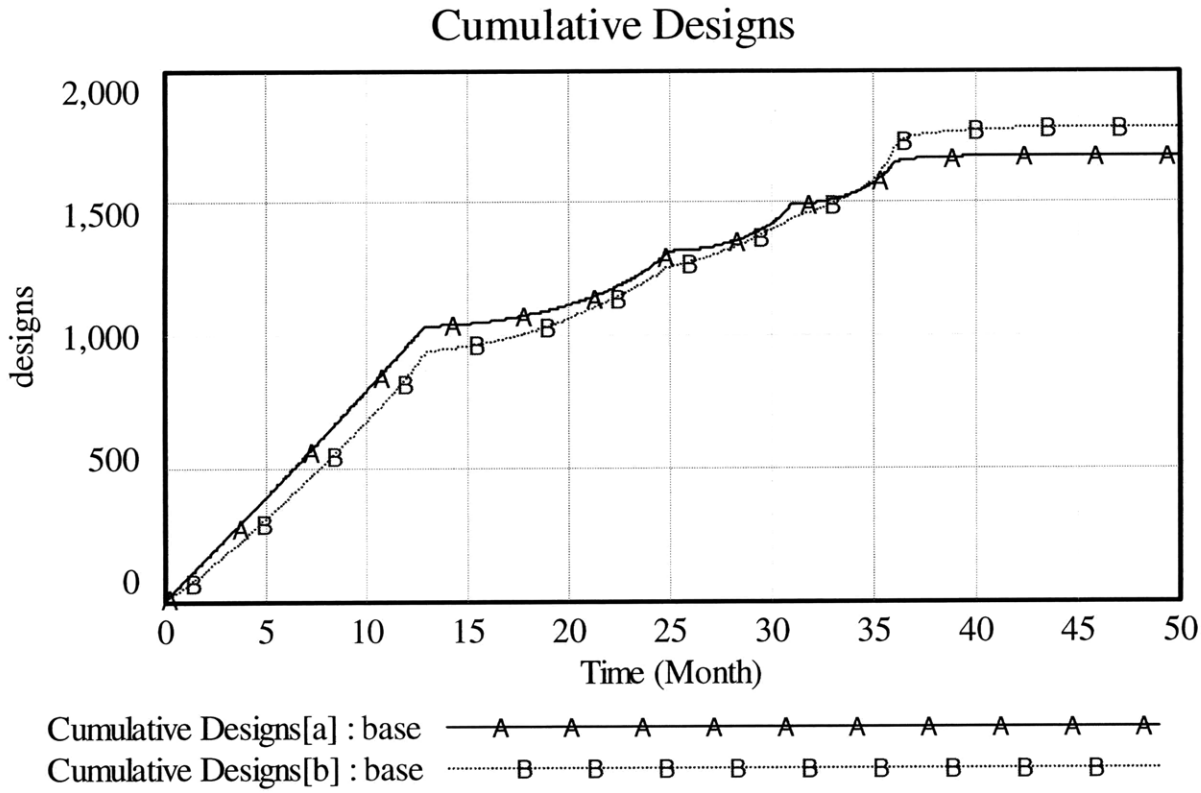


Figure 61. Cumulative Designs Completed

While the total number of cumulative designs executed in a project provides a sense of the cost incurred by the company, when combined with the number of clean designs we can get a sense of the cost versus value tradeoff the company makes in executing a project. We refer to this measure of performance as *clean design churn* which divides the total number of designs and redesigns completed by the number of clean designs. A graph of this measure over time is included in Figure 62 .

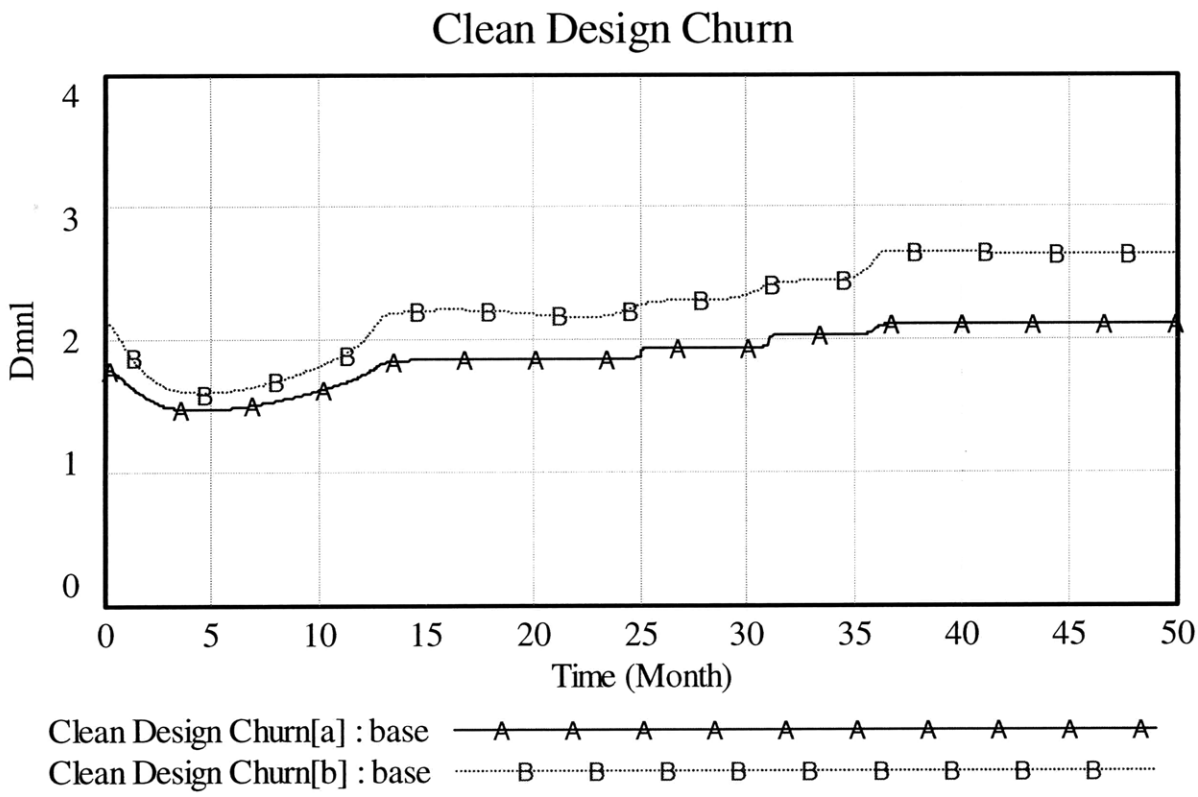


Figure 62 Base Model Clean Design Churn

By definition and construction, *Clean Design Churn* must be greater than or equal to 1 with a value of 1 being ideal. A value of 1 indicates that it only takes one design and no redesigns for each clean design to be completed. In other words, every design completed is done so without error. From Figure 62 we can see that component A has a lower clean design churn than component B in the base model and is thus more efficient. While component A is able to complete clean designs at a rate of just over 2 designs executed per clean design, component B takes more than 2.6 designs per clean design. Much of the difference between components in the clean design churn can be attributed to the lower quality of design and redesign work achieved by component B, the slower component, as its designers are forced to work faster than their normal work rate (thus reducing quality) in response to schedule pressure.

The limitation of using *Clean Design Churn* as a performance measure is that it doesn't account for the percentage of the original project work that gets accomplished during the course of the project. For example, a project may achieve the ideal value of 1 for clean design churn perhaps by working very slowly and carefully but in the end may only have completed half of the required design work. To account for this potentially misleading scenario, I created another measure of effectiveness which I call *MOE1*.

MOE1=(Clean Designs/Initial Designs)*zidz(1,Clean Design Churn) (1)

MOE1 is calculated in two parts. First the number of *Clean Designs* is divided by the number of *Initial Designs* – the number parts that must be designed for the project. This calculation gives the fraction of required designs that are completed correctly. Obvisuly, this fraction ranges between 0 and 1 with a higher number indicating better performance. This fraction is then multiplied by the inverse of the Clean Design Churn to calculate MOE1. Since larger values of Clean Design churn indicate a decrease in efficiency, in multiplying by the inverse of this number MOE1 penalizes a project for taking more design iterations in order to complete a design correctly. A graph of MOE1 in the base case is provided in Figure 63.

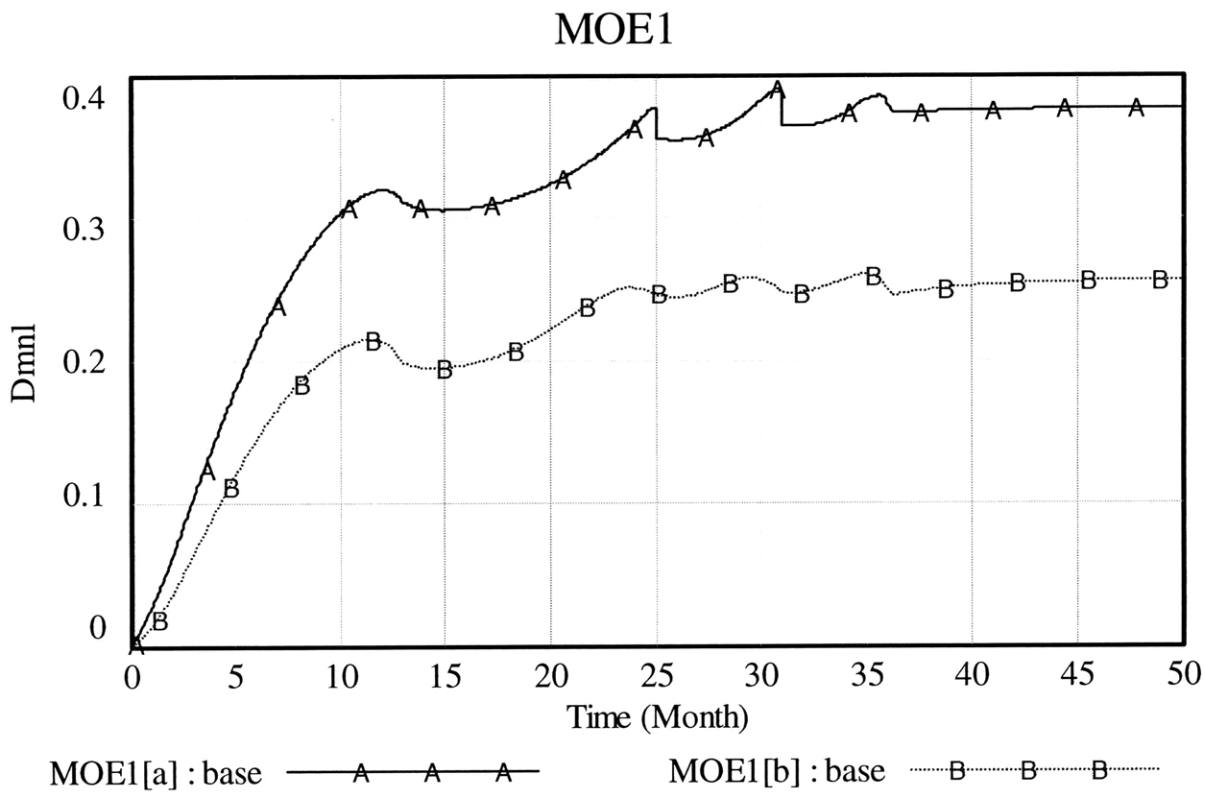


Figure 63. MOE1

As can be seen in Figure 63, component A performs much better than component B. Again, much of this difference in performance can be attributed to the lower quality of design and redesign work achieved by component B as it works at a faster than normal rate due to schedule pressure when compared with component A.

Decomposing the equation for MOE1, one can see that the range of values for MOE1 is between 0 and 1 with 1 being the ideal value. This ideal performance would be achieved when all of the initial designs are completed correctly (the fraction Clean Designs/Initial Designs in MOE1 equals 1) and each correct design only requires a single design iteration (Clean Design Churn and its inverse both equal 1). As can be seen in the graph of MOE1, while component A performs better than component B, both perform much lower than the ideal.

Sensitivity Analysis

Introduction

After understanding the basic behavior of the model, it is important to understand the sensitivity of the model's behavior to changes in its parameter values. The NPD model has a fairly long list of parameters which are fixed in the model. Table 1 includes a partial listing of the parameter values from the NPD model that were tested for sensitivity. In addition to the parameters analyzed here, I will discuss the sensitivity of such parameters as the number and timing of build events as part of the policy analysis section of this chapter.

The table provides the base value of each parameter as well as the minimum and maximum values assigned to the parameters for the purpose of conducting sensitivity analysis. The parameters are organized in the table by NPD model sector for ease of understanding. In general, to establish the high and the low values for the parameters, I divided the base value in half to establish the lower bound and I doubled the base value to establish the upper bound. These high and low parameter values used to check model sensitivity were adjusted as necessary based on the individual parameter and the appropriate region of interest.

The base values assigned to these parameters were set based on my discussions with key members of the Company including design engineers, test and evaluation engineers, as well as key leadership. While every effort was made to assign accurate values to each of the parameters, historical data was not available from the Company for every parameter and ultimately many of the parameter values had to be estimated based on interviews with subject matter experts. For this reason, it is especially important to conduct sensitivity analysis on these parameter values.

Through sensitivity analysis we can gain an understanding of the parameter values that have the most impact on the simulation model's behavior as well as identify those parameters that have

little to no impact on the model behavior. In this manner, we can determine those parameters for which we might want to collect more information or data in order to improve our estimate of the appropriate parameter values. Again, in each case the base parameter values used here were vetted with the Company leadership.

Sector	Variable	Parameter Value		
		Min	**Base**	Max
Design	Min Design Time	0.125	**0.25**	0.5
	Min Redesign Time	0.125	**0.25**	0.5
	Min Coordination Time	0.125	**0.25**	0.5
	BT Effectiveness	0.25	**0.5**	1
Quality	Base Design Component Quality	0.5	**0.85**	1
	Base Design Integration Quality	0.25	**0.5**	0.75
Labor	Time to Change TGT Designers	0.5	**2**	4
	AVG Time to Add Designers	0.25	**0.5**	1
	AVG Time to Mature	1	**2**	4
	AVG Time to Release Designers	0.125	**0.25**	0.5
	New Designer Productivity Fraction	0.25	**0.5**	0.75
	Init Designers	5	**10**	15
Speed Factor	Speed Factor	1	**2**	3
	Base Design Productivity	5	**10**	20
	Base Redesign Productivity	2.5	**5**	10
	Base Bench Test Time	0.5	**1**	2
	Base Vehicle Testing Time	1.5	**3**	6
Coordination Fraction	Intended Coordination Fraction	0.5	**1**	
Designer Allocation	Coordination Priority	0.25	**0.5**	1
Integration Error Acct	Fraction of Coupled Designs	0.5	**1**	**NA**
External Rework	External RW Probability	0.25	**0.5**	1
Bench Test Fraction	Intended Bench Test Fraction	0.25	**0.5**	1

Table 1 NPD Model Paramter Values

Sensitivity analysis was accomplished by varying the parameter values in the simulation model from their base case values. In this case, I individually varied the NPD model's parameters between the Hi and a Low values which are listed in the table for each parameter and ran new simulations of the base case model. Given these simulation runs, we can compare the behavior of the model in terms of its key performance measures under these varying conditions. For this purpose, we used the performance measure of the number of *Clean Designs* at launch as a measure of quality and both *Clean Design Churn* and *MOE1* as a measure of the cost (and value) incurred by the company to achieve that quality.

After identifying a few critical parameters to which the model appears to be sensitive, we further explore the values of these parameters and their corresponding impact on model behavior. In some cases, we look at the joint effect of these parameters on model behavior.

Sensitivity Results

Table 2 provides the initial results of the sensitivity analysis described above. This table provides the values for three key performance measures: Clean Designs, Clean Design Churn, and MOE1. These values are extracted from the simulation at the notional time of product launch. In the case of the NPD model, the product is presumed to be launched at month 36. The values provided in Table 2 represent raw values.

Performance Measure	Clean Designs		Clean Design Churn		MOE1	
Base Case	**1437**		**2.3**		**0.31**	
Variable	Min	Max	Min	Max	Min	Max
Min Design Time	1437	1437	2.31	2.31	0.31	0.31
Min Redesign Time	1439	1462	2.29	2.28	0.315	0.32
Min Coordination Time	1436	1444	2.32	2.3	0.31	0.314
BT Effectiveness	1431	1448	2.31	2.32	0.309	0.313
Base Design Component Quality	1040	1497	3.9	2.12	0.135	0.348
Base Design Integration Quality	1112	1767	3.3	1.67	0.17	0.53
Time to Change TGT Designers	1481	1430	2.27	2.3	0.327	0.31
AVG Time to Add Designers	1445	1430	2.31	2.31	0.313	0.309
AVG Time to Mature	1480	1357	2.22	2.51	0.334	0.27
AVG Time to Release Designers	1437	1437	2.31	2.31	0.31	0.31
New Designer Productivity Fraction	1415	1461	2.35	2.27	0.301	0.321
Init Designers	1166	1671	2.76	1.943	0.211	0.43
Speed Factor	1616	1310	2.02	2.56	0.399	0.255
Base Design Productivity	1212	1553	2.74	2.1	0.221	0.37
Base Redesign Productivity	1337	1637	2.45	2.03	0.273	0.403
Base Bench Test Time	1437	1437	2.32	2.31	0.31	0.311
Base Vehicle Testing Time	1474	1420	2.3	2.26	0.32	0.312
Intended Coordination Fraction	1429	**NA**	2.32	**NA**	0.308	**NA**
Coordination Priority	1398	1446	2.31	2.31	0.302	0.312
Fraction of Coupled Designs	1754	1344	1.68	2.54	0.523	0.265
External RW Probability	1571	1178	2.02	3.05	0.389	0.193
Intended Bench Test Fraction	1430	1451	2.32	2.31	0.309	0.314

Table 2 Sensitivity Analysis Results

The values for the three performance measures are provided initially for the **Base Case** scenario as well as when each of the model parameters is set at its minimum value and maximum value. Comparing these values with the base case allows us to identify the parameters that affect the behavior of the model the most and therefore should be estimated with care. For example, the minimum design time (*Min Design Time*) appears to have little to no effect on the project outcome as the key performance measures do not appear to fluctuate very much as the minimum design time varies between .0125 months (one half week) and 0.5 months (two weeks). We can conclude that the model's behavior is not sensitive to value of this parameter and therefore we don't need to worry about estimating its value with any greater fidelity.

Comparing the values in Table 2 to the base values for each of the performance measures isn't particularly easy. Table 3 is included below to facilitate the comparison of performance measures at the extreme values for each of the parameters insofar as it provides the percentage change from the base case that each of the performance measures achieves at the high and low values for their respective parameters. Using this table, it is much easier to identify the parameters that seem to have the largest impact on the model behavior. For example, it is easy to see by virtue of the large percentage changes listed for the *Base Design Component Quality* and the *Base Design Integration Quality* that the models behavior is sensitive to the values for these parameters and should be estimated with care.

Performance Measure	Clean Designs		Clean Design Churn		MOE1	
	(% Change)		(% Change)		(% Change)	
Variable	Min	Max	Min	Max	Min	Max
Min Design Time	0.0%	0.0%	0.4%	0.4%	0.0%	0.0%
Min Redesign Time	0.1%	1.7%	-0.4%	-0.9%	1.6%	3.2%
Min Coordination Time	-0.1%	0.5%	0.9%	0.0%	0.0%	1.3%
BT Effectiveness	-0.4%	0.8%	0.4%	0.9%	-0.3%	1.0%
Base Design Component Quality	**-27.6%**	4.2%	**69.6%**	-7.8%	**-56.5%**	12.3%
Base Design Integration Quality	**-22.6%**	**23.0%**	**43.5%**	**-27.4%**	**-45.2%**	**71.0%**
Time to Change TGT Designers	3.1%	-0.5%	-1.3%	0.0%	5.5%	0.0%
AVG Time to Add Designers	0.5%	-0.5%	0.4%	0.4%	1.0%	-0.3%
AVG Time to Mature	3.0%	-5.6%	-3.5%	9.1%	7.7%	-12.9%
AVG Time to Release Designers	0.0%	0.0%	0.4%	0.4%	0.0%	0.0%
New Designer Productivity Fraction	-1.5%	1.7%	2.2%	-1.3%	-2.9%	3.5%
Init Designers	**-18.9%**	**16.3%**	**20.0%**	**-15.5%**	**-31.9%**	**38.7%**
Speed Factor	**12.4%**	**-8.8%**	**-12.2%**	**11.3%**	**28.7%**	**-17.7%**
Base Design Productivity	**-15.7%**	**8.1%**	**19.1%**	**-8.7%**	**-28.7%**	**19.4%**
Base Redesign Productivity	**-7.0%**	**13.9%**	**6.5%**	**-11.7%**	**-11.9%**	**30.0%**
Base Bench Test Time	0.0%	0.0%	0.9%	0.4%	0.0%	0.3%
Base Vehicle Testing Time	2.6%	-1.2%	0.0%	-1.7%	3.2%	0.6%
Intended Coordination Fraction	-0.6%	NA	0.9%	NA	-0.6%	NA
Coordination Priority	-2.7%	0.6%	0.4%	0.4%	-2.6%	0.6%
Fraction of Coupled Designs	**22.0%**	**-6.5%**	**-27.0%**	**10.4%**	**68.7%**	**-14.5%**
External RW Probability	**9.3%**	**-18.0%**	**-12.2%**	**32.6%**	**25.5%**	**-37.7%**
Intended Bench Test Fraction	-0.5%	1.0%	0.9%	0.4%	-0.3%	1.3%

Table 3 Sensitivity Analysis (Percent Change)

You should note that the values in this table are normalized in that they represent a percentage change (+ or -) in the base case performance. Again, this makes it easier to identify the parameters that drive the behavior of the model.

Lastly, the table below provides information regarding the range (maximum minus minimum value) of model behavior in response to changes in the parameter values. In this case, I highlighted those ranges that were particularly large. Again, this information helps us to decide which of our model parameters are sensitive and causing this abnormal behavior. In this case, as expected from the previous table, the range of values for the performance measures is quite large for base design component and integration quality. Additionally, it looks like the model's behavior is sensitive to the number of initial designers assigned to each component's design team as well as the *Speed Factor* and the Design and Redesign Productivity. This sensitivity should be expected. Perhaps more importantly, the model's behavior (especially MOE1) is sensitive to the *Fraction of Coupled Designs* and the *External Rework Probability* both of which are reflective of the degree to which the components are coupled or dependent on one another in their design work.

Performance Measure	Clean Designs	Clean Design Churn	MOE1
Variable	Range (%)	Range (%)	Range (%)
Min Design Time	0.0%	0.0%	0.0%
Min Redesign Time	1.6%	0.4%	1.6%
Min Coordination Time	0.6%	0.9%	1.3%
BT Effectiveness	1.2%	0.4%	1.3%
Base Design Component Quality	**31.8%**	***77.4%***	**68.7%**
Base Design Integration Quality	**45.6%**	**70.9%**	***116.1%***
Time to Change TGT Designers	3.5%	1.3%	5.5%
AVG Time to Add Designers	1.0%	0.0%	1.3%
AVG Time to Mature	8.6%	12.6%	20.6%
AVG Time to Release Designers	0.0%	0.0%	0.0%
New Designer Productivity Fraction	3.2%	3.5%	6.5%
Init Designers	**35.1%**	**35.5%**	**70.6%**
Speed Factor	21.3%	23.5%	**46.5%**
Base Design Productivity	23.7%	**27.8%**	**48.1%**
Base Redesign Productivity	20.9%	18.3%	**41.9%**
Base Bench Test Time	0.0%	0.4%	0.3%
Base Vehicle Testing Time	3.8%	1.7%	2.6%
Intended Coordination Fraction	NA	NA	NA
Coordination Priority	3.3%	0.0%	3.2%
Fraction of Coupled Designs	**28.5%**	**37.4%**	***83.2%***
External RW Probability	**27.3%**	**44.8%**	**63.2%**
Intended Bench Test Fraction	1.5%	0.4%	1.6%

Table 4 Sensitivity Analysis (% Ranges)

Based on the sensitivity data presented above, it is apparent that the behavior of the base NPD model is most sensitive to changes in the values of following model parameters:

- Base Design Component Quality
- Base Design Integration Quality
- Base Design Productivity
- Base Redesign Productivity
- Speed Factor
- Fraction of Coupled Design
- External Rework Probability

As such, it is important to further examine the impact of changes in the values of these parameters on the behavior of the NPD model, especially the performance measures of interest. In particular, we can look at the effect of changes in the values of these parameters on the project performance measure MOE1. In doing so, we can look at a given parameter's value in finer detail across a range of possible values. We do this below for both *Base Design Component Quality* and *Base Design Integration Quality.* In both cases we can see that project performance (MOE1) increases as the component and the integration quality increase. Figure 64 demonstrates the increase in performance as the Base *Design Component Quality* varies between 0 and 1. Figure 65 demonstrates similar behavior for an increase in the *Base Design Integration Quality.*

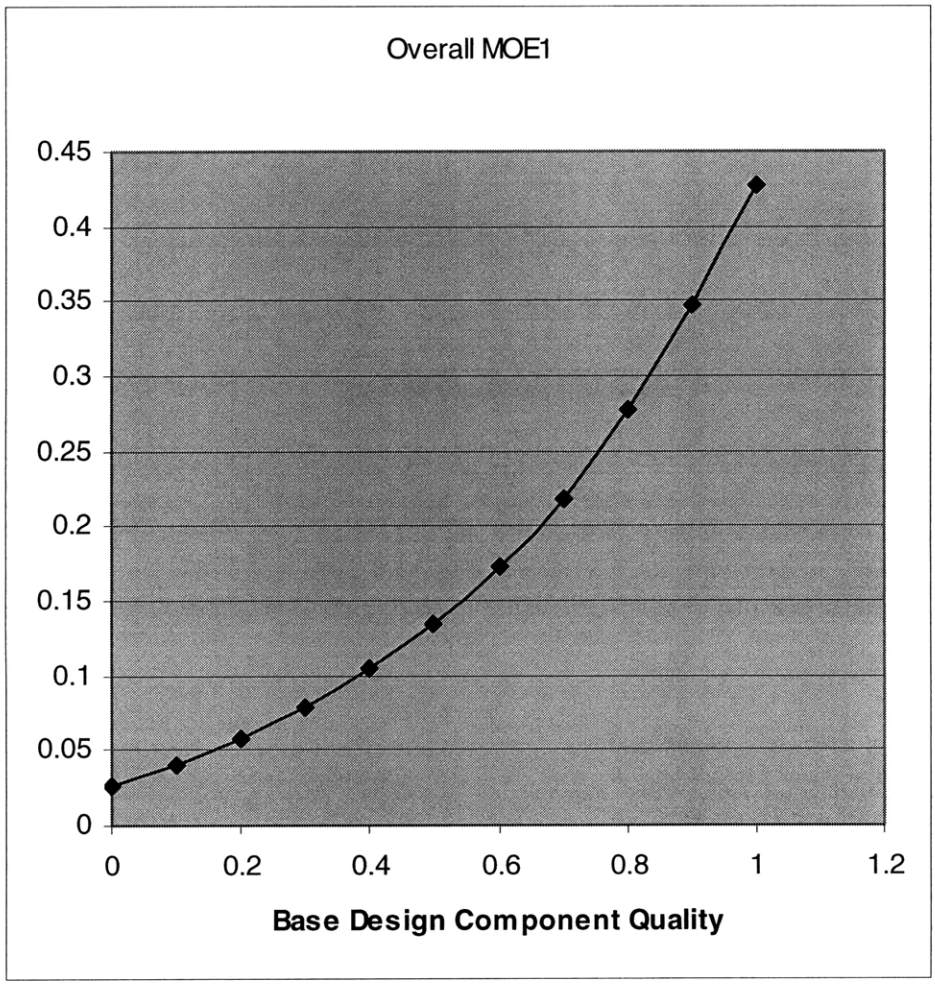


Figure 64 MOE1 vs Component Quality

We would expect this to happen as a higher base level of the quality of design should result in fewer mistakes made in completing designs. In turn the higher base level of quality also impacts the quality of redesign work. It does this both directly because the equations for the base redesign component quality and redesign integration quality are a function of the base quality of design. However, perhaps more importantly the higher quality of design work reduces the number of errors created in initial designs and thus decreases the amount of rework that is generated. With all else being equal, this will reduce the schedule pressure felt by individual designers and allow them to work closer to their normal productivity rate and thus minimize any reduction in the quality of their

work attributable to cutting corners. This reduces the number of errors generated in both initial design work and redesign work.

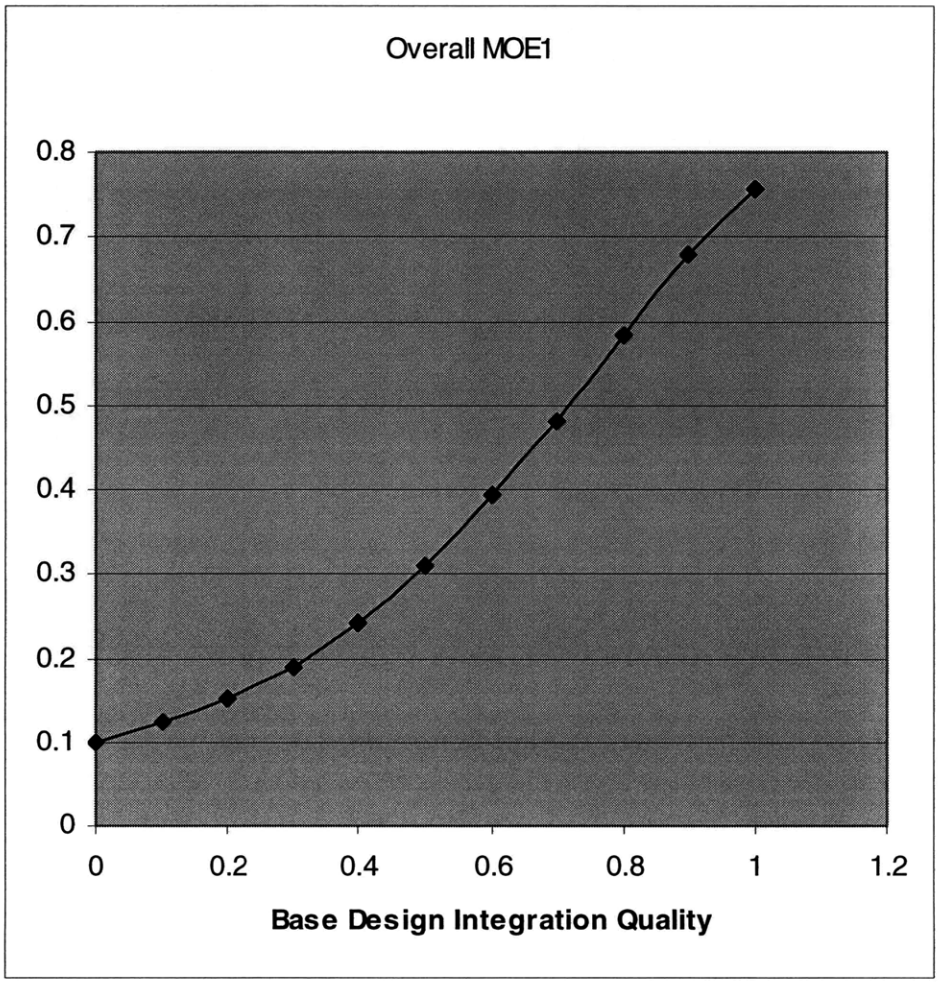


Figure 65 MOE1 vs Integration Quality

As you compare the graphs of MOE1 in Figure 64 and Figure 65 versus the two different base design qualities (component and integration), it is interesting to note that even when component quality is perfect (i.e. it equals 1) the best that the project can achieve is an MOE1 score of approximately 0.43 where a value of 1 is ideal. Conversely, when integration quality is perfect the project is able to achieve an MOE1 score of better than 0.75. These contrasting scores show how

strongly project performance is impacted by integration quality and subsequently integration errors.

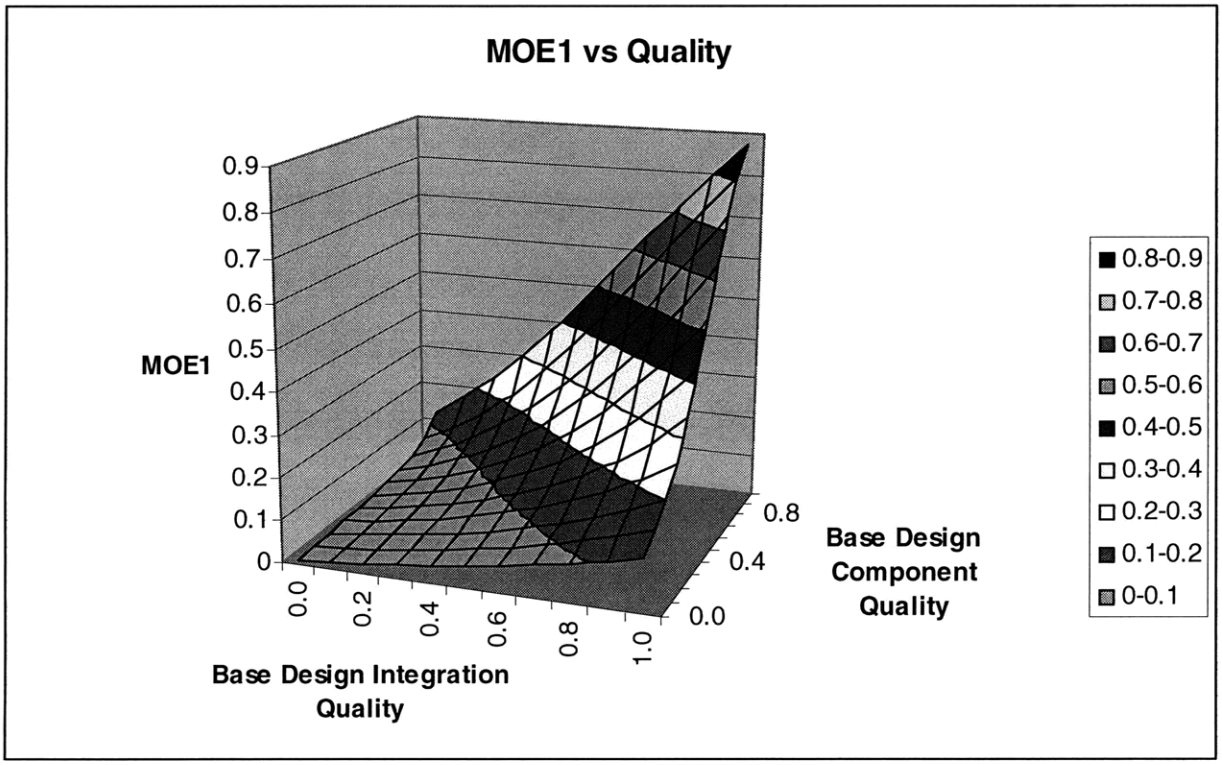


Figure 66 MOE1 vs Quality

It is clear from Figure 66 that project performance is traded off with both component and integration quality. That is not surprising. This same degradation in performance can be seen in graphs of the total number of clean designs completed by the scheduled launch date and the clean design churn necessary to achieve that number. In both cases, as the quality goes down so too does overall project performance. In addition to passing the common sense test, this relationship between design quality and overall project performance is in keeping with past project performance at the Company I studied. The values that I selected for the base design Component and Integration Quality were based on discussions I had with design engineers, testing officers, and project managers in the Company. All agreed that the values I selected were reasonable and

expressed the difficulty with selecting the "right" value because the quality of design could (and would) vary based on the experience of individual designers, schedule pressure, and the complexity of the part being designed, among other factors. While the NPD model developed here endogenously considers the impact of experience and schedule pressure on design quality, it does not directly account for the impact of part complexity on the probability of generating a component or integration error in design work.

Similar to the effect of an increase in design quality on project performance, we'd also expect an increase in the *Base Design Productivity* to increase project performance – and for the most part that expectation is fulfilled. However, there are certainly limitations and the amount of improvement varies greatly with the parameter value. The graph below shows the project performance (MOE1) under different conditions of design productivity.

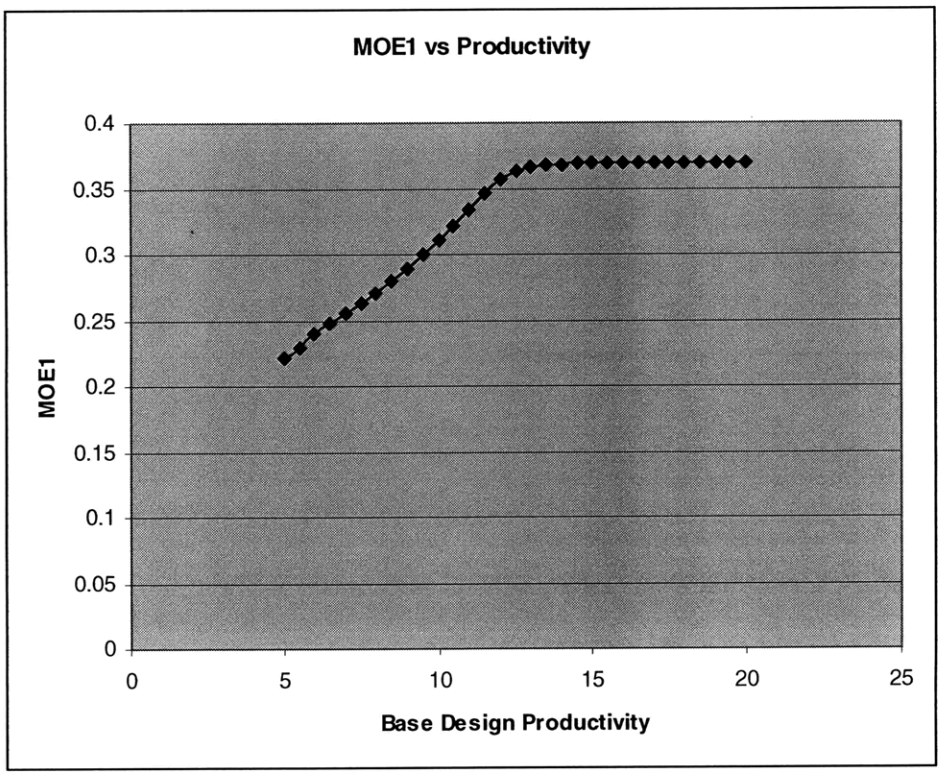


Figure 67 Productivity and Project Performance

In this case, as the *Base Design Productivity* increases the overall project performance increases in terms of MOE1. The number of clean designs also increases as can be seen in the graph below. However, in both cases the improvement in project performance is limited and levels off even as productivity continues to increase.

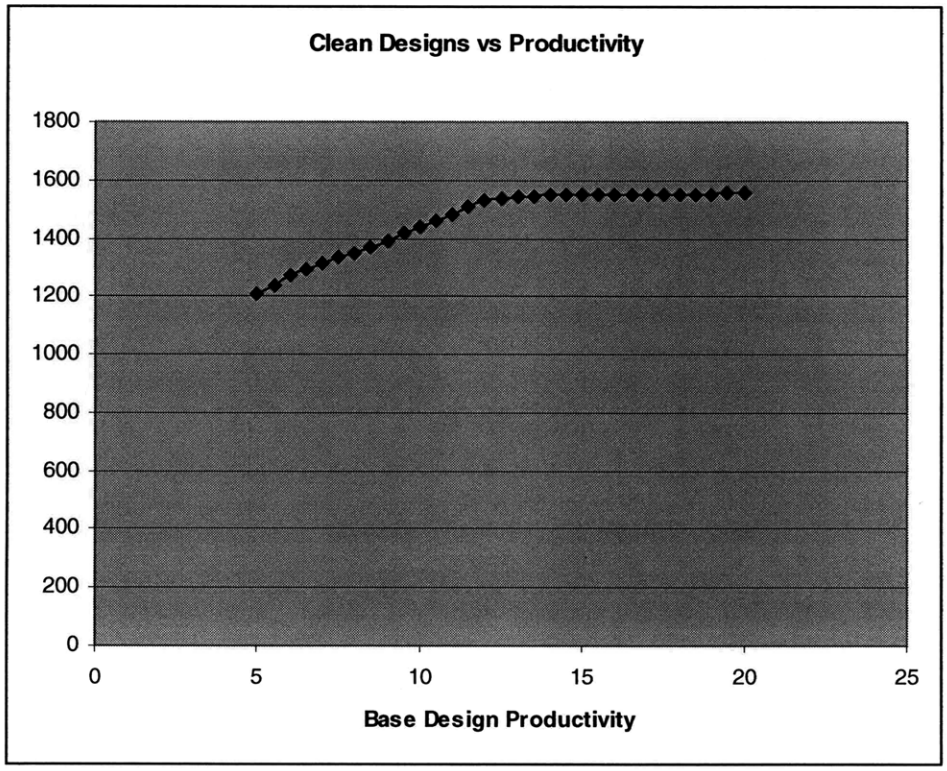


Figure 68 Productivity and Clean Designs

One of the key features of the NPD model is its ability to account for the complexity of a new product development project especially as that complexity is experienced at the client Company. The NPD model accounts for project complexity in three ways. First, it is able to vary the degree of interaction between components. It does this through the use of the parameter *fraction of coupled parts* which determines the fraction of parts for a given component that interacts with parts from another component. The second way that the model accounts for complexity is with the *probability of external rework.* As explained in chapter 4, this parameter determines the probability that a given part will require redesign upon coordinating an integration error with a coupled design/part from a coupled component. The last piece of complexity the NPD model accounts for directly is the *speed factor* which accounts for the imbalance between components in

the time it takes to design, redesign, or test the parts of a given component. The preliminary sensitivity analysis results above indicate that the model's behavior is sensitive to the values of these three parameters. As mentioned above, the effect of these parameters on project performance will be examined in detail in the Policy Analysis section later in this chapter.

Model Verification

After understanding the basic behavior of the model, it is important to ensure that the stylized model that I have developed accurately reflects the new product development process in place at the Company that I researched. While no model can replicate the product development process exactly (indeed, by definition a model should not) it is important that the model capture the key features of the process and replicate, at least qualitatively, the behavior of the actual system or process. In this case, I have developed my model based on extensive interviews of engineers and managers within the Company's new product development organization and vetting the model with them at various points in time.

After completing the development of the model, I "opened the black box" of the model to key members of the Company's new product development organization. In addition to having extensive experience as design engineers and project leaders of new product development projects, these key organizational members were also intimately familiar with the formal new product development process in place at the company as they each had spent time working in the product development office (PDO) which was responsible for developing the formal design process and providing oversight to new product development projects. Additionally, these members were all familiar with system dynamics and system dynamics simulation models. Their experience made them perfectly suited to review the model I developed. These key members of the Company's new product development organization uniformly agreed that the NPD model developed here reflects

the key features of the new product process in place at the Company. They also agreed that the behavior generated by the model reflects the behavior they had experienced within the new product development organization in past and current new product development projects.

In addition to selecting parameter values based on my interviews, I also reviewed the results, both qualitative and quantitative, of past NPD projects in the Company. In doing so I was able to gain an appreciation for the kind of behavior that characterizes many of the new product development projects. Indeed, in many of my conversations with engineers at the Company I heard a familiar story of NPD projects that were falling further and further behind each year. Problems were being discovered and subsequently resolved later and later in the development process. There was one project in particular, which I'll call Project ATOMac that many of the design engineers mentioned as a kind of "poster child" for the development problems that were becoming more and more ubiquitous in their organization. I decided to use this particular project as the basis for the NPD model I developed here while still keeping the model generic enough to capture the formal and informal processes that existed across projects and in the New Product Development organization in the Company.

Project ATOMac

I based my NPD simulation on Project ATOMac, a large scale project referred to as a Bin 6 project in the Company to reflect both the size (the sheer number of new parts that needed to be developed) and complexity of the project as well as the resources it was expected to consume. As mentioned earlier, a Bin 6 project was the "largest" project the company would take on and normally the company would never have more than one Bin 6 project due for completion in any given model year. In this particular project, two major subsystems were scheduled to undergo major changes. Normally, it would only take one of these two particular subsystems to undergo a

major overhaul for a project to qualify as a Bin 6 project. Additionally, the two subsystems in Project ATOMac were tightly coupled meaning that they had many parts that interfaced with the other subsystem and the design engineers from both of these components would need to work closely together in order to avoid the type of integration problems that were becoming more and more common in projects at the Company. Project ATOMac was slated to follow the formal development process in place at the Company and to follow the normal development timeline of 3 years.

From the very beginning ATOMac began to experience problems. Both of the major subsystems undergoing a significant redesign encountered a number of technical development challenges and problems that put the project behind. At various points along the way, the two major subsystems made major design changes to their systems that, in turn, caused the other subsystem to have to make significant changes. As mentioned, the two systems were tightly coupled with one another. With poor coordination and poor integration as the norm in Project ATOMac, each set of changes from one subsystem would render some part of the coupled subsystem obsolete and required major rework.

At one point, the ATOMac project's planned launch date was slipped an entire year in the Company's product plan timeline because of the difficulties the project was experiencing and the uncertainty as to whether or not the project team would be able to make the planned technology work. Slipping a project an entire year, especially one of this size, was not taken lightly at the company especially given their strong desire to get their products into the market. Given the size of this project and the degree of overhaul planned, Project ATOMac represented a major portion of the Company's new product plan.

As a result of the slipped timeline, there was a renewed level of interest (and the requisite additional oversight) in the project and its progress. Given the significant technical problems that the project was experiencing and increased pressure to get the project back on track, the project team executed a series of prototype builds. Whereas the normal NPD project will typically execute a single prototype build prior to the Design Intent build (DIB), this project executed 5 prototype builds before executing the regularly scheduled DIB and First Production Event build (FPE). To add insult to injury, the project was forced to execute a Second Production Event (SPE) build because there were too many problems with the FPE build that would prevent the launch of the new product. You'll recall that the FPE build event is held about 3 months before launch and is intended to demonstrate the product design and the production process have matured such that the vehicle can be produced using production tooled parts, equipment and processes. In this case, it was necessary for the project to execute a second production event just 6 weeks before the scheduled launch.

Engineers, who had worked on this project, from both of the major subsystems, spoke of a project that had gone wrong from the beginning. They described how the two subsystems and their respective design engineers were out of synch right from the beginning. They found themselves showing up at the many build events in various states of readiness. They described a situation where one system's learnings from a build event were leapfrogging the other system's learning cycle. In other words, one component would be ready for a given build event having had time to complete testing, conduct the appropriate fault analysis, explore possible design changes, and then settle in on a design and have it prototyped in time for the build. Meanwhile, the other system group would always seem to be one cycle out of phase. Perhaps having just recently completed testing and not yet complete with their analysis and/or redesign, facing a longer lead time for

prototyping, the "other" subsystem team would come to the build event with a part that they knew was incomplete and not representative of their design intent.

In replicating Project ATOMac using the NPD model, I made two changes to the base NPD model. First, I scheduled 4 prototype builds at the same times these builds occurred during the three year development cycle in the original project. The fifth prototype build that I mentioned the project team had executed, actually occurred prior to the 3-year development window (presumably before the project was slipped a year) and I do not have data on that build. A table of the scheduled build dates that match the ATOMac project is included below.

Month	Build
4	Proto1
12	Proto2
17.5	Proto3
22.25	Proto4
28.75	DIB
34.5	FPE
37.75	SPE
39.5	Launch

Figure 69 Project ATOMac Build Schedule

The second change that I made was to reset the schedule pressure experienced in the NPD model. In the original NPD model, designers experience schedule pressure as they trade off the work they have remaining with the time until the next build event. These build events include prototype builds, any DIB builds and/or any FPE (including SPE) builds. As mentioned in the description of the model, designers plan their work based on these build dates and there is significant pressure on individual designers to make sure that they have their part(s) ready for the

build events. In the main hallway of the building that housed the engineering organization where I conducted my research, the Company had electronic countdown clocks prominently displayed that count down the time (months, days, hours, and minutes) until the various scheduled build events including the actual launch of the product. The NPD model is constructed to represent schedule pressure from this perspective.

However, based on my interviews with engineers from Project ATOMac and others, prototype builds are not taken as seriously as the DIB and FPE builds. To begin with, DIB and FPE builds are completed on the actual production and assembly line. Prototype builds are not. With the normal production shutting down for the duration of these DIB and FPE builds, they quickly become the main company effort and the leadership takes a keen interest in their outcome. Often times, the leadership will even "test drive" one of the products right off of the line. By all accounts, that same level of interest just doesn't exist for the prototype builds. And while the DIB and FPE builds are appropriately scheduled at the company level given their impact on normal production, prototype builds are scheduled by individual project leaders and consequently they often get delayed, rescheduled, and/or cancelled. Additionally, many engineers indicated that when too many prototype builds are scheduled for a given project they lose even more significance and the engineers are often left wondering if this is the "real prototype" build. Lastly, the prototype vehicles often get pushed to the back of the queue when it comes to competing for scarce vehicle testing resources. As a result, most design engineers I spoke with indicated that while they understood the value of executing prototype builds they never really perceived them to be very important and rarely gave them much priority.

In order to capture this behavior in the NPD model, I set the perceived schedule pressure based on the tradeoff between work to do and the time remaining until the next "real" build – those builds

that occur at DIB or beyond. The model is able to accommodate pre-DIB builds as "real" builds that engineers prepare for like any other build. This will be discussed further in the policy analysis section of this chapter.

Having made minor adjustments to the basic NPD model to accommodate the idiosyncrasies of the ATOMAC project, I had to gain access to data that I could use to check the calibration of my model. In doing so, not only was I looking for data that would be readily available but I wanted to use a measure of project performance that would be meaningful to design engineers, project leaders, and managers alike. Based on the data that was made available to me from the Company, I settled on the cumulative number of Test Incident Reports (TIRs) which reflect the errors that are discovered in both bench testing and vehicle testing. The number of open (unresolved) TIR's is a common metric used to measure the risk of a NPD project in the Company. Additionally, the discovery of more and more errors at vehicle builds rather than in bench testing is becoming problematic in the Company especially as these errors are being discovered later and later in the development process. Late discovery of design errors are problematic for a number of reasons not the least of which is the significant cost associated with retooling manufacturing machines and processes. Perhaps most costly (and feared) is the design flaw that is not discovered until after the product has been launched and is so significant that it requires a recall. It was this fear that initiated the Company effort that led to my research. The following is a picture of the trend of issues being discovered and remaining unresolved later in the development and closer to launch. This picture was developed in conjunction with the Company's product development leadership to represent their perceptions and fears of NPD project management.

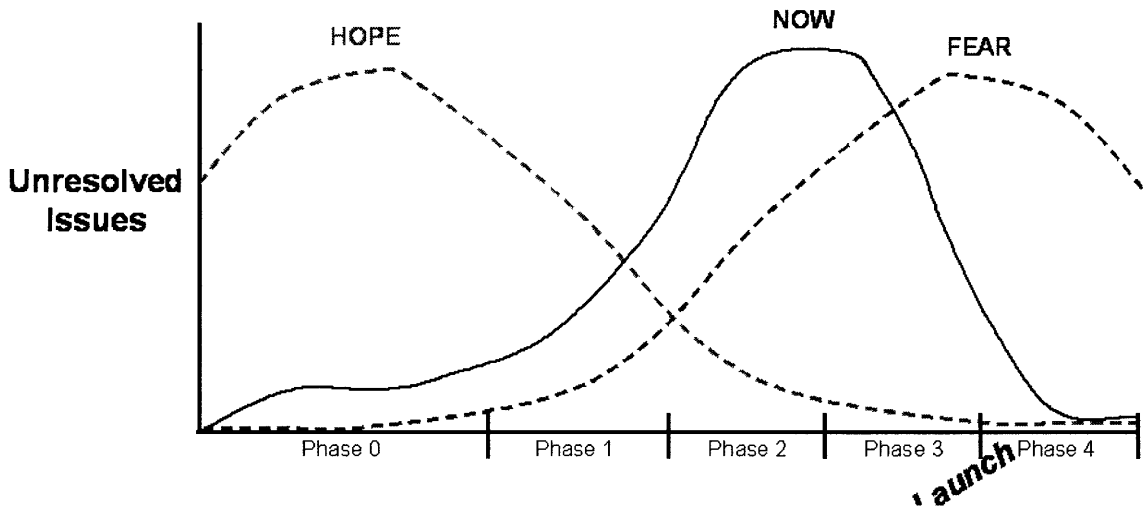


Figure 70 Managerial Perceptions and Fears

Besides the two adjustments described above, no adjustments were made to the NPD model in order to compare the output of the model to the Test Incident Report (TIR) data I had obtained for Project ATOMac. As discussed in the model documentation, the NPD model is set up with two main components which represent the two main subsystems involved in the ATOMac project. Each component has 1,000 parts that need to be redesigned. Each component initially has 10 designers assigned to work on the parts associated with his or her component. Additional design engineers can be added to the project team as necessary throughout the duration of the NPD project as outlined in Labor sector of the model. The simulation is run for 50 months with the realization that product launch for Project ATOMac occurred at month 39.25 historically. The project data that I have shows the cumulative number of TIR's that accumulated during the 39 months of the project that preceded the product launch. This data is compared with the simulation output of the NPD model in Figure 71.

In comparing the simulated data (***ATOMac Simulation***) with the historical data (***ATOMac Data***) the simulated data tracks the historical fairly closely. Some observations: First, the NPD

simulation model appears to discover errors earlier in the development cycle than occurred historically. A behavioral explanation for this difference is that many times the vehicles built during the early Proto builds get superseded in vehicle testing priority by vehicles that are later in their development cycle. A modeling explanation for this difference in behavior is that the NPD model assumes a first order delay in vehicle testing and the discovery of errors which begins in conjunction with each of the build events. The first order delay and the resulting behavior are consistent with the discovery of errors in real world vehicle testing. A more complicated formulation was considered that included the accumulation of test miles and the corresponding discovery of errors. In this formulation one would be able to address the shifting priority of vehicle testing from early prototype build vehicles to later build vehicles such as those from DIB and FPE. However, based on testing the formulation I determined that the additional model complexity would not significantly improve the performance of the model and would unnecessarily complicate the formulation of vehicle testing and the accounting of errors in multiple builds.

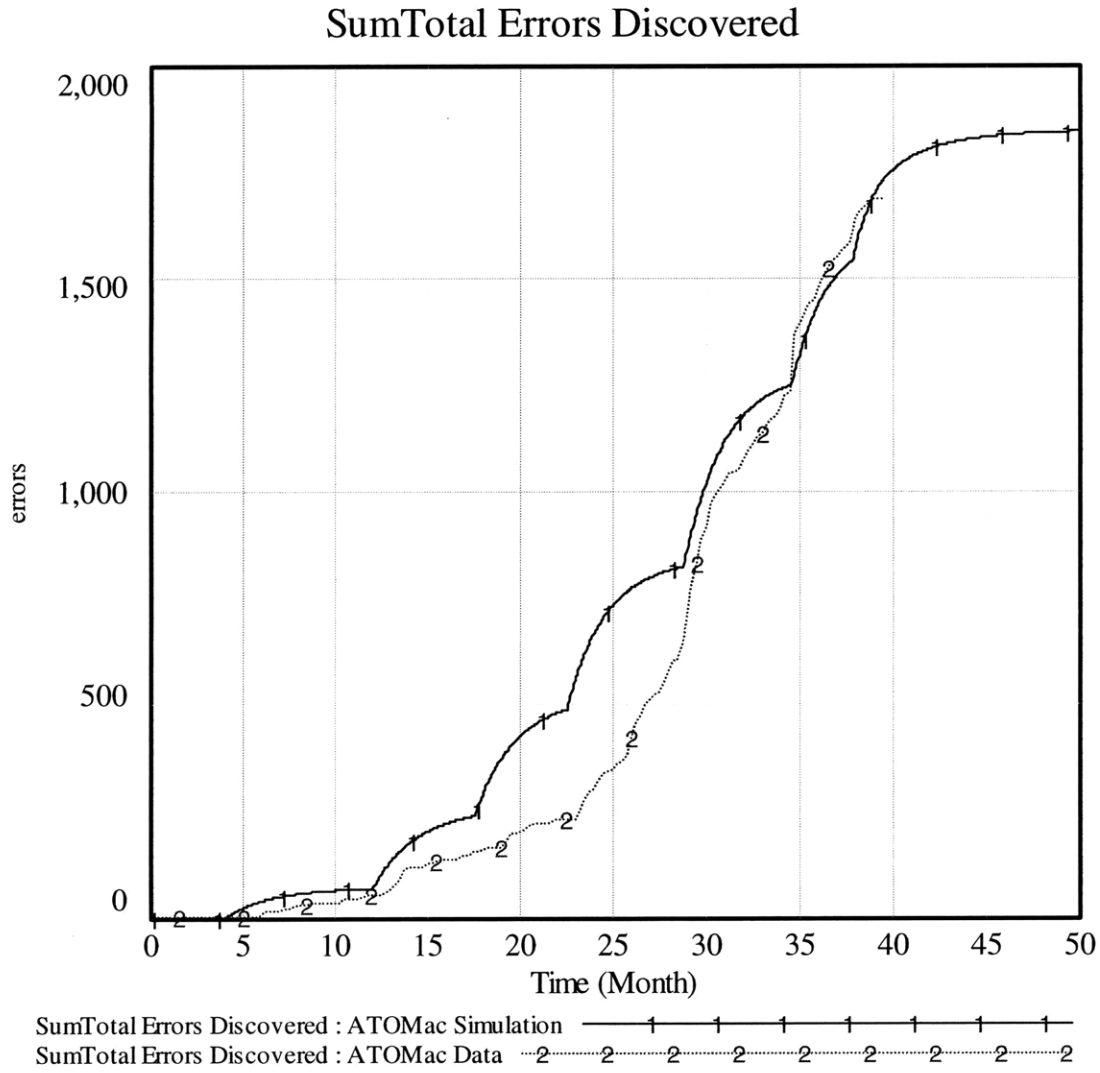


Figure 71 Model Calibration - Cumulative Errors Discovered

While Figure 71 suggests that the NPD model developed here roughly approximates the historical behavior of Project ATOMac, it is important to remember that the NPD model has a number of simplifying assumptions in its formulation which will be discussed in the next chapter. In the end, the purpose of the NPD model is not to perfectly model and replicate a particular new product development project but rather to capture the important decision processes and structural feedbacks that govern the behavior of new product development and in particular the concurrent engineering. Doing so allows the modeler to examine how the structure of these processes and

feedbacks affects project performance and therefore test a variety of possible policy initiatives aimed at improving project performance.

POLICY ANALYSIS

> ***"One thing's for sure: If we keep doing what we're doing, we're going to keep getting what we're getting."***[11]
>
> -- Stephen R. Covey, Organizational Leadership Consultant and Author of the International Bestseller, *The 7 Habits of Highly Effective People*

Research Questions

Having confidence that the system dynamics model that I have developed accurately reflects the given Company's new product development process but more generally that it reflects a reasonable new product development process, it allows me to use the NPD Model to consider various policies that the Company currently employs in its new product development process as well as to examine potential policies that they might consider. The primary question that is of concern in the Company is as follows: ***How often should we execute a design-build-test cycle in order to minimize the new product development time and cost while attaining a desired level of quality?***

In support of the effort of answering this question, I have outlined a number of more specific research questions which I will attempt to answer in the remainder of this chapter. These research questions and the motivations for each are described here.

---

11 Covey, S. R., A. R. Merrill, et al. (1994). First Things First: To Live, to Love, to Learn, to Leave a Legacy. New York, Fireside. page 30

First, to answer the fundamental question above I will examine the relationship between the number of build events and the level of quality as measured using the number of unresolved issues (component and integration errors) attained by a fixed time (i.e. launch date). I will also look at the number of designs that it takes to attain that level of quality. In examining the cost of attaining a given level of quality we use the *Clean Design Churn* performance measure described earlier. Lastly, phantom work plays a key role in the computation of cost because it represents potentially unnecessary design work which contributes to the cumulative number of designs executed. More importantly, it ties up designers who might otherwise be doing required design or coordination work within the given project or on other projects.

How many prototype build events should be executed given a fixed deadline (i.e. product launch date) and disparate design iteration cycle times between coupled subsystems that are developed concurrently?

To answer this first question we will look at the performance of the base NPD model and include none, one, two, three and four prototype builds. In all simulations we assume that the DIB and FPE build events will be conducted according to the Company's planned development schedule. We also assume that the design engineers will treat each prototype build as "real builds" and work to complete their design work prior to each prototype build. In this analysis, the prototype builds are scheduled by spreading out the build events evenly in the months preceding the DIB build. The results of these simulations are shown below.

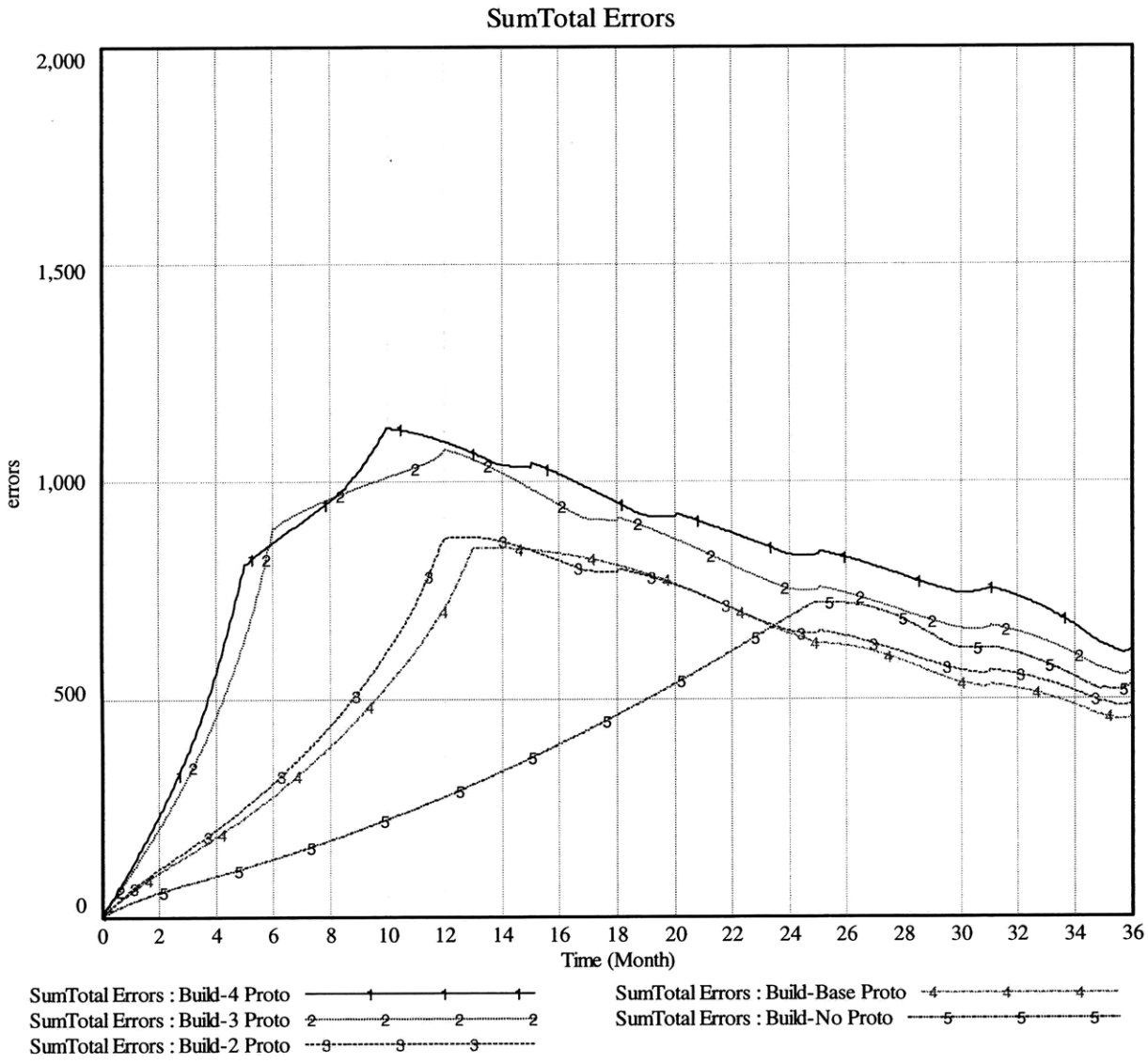


Figure 72 To Build or Not to Build?

Figure 72 shows the number of errors that remain at product launch given varying numbers of prototype build events. It is clear from the graph that the worst performance is realized when the NPD project tries to execute 4 prototype builds (in months 5, 20, 15 and 20) in addition to the DIB and FPE build events (4 Proto). Conversely, the best performance is attained when a single prototype build is executed at month 13. These results indicate that fewer build events may indeed be better than having too many.

By focusing on the last six months of the graph above, we get a good picture of the relative performance of the projects using different numbers of prototype builds. The general rule that can be seen here is that performance gets worse as the number of prototype builds increases. The notable exception is the case where no prototype build is executed. In this scenario, represented by No-Proto in Figure 73 below, not conducting a prototype build outperforms the 3 Proto and the 4 Proto build scenarios but does not perform as well as the 2 Proto and Base (1) Proto build scenarios.

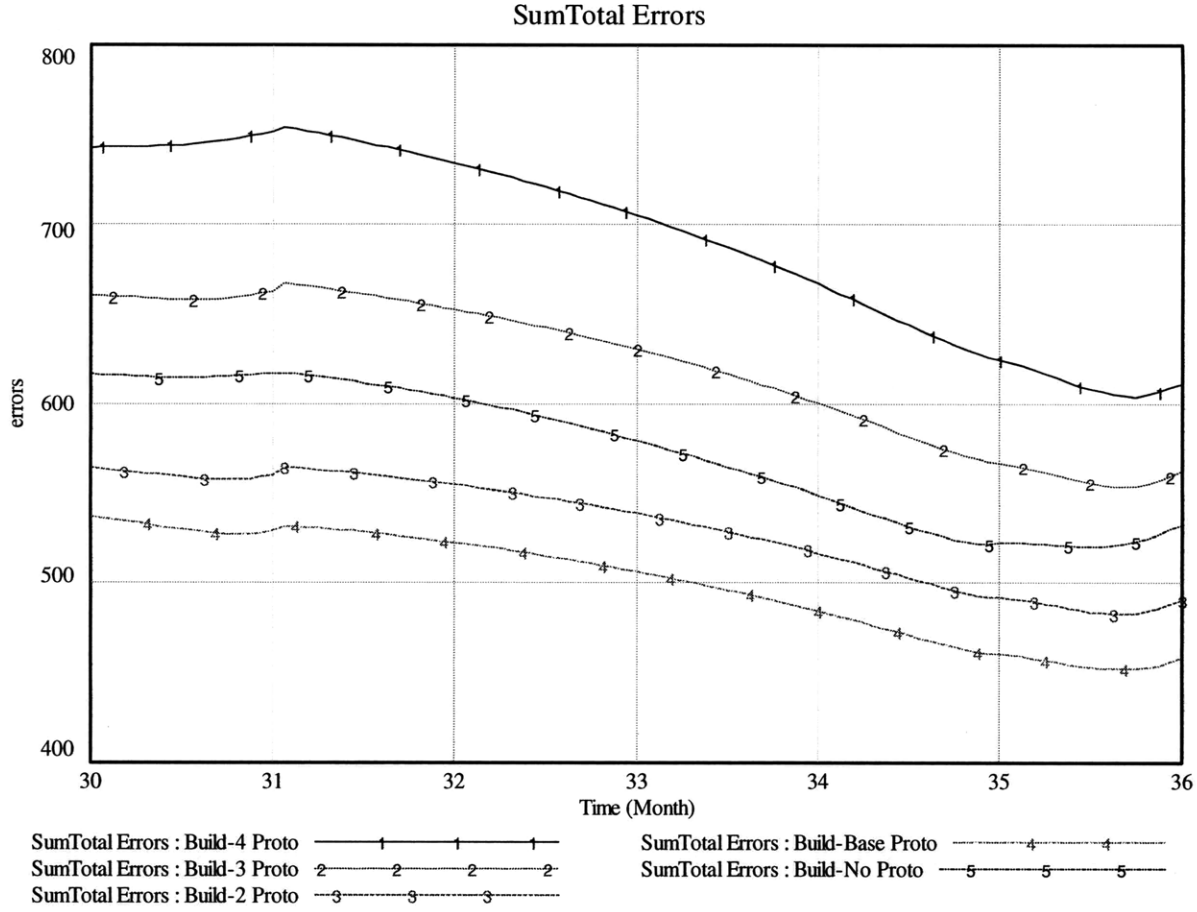


Figure 73 To Build or Not to Build? (Zoomed in View)

It is interesting to note that the *No Proto* build alternative actually outperforms all of the other options in terms of clean design churn. This is true because while the No Proto alternative results in more errors (and fewer clean designs) it does so while requiring fewer cumulative designs. These results are included below - again with only the last 6 months of the NPD project shown.

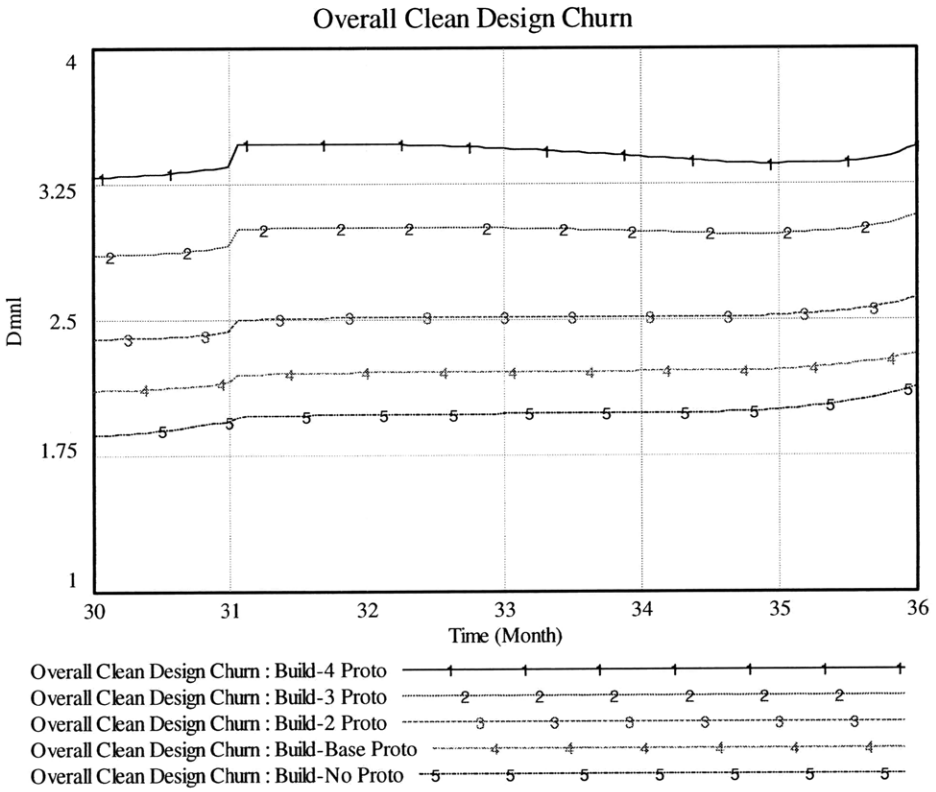


Figure 74 No Proto Build and Clean Design Churn

The reason that project performance decreases with the increased number of builds is that the quality of design work decreases with the increased schedule pressure that arises from the various build events. Because the quality of design dips, more rework is created which, when discovered, increases the schedule pressure even higher than it would be otherwise. This effect can be seen in the graph below which shows the cumulative errors generated for each of the prototype scenarios.

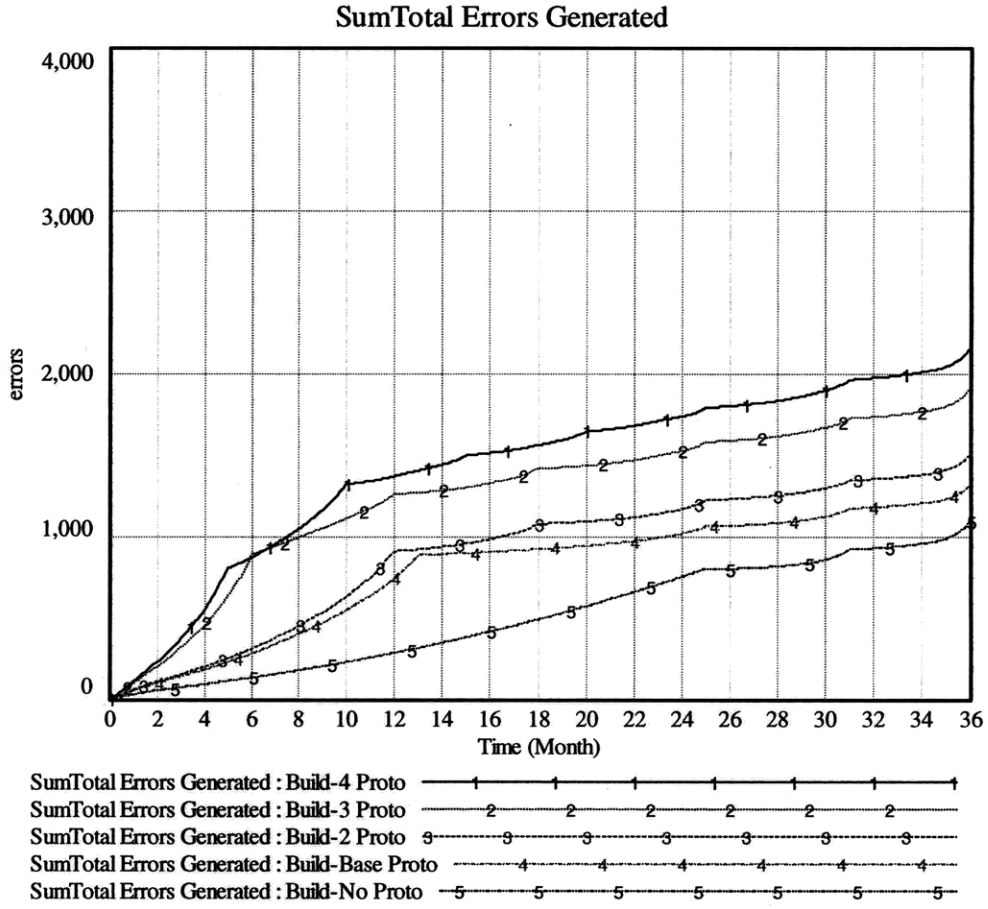


Figure 75 Total Errors Generated - Proto Build Alternatives

Given the results presented above, having a single prototype build results in the best overall quality since it has the fewest number of errors remaining at the time of product launch (see Figure 72). Assuming that the average NPD project manager would be willing to incur the "cost" of additional designs and an additional build in order to end up with fewer remaining errors at the time of product launch, it begs the question: *When is the best time to execute a single prototype build?*

To answer this question we can simply vary the date of the prototype build between the beginning of the project and the DIB build and observe the project performance. To do this we established an Early Proto at 6 months, the base Proto at month 13 and a Late Proto at 19 months.

The results are presented below. For comparison, we include the performance of the No Proto alternative as well.

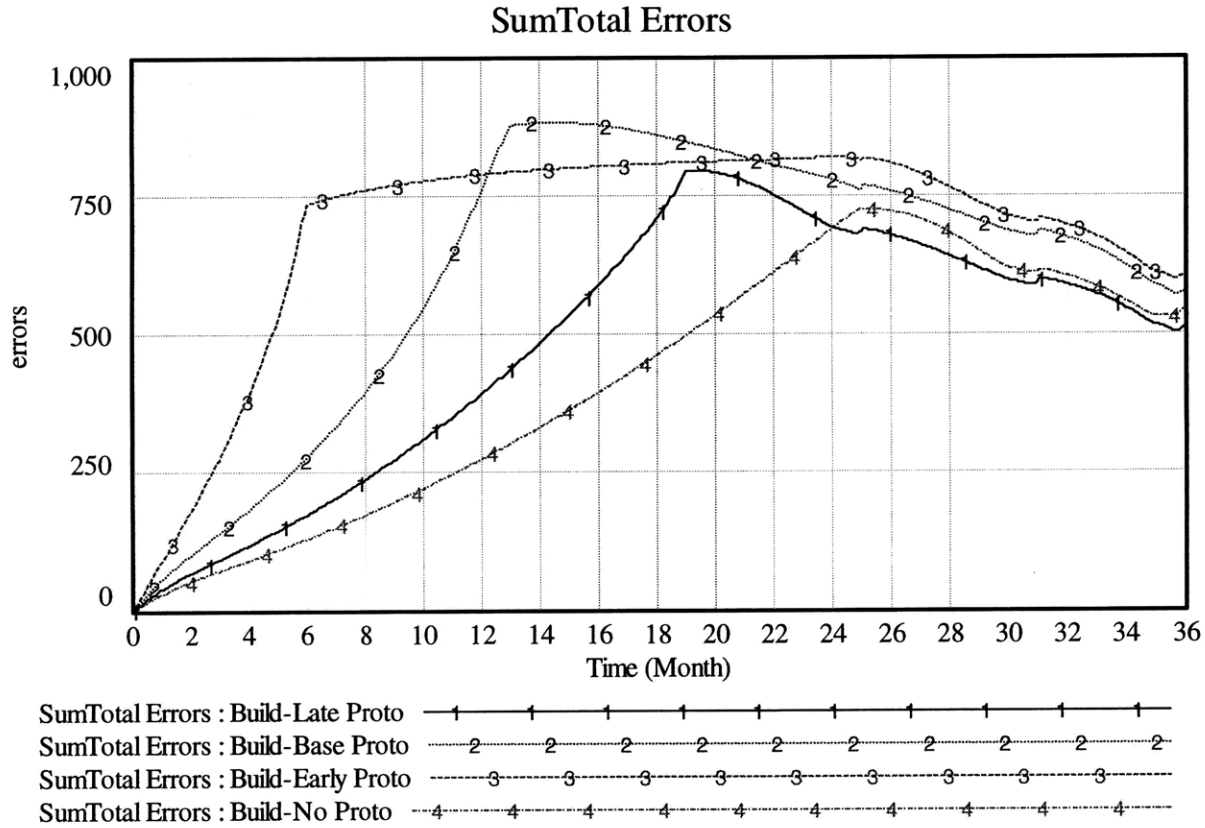


Figure 76 Proto Timing and Total Errors

The graph shows that the Late Proto build alternative results in the fewest number of remaining errors at the time of product launch with the No Proto alternative performing nearly as well. In fact the No Proto alternative performs the best in terms of Clean Design Churn and MOE1. Because of this, a decision as to when is the best time (or no time at all) to execute a proto build is subject to debate. In an effort to bring clarity to the question, we can look more closely at the performance of a single prototype build.

The graph below shows the total number of clean designs as of the product launch date given that a prototype build is executed on a given date (represented along the x-axis) which spans

the possible dates for a single prototype build event. Based on the graph, it appears that the optimal time to execute the single prototype build is month 21 in order to maximize the number of clean designs. However, with the design intent build (DIB) regularly scheduled for month 25 it might prove problematic to wait until month 21 to execute the prototype build. There is a clear tradeoff between providing sufficient time before the proto build to allow design engineers to work at a comfortable pace and not degrade the quality of their work due to schedule pressure and providing them with enough time to find and fix errors between the proto build and the design intent build. Additionally, from an organizational view point it may prove difficult to wait until month 21 to get the first "real" check on design progress.

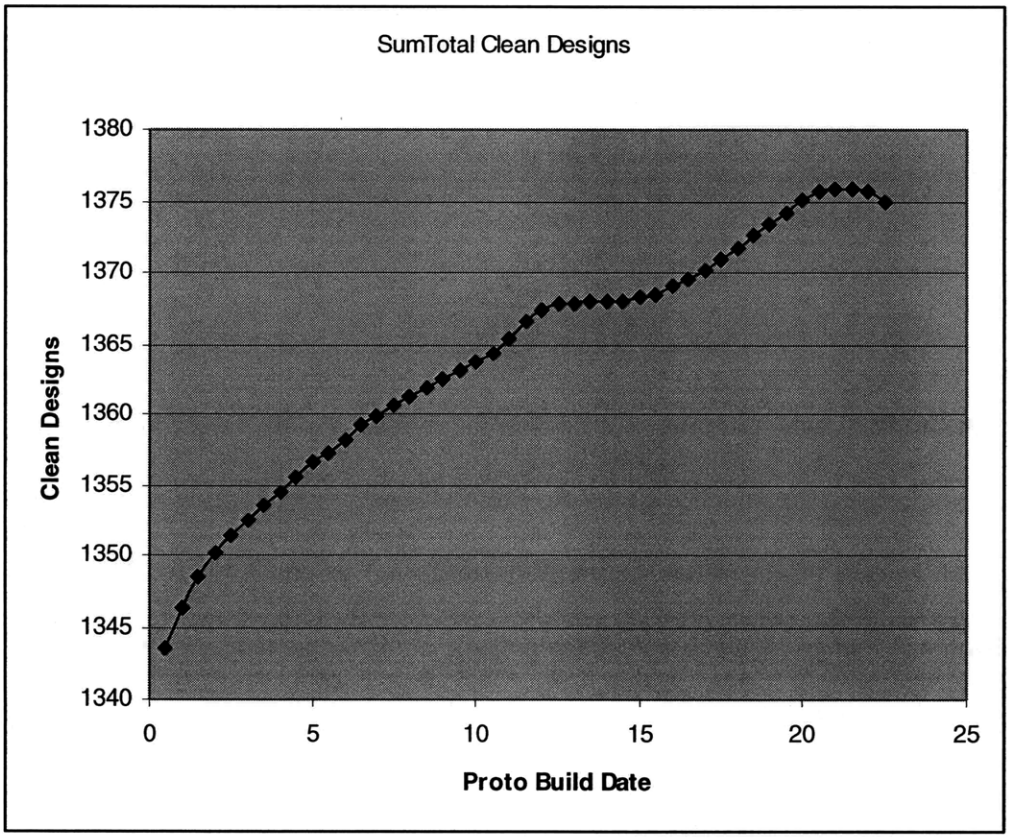


Figure 77 Clean Designs and Proto Build Dates

When you consider the project performance measure MOE1, you might conclude that it is better to do the prototype build much earlier. Figure 78 suggests that a prototype build as early as month 5 would be best. However, the extremely small range over which the MOE1 values fall makes this consideration virtually meaningless. This suggests that the previous results indicating a later prototype build is better are more noteworthy.

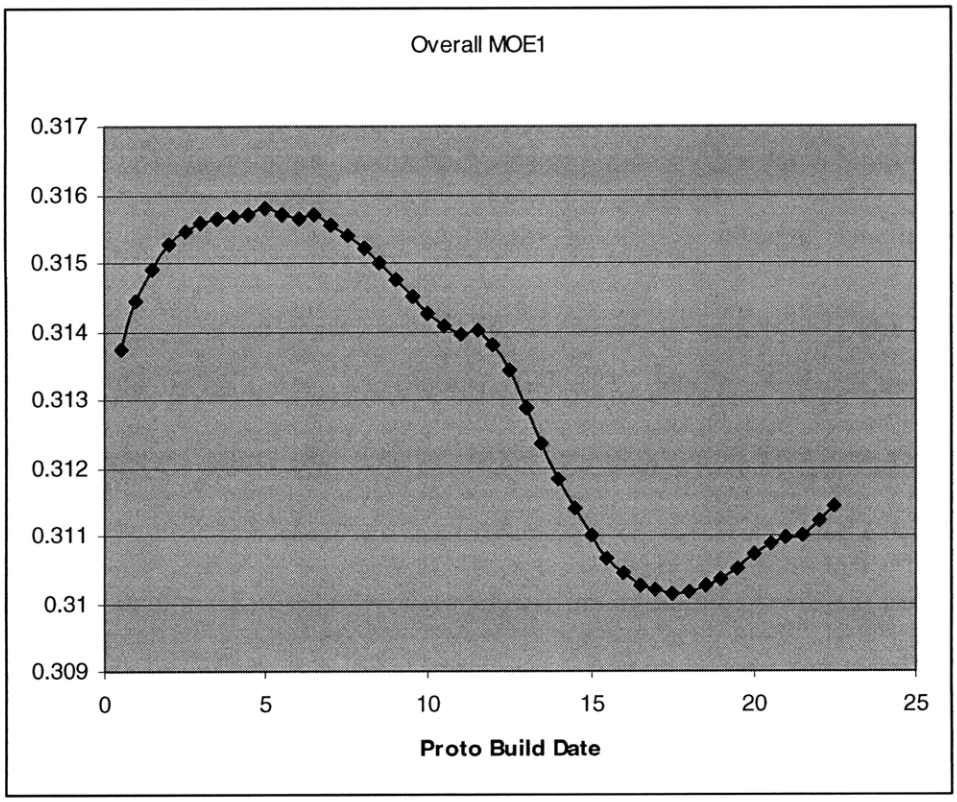


Figure 78 MOE1 versus Proto Build Dates

I further examine these relationships by varying the timing of each of the build events (Proto, DIB, and FPE) relative to the project start date and a fixed launch date. Doing so helps answer the question: *How should these build events be timed (or spaced) in order to maximize new product development project performance?* In order to answer this question, I set up a Monte Carlo simulation using Vensim. Based on the earlier findings where one prototype build appeared to

result in the best performance, I set up the Monte Carlo simulation to set dates for 3 build events so that I could observe the relative performance of various combinations of build dates. In setting up the Monte Carlo simulation, I split the NPD project development timeline of 36 months into 3 equal bins requiring a build to occur in each of the three bins. By splitting the development timeline into three equal bins, I prevent the simulation from trying to schedule a later build to occur before an earlier build. For example: scheduling the FPE build before the DIB build wouldn't make sense.

The result of the Monte Carlo simulation is a set of performance data for all of the possible combinations of build dates for a 3 build project. Given the data, it is easy to sort it according to a particular performance measure in order to select the set of dates that results in the best score for that performance measure. In this case, executing the following build schedule results in the best overall performance: Proto (month 13), DIB (month 21), and FPE (month 36). This build schedule not only scores the best in terms of MOE1 but it results in the lowest number of errors at product launch.

In order to test this "best" 3 build alternative from the Monte Carlo simulation, we can run a set of simulations using this build schedule against the base case scenario, the single Proto at month 21 alternative, and the "All Builds" scenario. A summary of the associated build dates for these alternatives is included below:

Build	Monte Carlo	Base Proto	21 Proto	All REAL Builds
Proto1	**13**	**13**	**21**	**5**
Proto2	NA	NA	NA	10
Proto3	NA	NA	NA	15
Proto4	NA	NA	NA	20
DIB	**21**	**25**	**25**	**25**
DIB2	NA	NA	NA	28
FPE	**36**	**31**	**31**	**31**
FPE2	NA	NA	NA	33.5

Table 5 Build Schedule Dates

As you can see in the graph below, the *Monte Carlo* alternative does not appear to perform better than the other build schedule alternatives until the very end of the simulation at month 36 – product launch. However, it is clear that the *All Builds* alternative is outperformed by the other alternatives. This relative performance holds true when comparing the number of clean designs achieved by the various build event timing options.

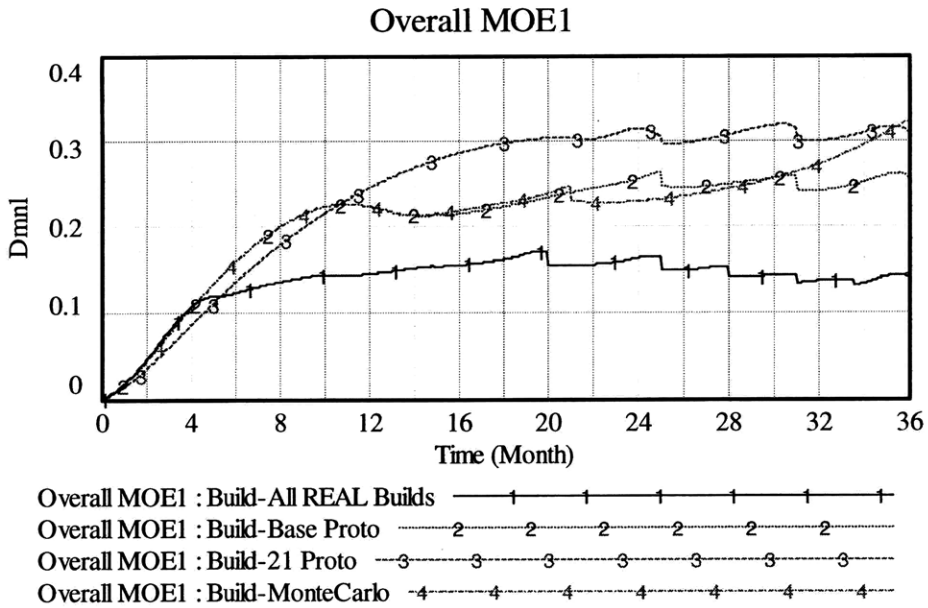


Table 6 Monte Carlo Performance Comparison

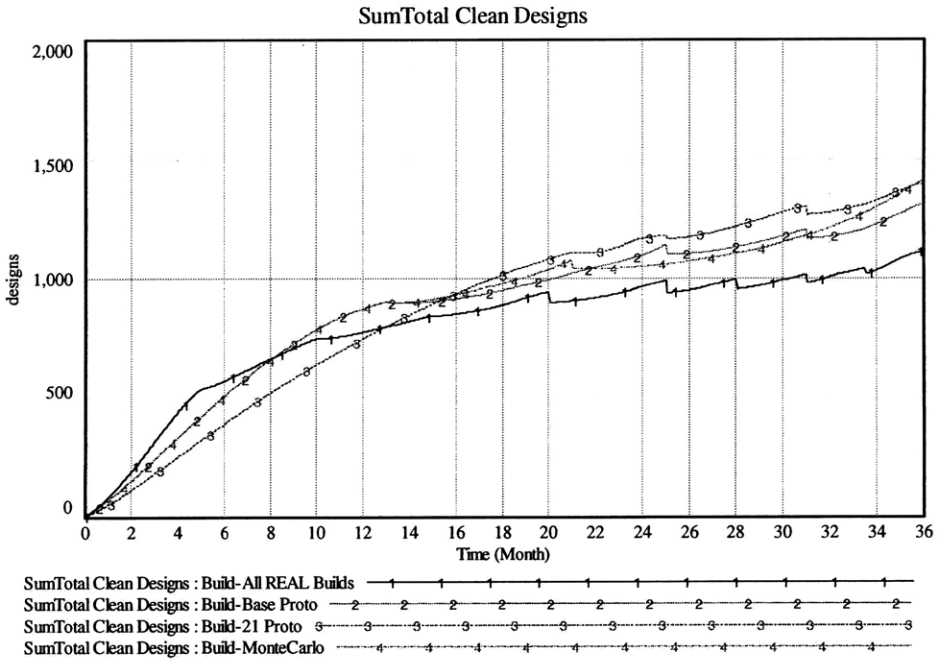


Figure 79 Clean Designs - Monte Carlo Comparison

The obvious drawback to the Monte Carlo suggested build plan is that it suggests that the first production event (FPE) build not occur until month 36 which is also the intended launch date. From an organizational perspective, not conducting a full up production parts and production process test build prior to the launch date is probably unacceptable.

In this section, I examine how the above model behavior and relationships are affected by the degree to which the design iteration cycle times for the coupled subsystems are different and the degree to which the subsystems are coupled. As mentioned earlier in the sensitivity analysis discussion of the various model parameters, the NPD Model showed sensitivity to the degree to which the design iteration cycles for the coupled components are imbalanced (*Speed Factor*), the degree to which the components are coupled (*Fraction of Coupled Designs*) and the probability that a design change in one component will require a change in a coupled component (*External Rework Probability*). In order to examine how the model behavior is affected by these parameters, I conducted experiments varying the relative design iteration speeds of the slow and fast iterating components and comparing the results. This is done using the *Speed Factor* parameter. I also vary the degree to which the subsystems are coupled by varying the number of parts that the components have that are coupled. This is accomplished using the *Fraction of Coupled Designs* parameter. Reducing the number of coupled parts within the components reduces the degree to which the components are coupled. The third parameter that directly impacts on project complexity is the external rework probability. This parameter will be discussed in more detail below.

The next series of graphs shows the relationship between the *Fraction of Coupled Designs* and the various performance measures when the components operate at different speeds. The performance measures in the graphs indicate the value of the given measure (e.g. Clean Designs) at

the time of product launch (month 36 of the simulation). In these experiments, the *Speed Factor* is set to either 1, 2 or 3 where a speed factor of 1 indicates that both components operate at the same speed, 2 indicates that component A operates twice as fast as component B, and a speed factor of 3 indicates that component A is three times as fast as component B. In all cases the average speed of design and testing between the components remains the same. In effect, as one component speeds up (normally component A in the model) the other component slows down to keep an overall average work speed consistent across simulations.

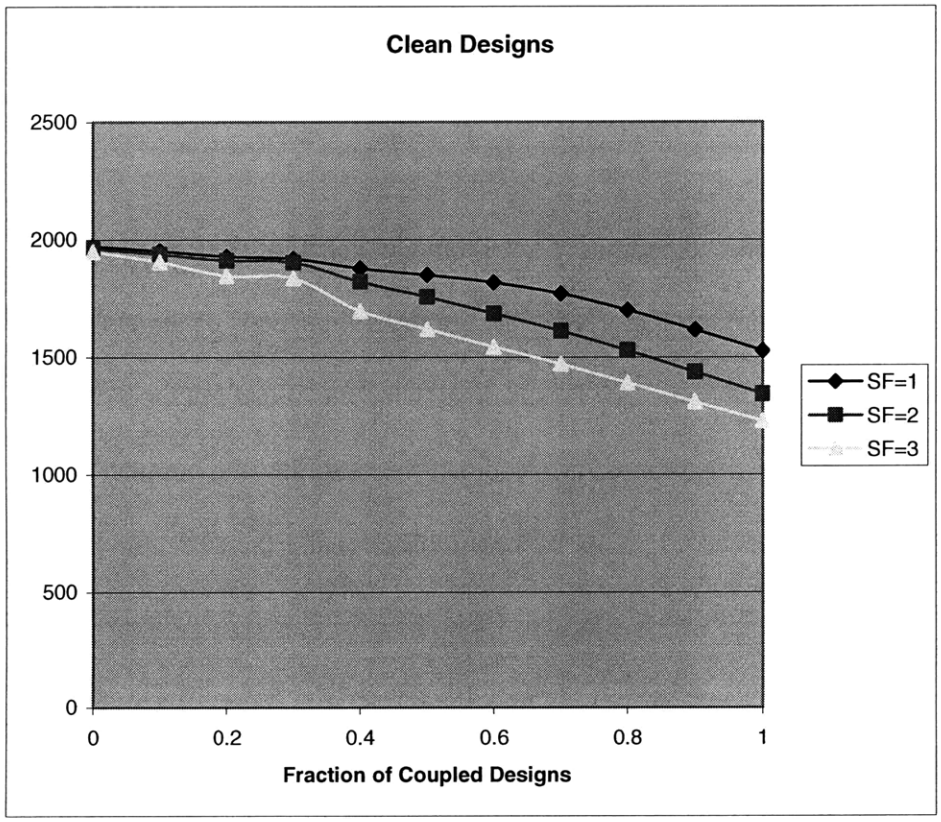


Figure 80 Coupled Designs, Speed Factor and Clean Designs

The graph above shows clearly that the number of clean designs completed during a given project goes down as the fraction of coupled designs increases. The effect is more pronounced when the two components work at increasingly different speeds as indicated by the speed factor.

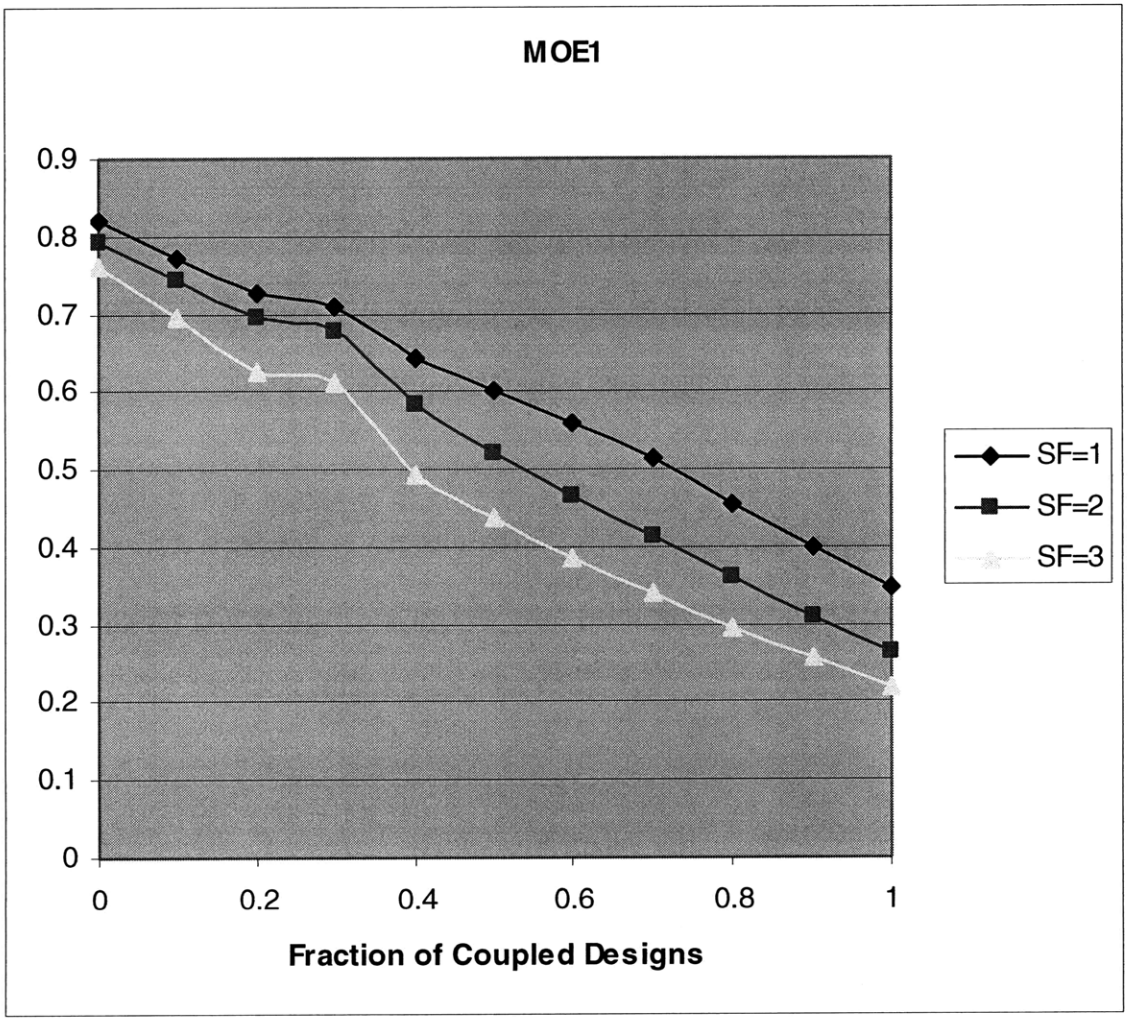


Figure 81 Coupled Designs, Speed Factor and MOE1

When using MOE1 as a measure of performance, as in the last scenario, performance gets increasingly worse as the speed factor increases, essentially placing the two components further

and further out of synch.. Regardless of the fraction of coupled designs, the performance of a project with a speed factor of 3 performs worse than one with a Speed Factor of 1 or 2.

The third parameter that directly impacts on project complexity is the external rework probability. As this probability increases, the linkage between the components and thus the complexity of the project is increased as coupled components that have integration errors require rework from both components more frequently. This situation results in more overall work and the reduction of quality as designers work to keep up with the schedule pressure. The end result is a decrease in the number of clean designs as the external rework probability increases as seen in Figure 82. The effect of the two components working at different speeds further degrades the performance.

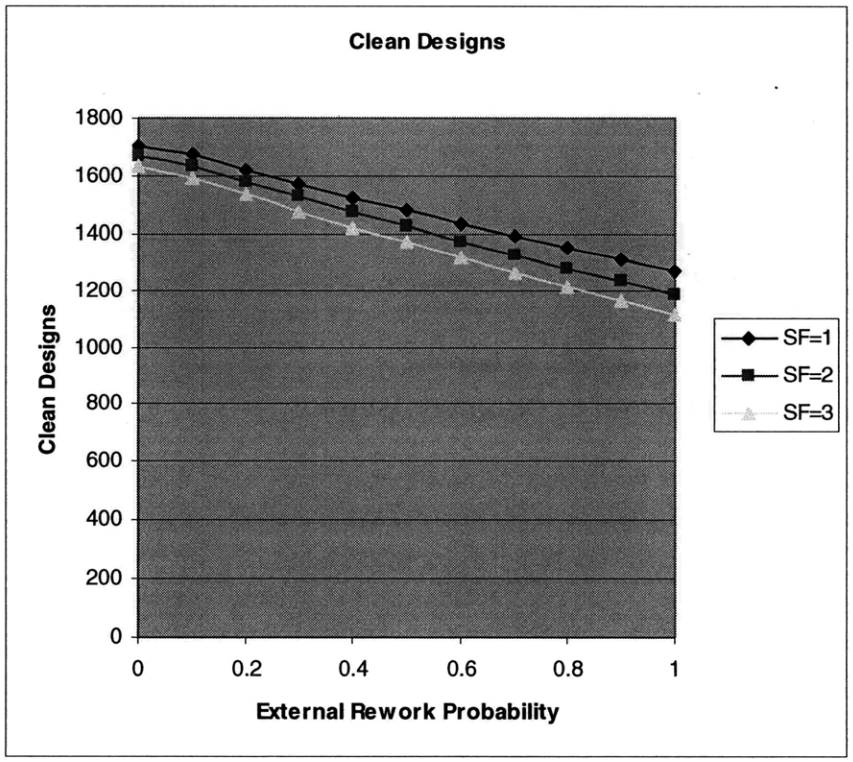


Figure 82 Clean Designs and External Rework Probability

The same effect can be seen in the graph of MOE1 below where project performance decreases with the increase of *External Rework Probability* and is further degraded with the increase in the *speed factor.*

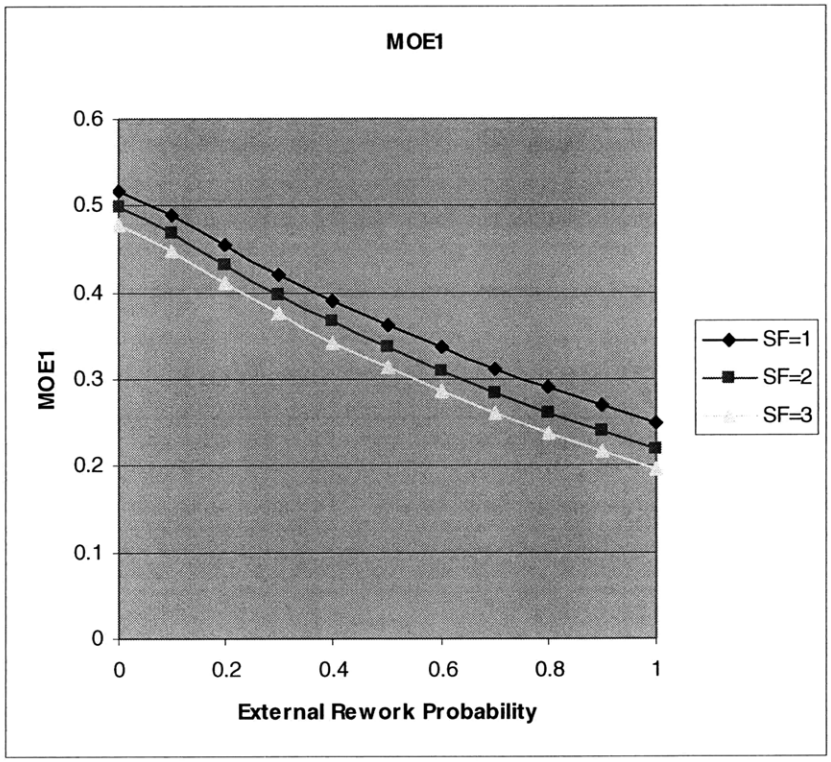


Figure 83 MOE1 and External Rework Probability

One might expect that the overall project performance would degrade as the complexity of the project increases. Here the complexity is reflected in the fraction of coupled designs and the external rework probability. And indeed the NPD model shows that project performance does indeed decrease as complexity increases. However, not as obvious but shown just as clearly is that project performance also decreases as the design iteration cycle times become imbalanced between

components despite the overall (or average) iteration time remaining the same. This result can be attributed to the amount of unnecessary, or phantom, work that is created when the components iterate at different speeds.

Figure 84 shows the phantom work that each component performs over the course of the NPD project. You'll note that the amount of work that the slower iterating component (in this case Component B) performs is more than the faster iterating component. Much of the extra phantom work performed by the slower iterating component is in work that it does after a build event on "obsolete" designs.

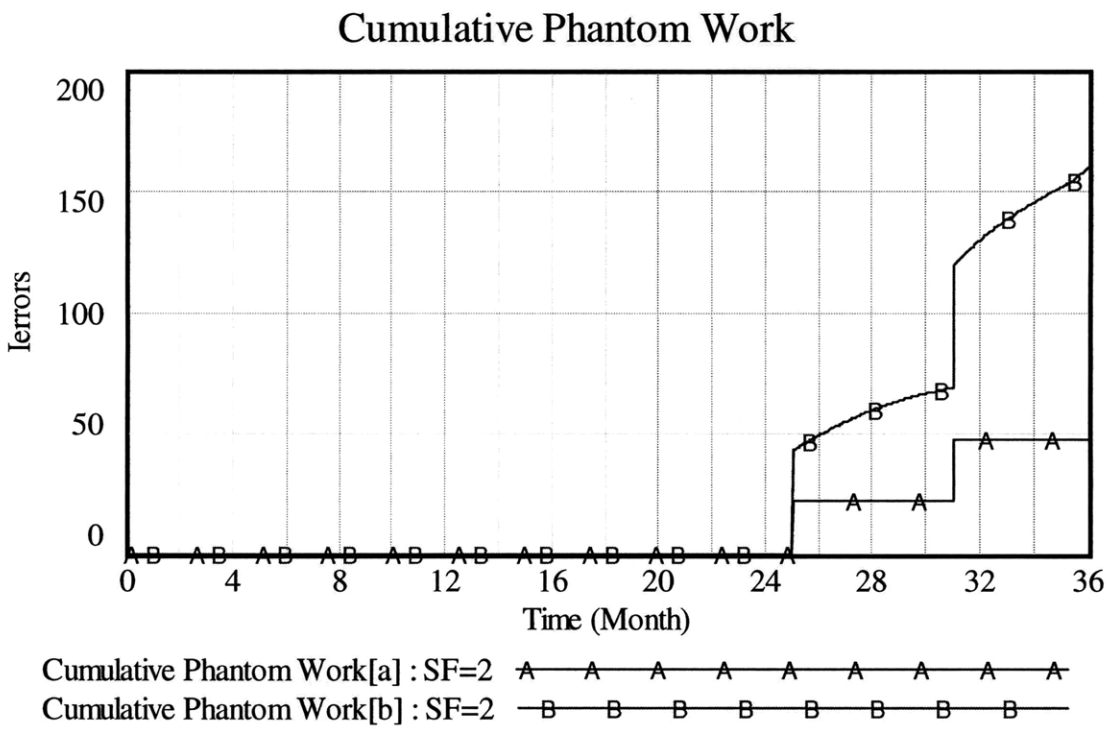


Figure 84 Phantom Work (between Components)

The amount of phantom work is clearly affected by the relative speed imbalance between coupled components. Figure 85 shows the overall amount of phantom work done by both components relative to the speed imbalance between components. You'll note from the graph, when the speed factor is 1, indicating that both components iterate at the same speed, there is no phantom work done at all. As the speed factor increases to 2 and then 3, the amount of phantom work increases significantly. This would indicate that as the two components operate increasingly out of synch, more and more unnecessary work is being done. This phenomenon was experienced and cited by numerous design engineers at the Company.

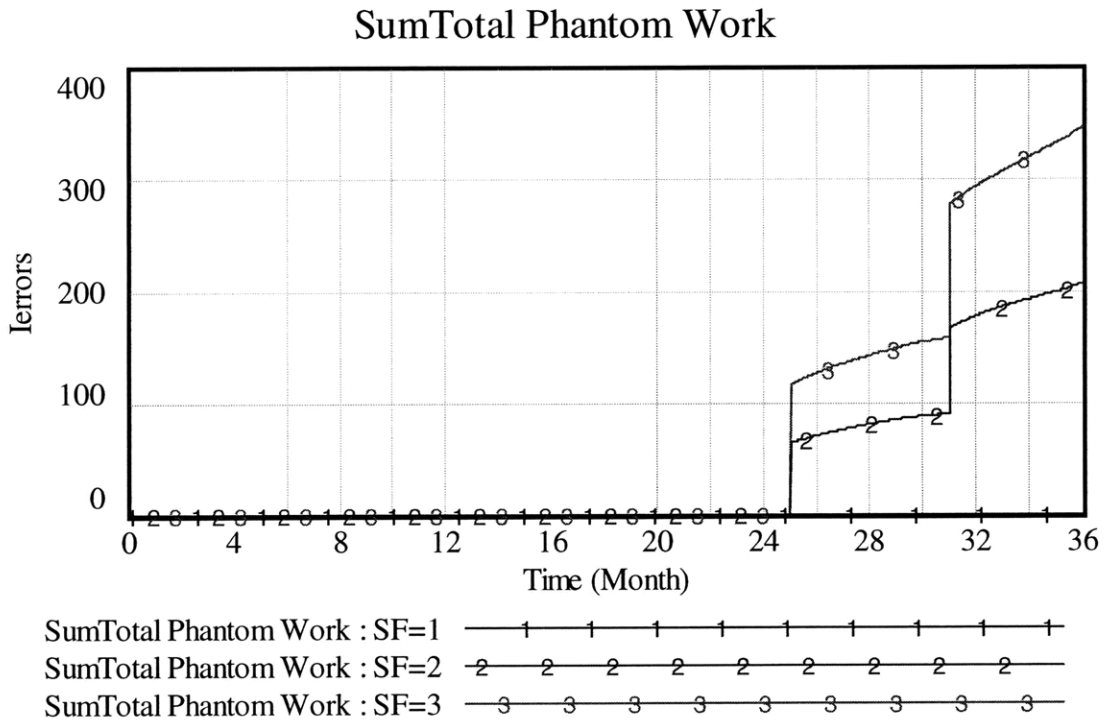


Figure 85 Phantom Work (relative to Speed Factor)

Policy Analysis

In this section of the analysis, we will look at some potential policies that the Company might consider to improve the performance of it NPD projects. For the purposes of this analysis, we will look at Project ATOMac and consider the base case which incorporates the policies as they existed during the project and compare it to various policies aimed at improving project performance.

The first policy that we will examine is one in which we would execute fewer builds than the ATOMac project by executing a single prototype build at 12 months and the required DIB and FPE builds. This schedule is in contrast to the 8 builds executed in Project ATOMac.

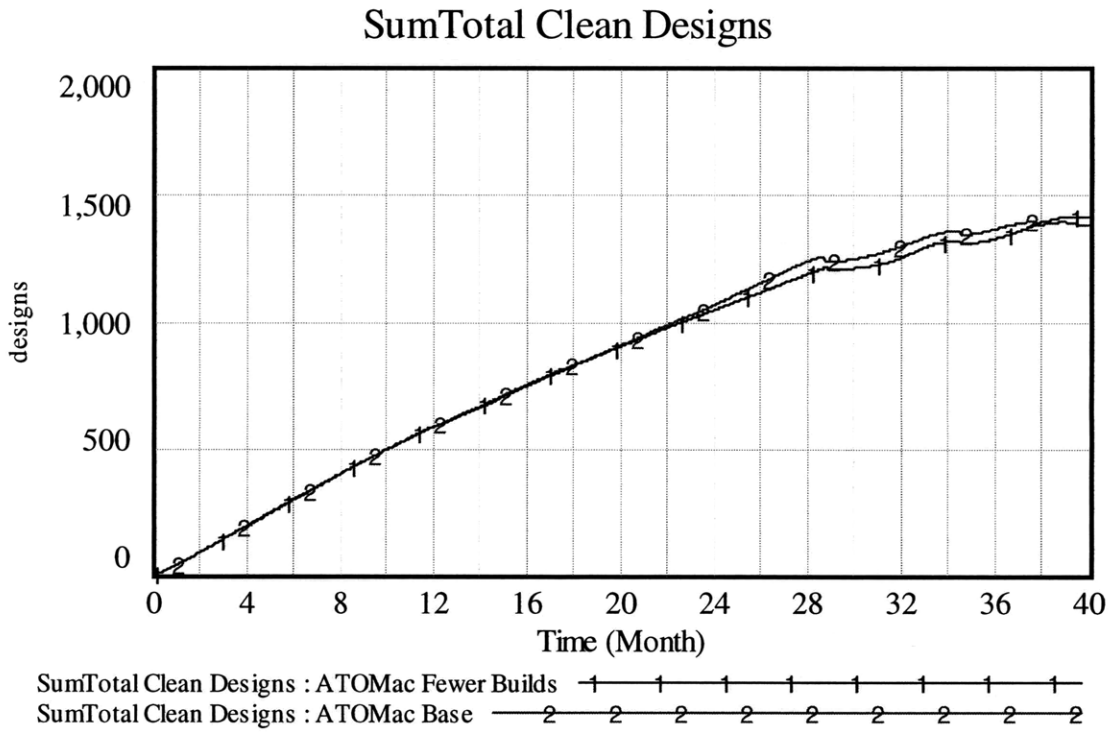


Figure 86 Policy Analysis (Fewer Builds - Clean Designs)

The graph above shows that by reducing the number of build events, the total number of clean designs is slightly improved (1,415 to 1,393) at month 39 when Project ATOMac launched. While

this does not seem like a large improvement, when one considers the total number of designs done (2,957 for the fewer builds versus 3,254 for the original case) to achieve the number of clean designs the performance improvement is more pronounced. This improvement is reflected in the graph of MOE1 below.

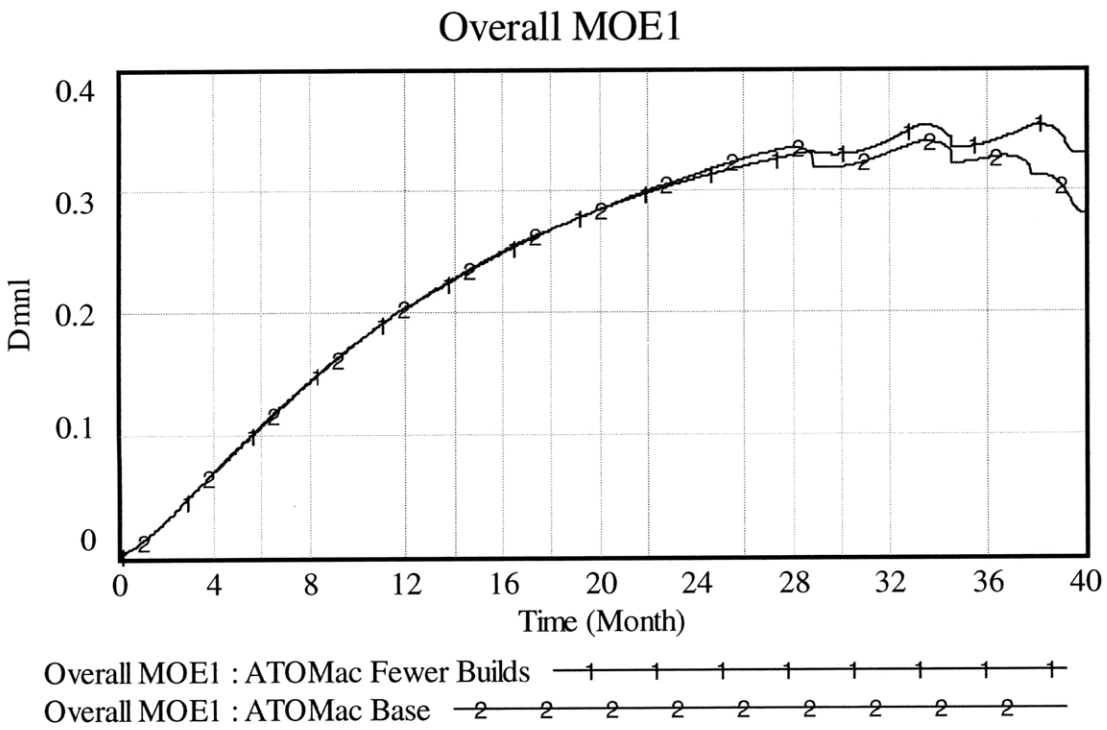
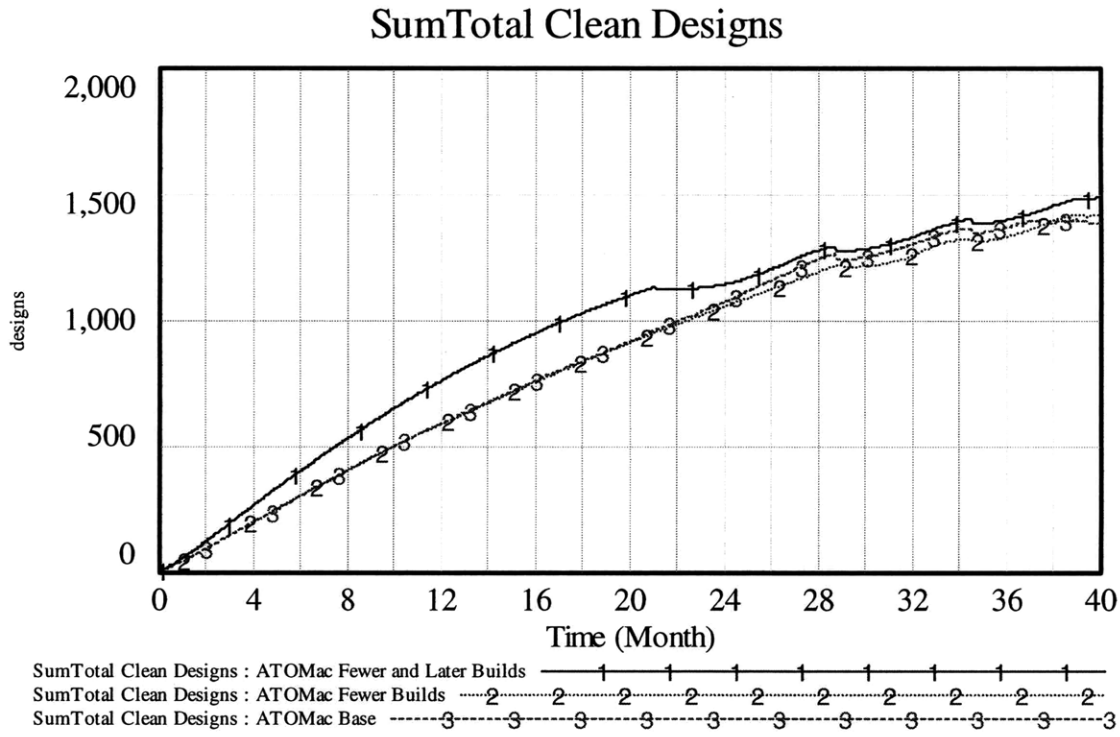


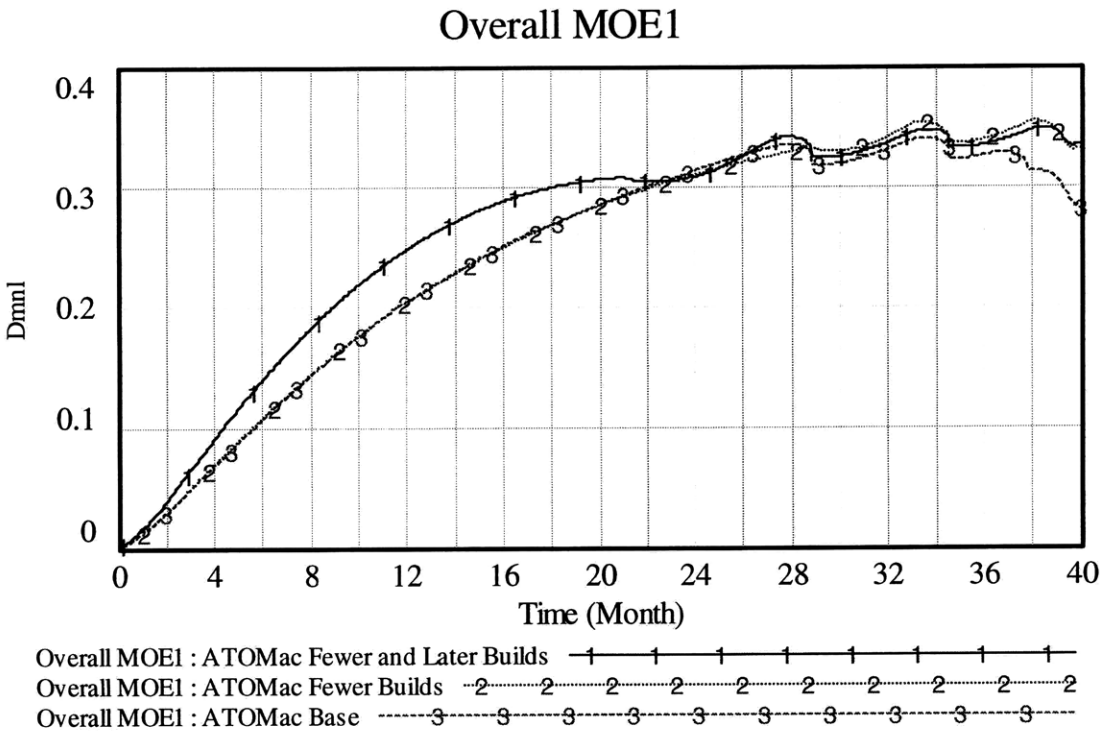
Figure 87 Policy Analysis (Fewer Builds - MOE1)

Perhaps more important than the design work saved by requiring fewer designs to complete more clean designs is the money saved by executing six fewer builds including two builds (a second Design Intent Build and a second First Production Event) which require a shutdown of the normal production line. Additionally, there are huge savings achieved by reducing the number of build events in terms of the workload and cost procuring and/or prototyping parts for the build events and the resulting test and analysis that is done on the vehicles.

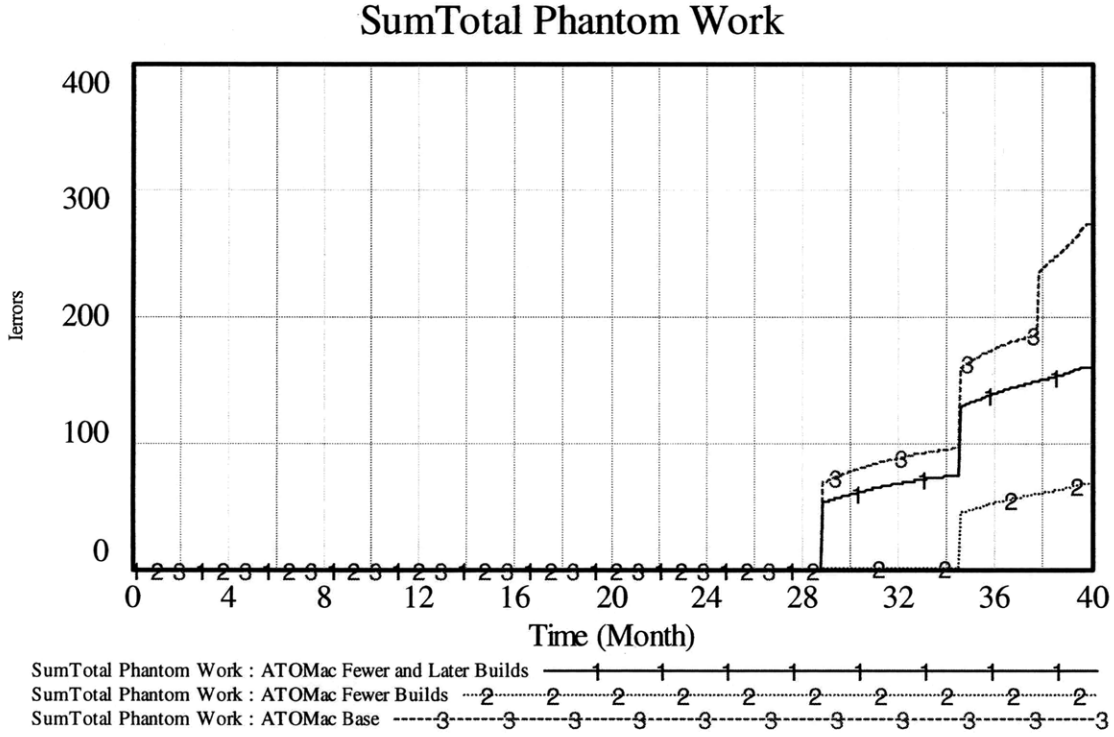
In light of the improved performance based on conducting fewer builds, we consider a second policy which again limits the build events to three but also delays the prototype build until month 21 as suggested by our earlier analysis. Included in this policy is the assumption that the designers will take this prototype build seriously given the late date at which it occurs. This means that the designers will feel and react to schedule pressure to make sure that they have their designs completed prior to the prototype build event. The results of this policy are shown below.



In this case we can see an even larger improvement in the total number of clean designs completed as of the launch date. In fact, the number of clean designs for the new policy is 1,481 compared to 1,396 clean designs in the original policy. Similarly, the overall performance as measured by MOE1 is better with this policy as indicated in the graph below.



Perhaps surprisingly, the overall performance of the Fewer and Later Builds policy is not improved compared to the Fewer Builds policy. Indeed at times throughout the project schedule the overall performance is worse. This behavior is the result of the Fewer and Later policy causing the two components to operate out of synch as the designers react to the schedule pressure created by the later prototype build. As we have shown, this mismatch in design iteration cycle timing creates phantom work which adds to the total work done and thus reduces the overall performance as measured by MOE1. A graph of the phantom work generated by the three policies is shown below.



Next we examine the effect of adding resources to a given project on the project's outcome to answer the question: *What is the impact of adding resources to a given project on the overall project performance?* We can do this by varying the number of designers assigned to each of the components[12] and manipulating when these resources are added. Quite naturally, we would assume that adding resources to a given project would increase project performance in terms of the development time and quality level attained. Indeed, this improved performance is achieved when we increase the initial number of designers assigned to the project as indicated in the sensitivity analysis of parameters previously discussed. Of course, any improvements in project performance need to be considered in light of the additional cost incurred by adding the additional resources. This will be discussed in context below. Perhaps more importantly, here we examine the impact

[12] We are assuming that designers can only work on one of the two components as they each represent sub-systems that are developed by different functional groups (e.g. power train and chassis) within the development organization.

on performance of *when* additional designers are added to the project's labor pool. The aim here is to examine the effect that adding these additional resources will have on overall project performance (both the quality and the total amount of work done) as well as the amount of phantom work generated.

In the context of the policy analysis above, in the original execution of Project ATOMac additional designers were added to the project very late in the project schedule in reaction to schedule pressure resulting from the approaching product launch date. Indeed, this policy is modeled in the base NPD model – ATOMac Base. The policy we examine here suggests that the Company should add resources in response to schedule pressure that is based on the next "real" build event. This means that the project team should add labor as necessary in an attempt to complete its design work prior to the DIB and FPE builds as well as to complete its work prior to the product launch date. This policy is indicated in the graphs below as *ATOMac - Labor*.

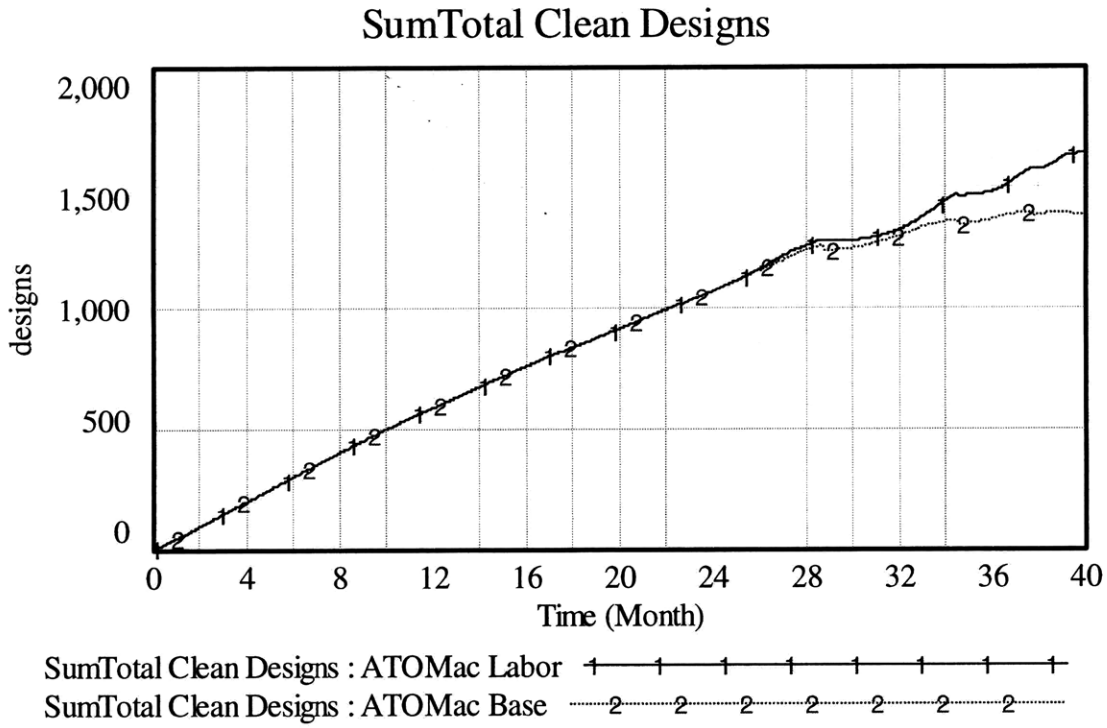


Figure 88 Policy Analysis (Labor - Clean Designs)

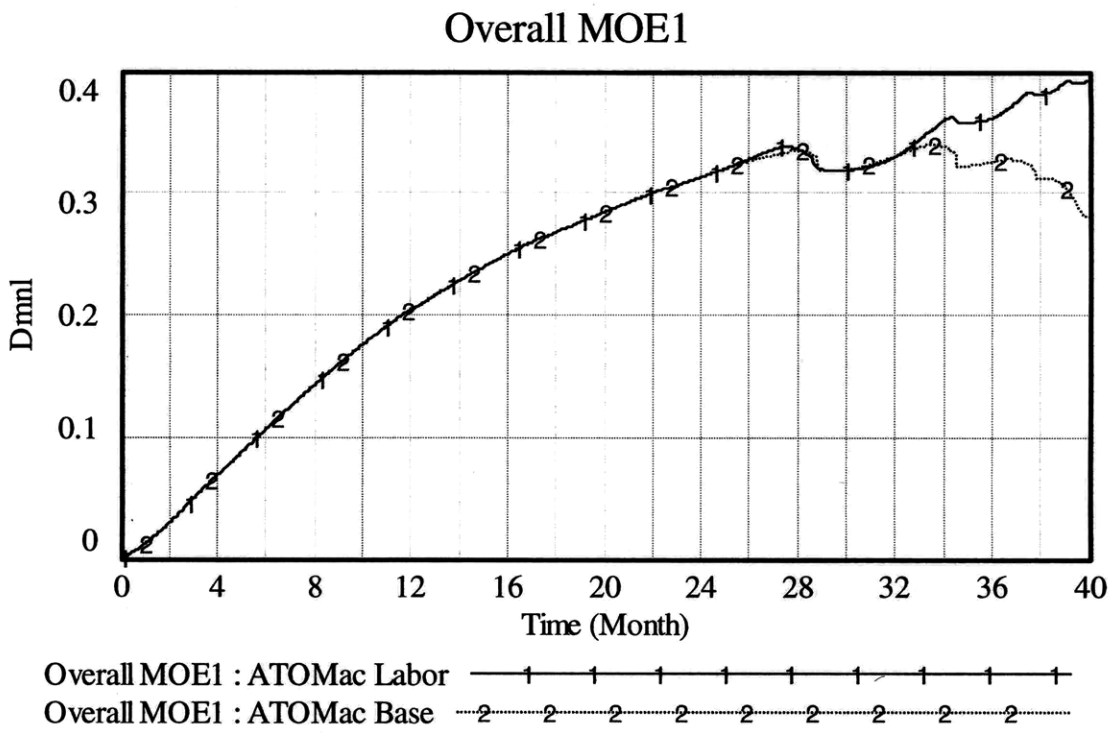


Figure 89 Policy Analysis (Labor - MOE1)

Here we see a significant improvement in both the total number of clean designs at launch and the overall performance as measured by MOE1. Indeed the number of clean designs increases from 1,396 in the base ATOMac model to 1,620 in the improved labor policy model. Additionally, the total amount of phantom work generated decreases significantly as indicated in the graph below.

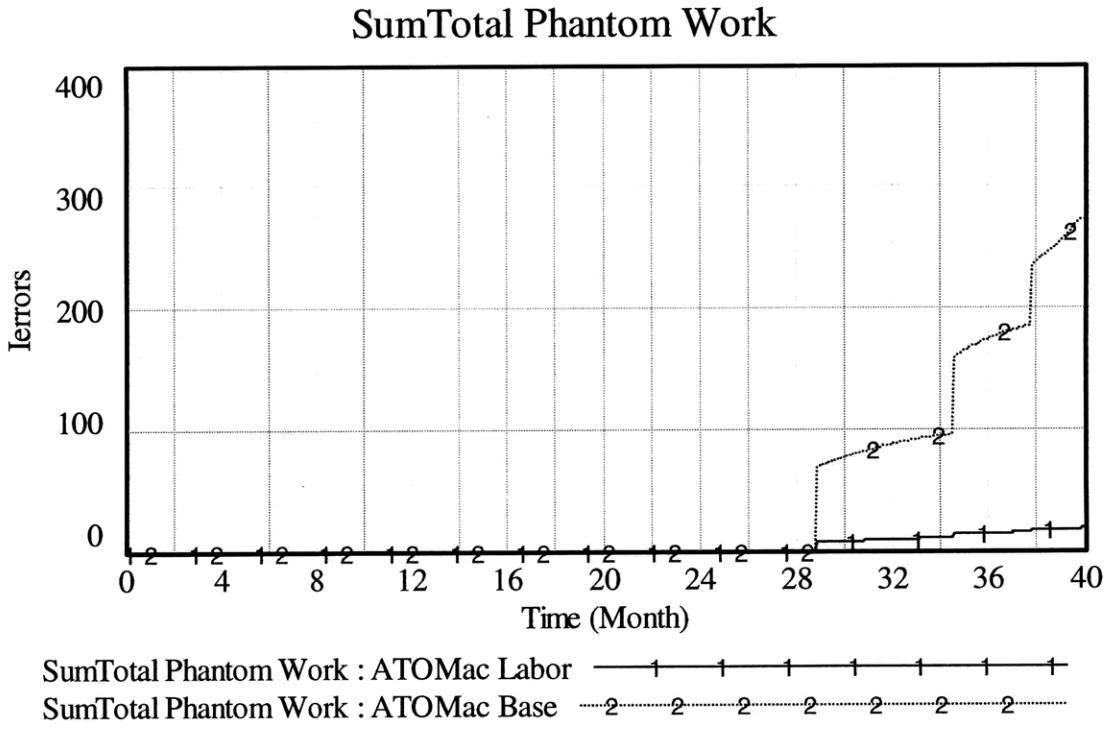


Figure 90 Policy Analysis (Labor - Phantom Work)

However, as expected the improved performance generated by adding labor sooner to the project comes at a cost. The graph below shows the labor profile for the project timeline under the two different labor policies. The end result is that the more aggressive labor policy results in an additional labor cost of almost 240 designer-months which is the equivalent of 20 designers for a one year period. This is a significant labor investment given the initial number of designers assigned to the project was 20 designers (10 for each component).

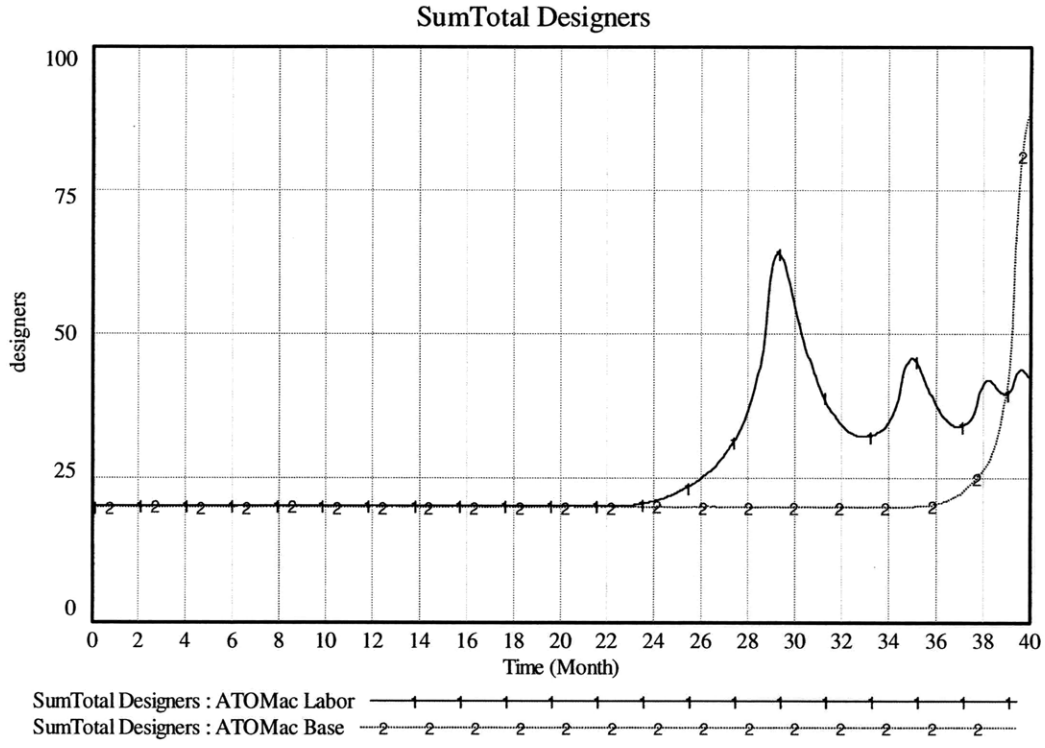


Figure 91 Labor Profile

In light of the improved project performance in terms of quality that results from adding labor resources more aggressively, it seems possible that we may be able to combine this policy with one that calls for fewer builds and achieve even better results. The reduction of the number of costly build events alone may more than pay for the cost of the additional engineering man-hours added to a given project. The performance results of this Combined Labor and Build policy are shown in the graphs below.

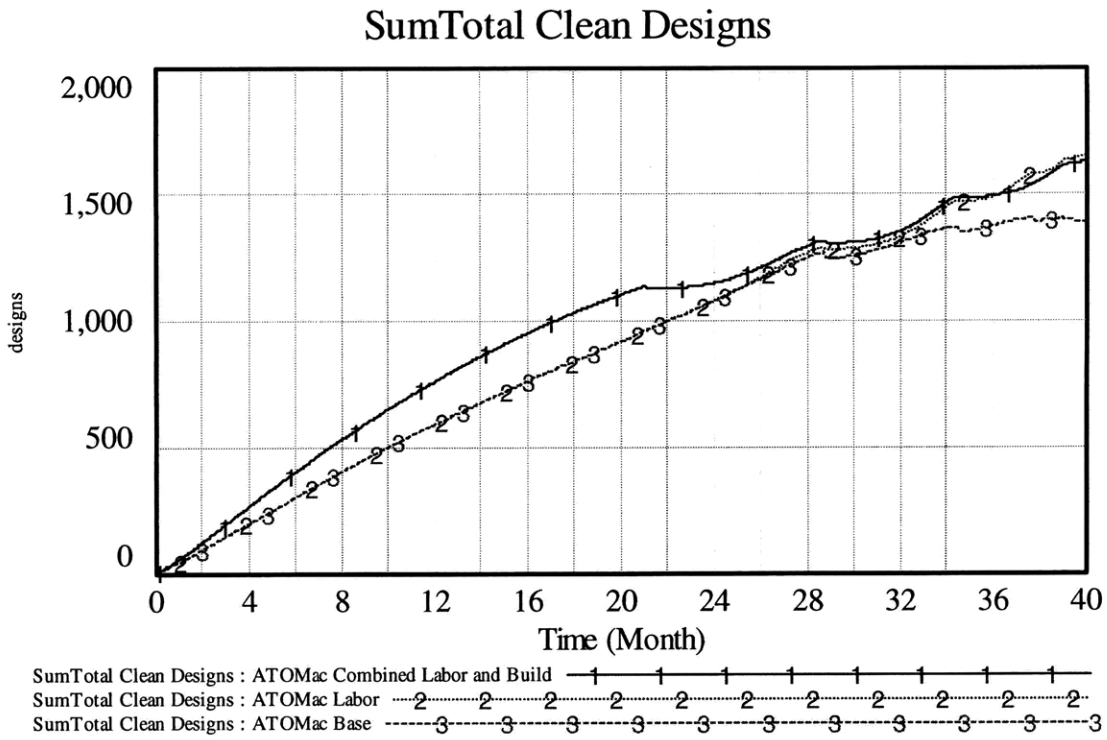


Figure 92 Policy Analysis (Combined - Clean Designs)

The graph above of clean designs shows that the combined policy achieves near identical results as the aggressive labor policy in terms of the number of clean designs. However, in this case it achieves this performance with 5 fewer build events. The overall performance in terms of MOE1 is similarly identical as shown below.

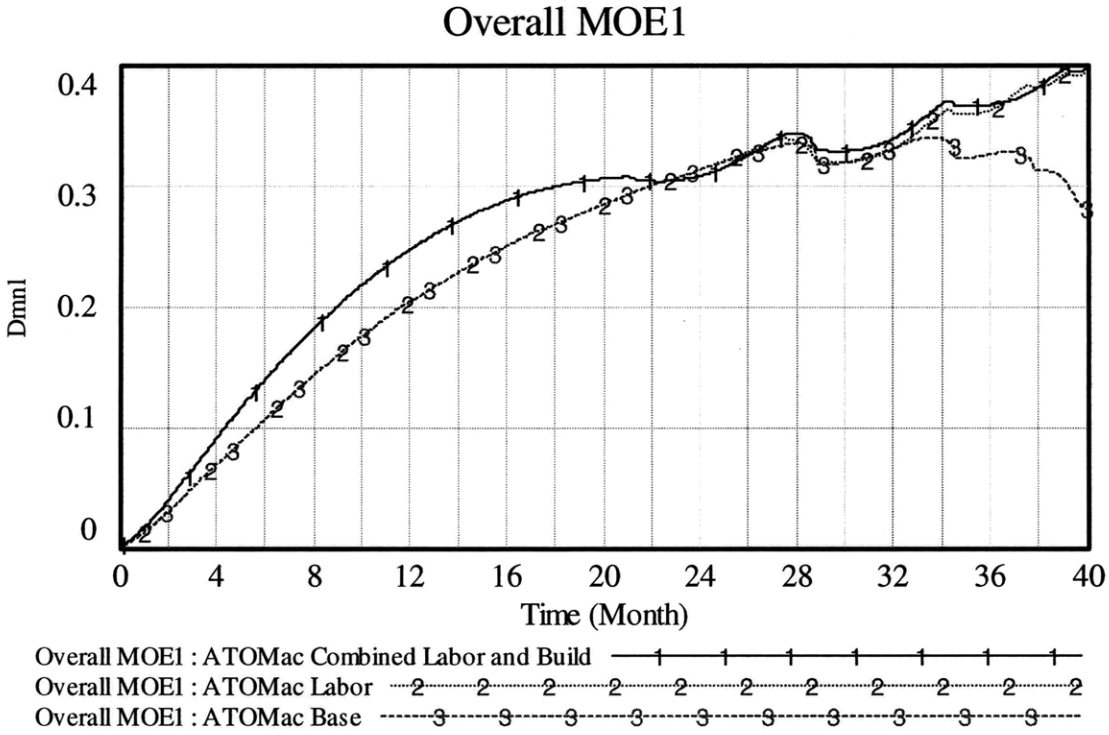


Figure 93 Policy Analysis (Combined - MOE1)

In addition to the cost savings achieved by the Combined policy in terms of fewer costly build events, this policy also results in fewer labor resources required as compared to the aggressive Labor policy. In fact, the combined policy results in 48 fewer designer-months of labor required for the project. The labor profile comparing these policies is shown below.

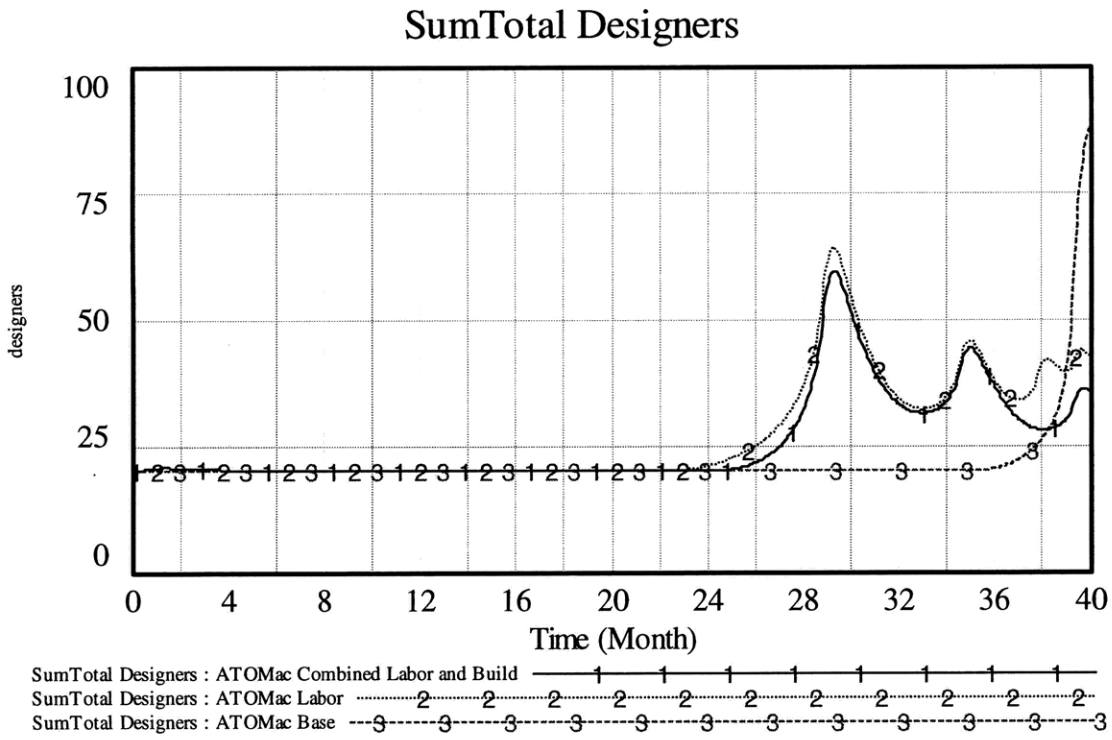


Figure 94 Labor Profile (Combined vs Labor Policies)

While the Combined Labor and Build policy still results in an increase of about 188 designer-months of labor over the base Project ATOMac policy, the reduction of costs achieved by requiring five fewer build events combined with the significant improvement in project quality performance seems to more than justify this increase in labor.

Analysis Conclusions

The analysis conducted here using the NPD Model and Project ATOMac as a case study seems to suggest a number of findings. First, of the many parameters used in the NPD model the model results are sensitive to just a few of the parameters. These include: base design and redesign quality (both component and integration), base design and redesign productivity, the number of initial designers assigned to the project, the relative speed (*Speed Factor*) at which the two

components iterated through the design-build-test learning cycle, and the level of complexity of the project in terms of the number of coupled components (*Fraction of Coupled Designs*) and the probability of a change in one component requiring a change in a coupled component (*External Rework Probability*).

In terms of project performance as predicted by the NPD model, having more build events does not result in improved project performance either in terms of the quality attained or in the cost of executing the project schedule, where cost is measured in terms of the number of designs required and the number of build events required. Indeed, quite the opposite is true. Executing more builds typically results in a decrease in the quality achieved and an increase in the project costs. While this finding was suspected by the Company, the NPD Model is able to demonstrate it clearly.

Additionally, while the scheduling (i.e. spacing) of the build events affects project performance there is no clear best policy. The scheduling of three builds at 12 (Proto), 21 (DIB,1 and 36 months (FPE) seems untenable given the effective elimination of the FPE build given that the product launches at 36 months. However, even if we fix the DIB and FPE build events to the original schedule set by the Company there seems to be some benefit in delaying the initial prototype build in that it allows designers to work at a reasonable pace and not induce errors due to cutting corners. However, delaying the initial build too long seems problematic from an organizational view point in that it means that the organization has to wait longer in order to get a "reliable" status of the project. Remember, the Company has become reliant on the build events to do its "learning". Secondly, delaying the initial build too long leaves less time for rework for those designs found to be flawed and can cause an increase in the schedule pressure felt by designers and thus cause them to work faster and perhaps out of synch with their coupled components inducing errors and phantom rework. This is particularly troublesome in that we found through our analysis

that the degree to which the components work out of synch definitely impacts on project performance.

Lastly, a more aggressive labor policy seems to be a worthwhile endeavor. While adding designers to the project sooner results in an overall larger labor cost, the resulting quality improvements seem to justify the additional cost. When combined with a policy of fewer build events, the additional cost of aggressively adding labor is seemingly subsumed by the cost reductions achieved by holding fewer build events. While the Combined Labor and Build policy still results in an increase of about 188 designer-months of labor over the base Project ATOMac policy, the reduction of costs achieved by requiring 5 less build events combined with the significant improvement in project quality performance seems to more than justify this increase in labor.

Chapter 6: Summary and Conclusions

> ***"The outcome of any serious research can only be to make two questions where only one grew before."***
>
> -- Thorstein Veblen, US Economist and Social Philosopher (1857 – 1929)

Summary

As companies compete in an increasingly global and competitive market, bringing new products to the market more quickly has become critical to gaining market share, increasing profits and creating and sustaining growth. The company that can develop products more quickly than its competitors can introduce new products sooner than its competitors and thus gain first mover advantage. Early entry into the market often leads to higher profitability as revenues are generated sooner and the early company is able to grab a greater share of the overall sales. This is especially true when the product's life cycle (and thus total number of sales) is limited.

However, it is no secret that many companies struggle with managing projects especially in the area of new product development. Given the vital nature of new product development to both the short-term health and long-term viability and growth of a company, it begs the question: *Why do companies with a recognized history of bringing new product development projects in late, over budget and with quality below expectations continue to do so?*

In this research, we set out to show that the best intentions of implementing a strategy of concurrent engineering in new product development can lead to difficult challenges. In particular, the scheduling of integration events that are used to identify errors in design that affect one or more coupled components can be difficult at best. Indeed, poor scheduling and an over-reliance on these integration build events can result in scheduling too many of these integration events which can lead to a degradation in project performance both in terms of quality and cost. This problem is exacerbated by the fact that often times coupled components often iterate through the design-build-test iteration cycles at different speeds. This results in components that are learning at different rates and can often lead to unnecessary rework.

The integration challenge that results from implementing concurrent engineering and the potential for creating unnecessary or phantom work naturally leads to the question facing companies, research and development organizations, and new product development project managers in particular: ***How often should we execute a design-build-test cycle in order to minimize the new product development time and cost while attaining a desired level of quality?*** In particular, how many build events should be executed given a fixed deadline (i.e. product launch date) and disparate design iteration cycle times between coupled subsystems that are developed concurrently?

In this research a stylized system dynamics simulation model was developed to explicate the challenges of concurrent engineering and to answer these and related questions. The new product development (NPD) model was based on extensive field research in the engineering organization of a mid-sized US company in the automotive industry.

The resulting model was validated and verified by using experts from within the Company who were both intimately involved in the new product development methodology in the Company and

knowledgeable about system dynamics modeling. Additionally, a historical NPD project (Project ATOMac) from within the Company was used to validate the NPD model. A detailed analysis of the model's behavior was conducted including a series of policy analyses using Project ATOMac as a point of comparison.

The analysis conducted here using the NPD Model and Project ATOMac as a case study suggested a number of findings. First, the NPD model is particularly sensitive to a few of the model parameters. These include: base design and redesign quality (both component and integration), base design and redesign productivity, the number of initial designers assigned to the project, the relative speed (*Speed Factor*) at which the two components iterated through the design-build-test learning cycle, and the level of complexity of the project in terms of the number of coupled components (*Fraction of Coupled Designs*) and the probability of a change in one component requiring a change in a coupled component (*External Rework Probability*).

In terms of project performance as predicted by the NPD model, having more build events does not result in improved project performance either in terms of the quality attained or in the cost of executing the project schedule. In fact, the model suggests that executing more builds will result in a decrease in the quality achieved and an increase in the project costs. While this finding was suspected by the Company, the NPD Model is able to demonstrate it clearly.

Additionally, while the scheduling (i.e. spacing) of the build events affects project performance there is no clear best policy. There seems to be some benefit gained from delaying the initial prototype build in that it allows designers to work at a reasonable pace and not induce errors due to cutting corners. However, delaying the initial build too long seems problematic from an organizational view point in that it means that the organization has to wait longer in order to get a "reliable" status of the project. Secondly, delaying the initial build too long leaves less time for

rework for those designs found to be flawed and can cause an increase in the schedule pressure felt by designers and thus cause them to work faster and perhaps out of synch with their coupled components inducing errors and phantom rework. This is particularly troublesome in that we found through our analysis that the degree to which the components work out of synch definitely impacts on project performance.

Lastly, a more aggressive labor policy seems to be a worthwhile endeavor. While adding designers to the project sooner results in an increased labor cost, the resulting quality improvements seem to justify the additional cost. When combined with a policy of fewer build events, the additional cost of aggressively adding labor is seemingly subsumed by the cost reductions achieved by holding fewer build events.

Finally, the NPD model is able to capture the phenomenon of phantom rework. It clearly shows the generation of additional work as a result of components operating out of synch (at different speeds) with each other. The amount of this potentially unnecessary work generated in a project is exacerbated by the degree to which components are coupled, the relative imbalance in their design-build-test iteration speeds, and the schedule pressure induced by an increased number of builds.

Contributions

I believe there are three main contributions of this research which can be roughly classified as conceptual, modeling, and practical. Each will be discussed in some detail below.

From a conceptual standpoint, this research introduces the concept of phantom work. Phantom work as described and modeled here is the unnecessary work (and rework) that is done as a result of coupled components being developed in parallel are designed, built and tested at different speeds in an effort to implement concurrent engineering in the detailed design stage. The amount

of phantom work generated was shown to be a function of the individual iteration speeds of the various subsystems, the speed imbalance between them, and the timing of the iteration cycles. There were two sources of this phantom work identified. The first source was unnecessary rework that is created when some components are ready for a build event and others are not. The second source of phantom work is the redesign of components after their design has been released for prototyping and before the next set of test results is available. As discussed, this unnecessary work can be treated as a cost of overlapping design-build-test cycles and should be considered by managers when setting the timing of integration (or build) events.

"Phantom" work presents a "real" problem for companies engaging in new product development. Clearly, doing unnecessary work has a direct cost to the organization in that engineers' time is being wasted and other resources (e.g. design, prototyping and test equipment) may be unnecessarily consumed.

From a modeling standpoint, this research is the first public work to model simultaneous, overlapped design and build events. While much of the model uses standard formulations (e.g. the labor sector), the model structure that accounts for the paper designs, the physical parts, and their related component and integration errors is a novel contribution to the system dynamics modeling community. The accounting of multiple and simultaneous builds while tracking parallel paper designs has not been modeled and documented publicly to the best of my knowledge. The model is certainly limited in a number of ways. First, it only accounts for two components each consisting of 1,000 designs. Each design is linked to at most one other design from the coupled component and thus is limited to have only one integration error at a time. Similarly, each design is limited to have no more than one component error at a time. These are clearly simplifications of the real world where there are many components each with individual parts (and thus designs) that

can be coupled with many other parts from multiple coupled components and thus have multiple integration errors. As a result, changes in one design can affect multiple other designs. Additionally, each part (or design) can have multiple component errors in and of themselves. However, despite these limitations the NPD model developed here provides the basic structure to scale the model to accommodate these additional complexities. This additional complexity was unnecessary to answer the questions posed here and in particular to demonstrate the concept of phantom work. Indeed, increasing the detail of the model to account for more complicated component design interactions would further demonstrate the concept of phantom work but at the expense of making the model more difficult to explicate simply.

Lastly, the practical management implications of this research are straightforward. When it comes to integration events (i.e. builds) more is not necessarily better and in many cases can result in worse project performance. Managers should avoid the temptation to build more frequently. In fact, one "real" prototype build can effectively replace multiple prototype builds that seem to get diluted and thus somewhat ignored by designers. Similarly, managers should avoid the temptation to build too soon. Not only can building too soon cause an increase in schedule pressure and thus increase the error rate, but building too soon can create phantom work as component designers begin to work at different speeds. By extension, building too soon can increase the pressure to build again as the number of errors seemingly grows while the deadline approaches. Rather, delaying the initial build can lead to a decrease in errors and avoidance of phantom work. Lastly, aggressively responding to a demand for additional designers by adding resources early in the project can lead to better project performance in terms of quality. Combined with a decrease in the number of builds, an aggressive labor policy can lead to an improvement in project performance in terms of both quality and cost.

Future Work

There are a number of opportunities for future work as a result of this research. With regard to the limitations of the NPD model, additional modeling effort could be undertaken to extend the ability of the model to handle cases of multiple components and multiple component interactions thus resulting in multiple integration errors per design. Similarly, we could extend the model to allow for multiple component type errors per designed part. Additionally, the NPD model assumes perfect testing in vehicle testing this is an assumption that could be relaxed and modeled with some additional effort although, on the surface, the effect of such work doesn't appear to change the fundamental findings and conclusions here.

Additionally, from a modeling perspective the NPD model does not account for learning through design iteration. For example, a designer that has iterated the design of the same part multiple times is no more efficient (i.e. productive) the third or fourth time than he or she was the first time. Modeling this type of learning is not trivial in the present construct of the NPD model. Again, it is my belief that including this additional model complexity will not necessarily change the fundamental findings and conclusions presented here.

From a new product development perspective, there are many other opportunities for a project manager to affect the integration of coupled components that are not considered here. For example, the increased capabilities and use of 3-dimensional computer-aided design (3-D CAD) have greatly improved the ability to identify fitment problems between components without having to physically prototype and test the geometry of the components. Including the ability to do more of these types of virtual builds and thus shorten the delay associated with prototyping and testing

may lead to different conclusions. Certainly one can imagine that increasing the frequency of these types of low cost builds may prove beneficial.

Conclusions

Concurrent engineering is a popular approach employed by companies competing in an increasingly competitive, often global market place in an effort to get new products to the market faster. The challenges associated with concurrent engineering are many. The research here attempts to tackle the particular challenge of scheduling integration (i.e. build events) as part of the detailed design phase of new product development. The concept of unnecessary or phantom work is introduced and modeled as a feature of poorly timed integration events that are used to test coupled components that are developed in parallel while operating at different speeds. Various policies aimed at improving new product development project performance both in terms of quality and cost are examined and practical implications for NPD project managers are identified. At the end of the day, NPD project managers must understand that build events are often a necessary evil in order to ensure effective integration of coupled components. However, despite the many positive outcomes (i.e. learnings) that come from these integration events, scheduling more of them is not necessarily a good thing. The challenge for the NPD project manager is to suppress the organizational desire for project status information (i.e. integration test results) and avoid the temptation to schedule more build events. Perhaps given more time between integration events, individual designers will come to rely less on vehicle builds for component and integration testing and instead engage in the necessary coordination with the designers of those parts that are coupled with theirs and thus reduce the need for integration events in the first place.

BIBLIOGRAPHY

AitSahlia, F., E. Johnson, et al. (1995). "Is Concurrent Engineering Always a Sensible Proposition?" IEEE Transactions on Engineering Management **42**(2): 166-170.

Allen, T. J. (1977). Managing the Flow of Technology: Technology Transfer and the Dissemination of Technological Information within the R&D Organization. Cambridge, MA, MIT Press.

Black, L. J. (2002). Collaborating Across Boundaries: Theoretical, Empirical, and Simulated Explorations. Sloan School of Management. Cambridge, MA, MIT. **PhD:** 282.

Black, L. J. and N. P. Repenning (2001). "Why Firefighting is Never Enough: Preserving High-Quality Product Development." System Dynamics Review **17**(1).

Blackburn, J. (1991). Time-Based Competition: The Next Battleground in American Manufacturing. Homewood, IL, Business One Irwin.

Blackburn, J. D., G. Hoedemaker, et al. (1996). "Concurrent Software Engineering: Prospects and Pitfalls." IEEE Transactions on Engineering Management **43**(2): 179-188.

Bussey, J. and D. R. Sease (1988). Manufacturers Strive to Slice Time Needed to Develop Products. The Wall Street Journal**:** 1.

Clark, K. B. and S. C. Wheelwright (1997). Organizing and Leading 'Heavyweight' Development Teams. Managing Strategic Innovation and Change. M. L. Tushman and P. Anderson. New York, Oxford University Press**:** 419-432.

Cooper, K. G. (1980). "Naval Ship Production: A Claim Settled and a Framework Built." Interfaces **10**(6): 20-36.

Cooper, R. G. (2001). Winning at New Products: Accelerating the Process from Idea to Launch. New York, Basic Books.

Covey, S. R., A. R. Merrill, et al. (1994). First Things First: To Live, to Love, to Learn, to Leave a Legacy. New York, Fireside.

Droge, C., J. Jayaram, et al. (2000). "The ability to minimize the timing of new product development and introduction: an examination of antecedent factors in the North American automobile supplier industry." Journal of Product Innovation Management **17**(1): 24-40.

Eppinger, S. (2001). "Innovation at the Speed of Information." Harvard Business Review(January 2001): 149-158.

Eppinger, S. D., M. V. Nukula, et al. (1997). "Generalised Models of Design Iteration Using Signal Flow Graphs." Research in Engineering Design **9**(2): 112-123.

Eppinger, S. D., D. E. Whitney, et al. (1994). "A Model-Based Method for Organizing Tasks in Product Development." Research in Engineering Design **6**(1): 1-13.

Fine, C. H. (1998). Clockspeed: Winning Industry Control in the Age of Temporary Advantage. Reading, MA, Perseus.

Ford, A. (1995). The Dynamics of Project Management: An Investigation of the Impacts of Project Process and Coordination on Performance. Sloan School of Management. Cambridge, MA, Massachusetts Institute of Technology. **PhD:** 341.

Ford, D. N. and J. Sterman (2003a). "Overcoming the 90% Syndrome: Iteration Management in Concurrent Development Projects." Concurrent Engineering Research and Applications **11**(3): 177-186.

Ford, D. N. and J. Sterman (2003b). "The Liar's Club: Impacts of Concealment in Concurrent Development Projects." Concurrent Engineering Research and Applications **11**(3): 211-219.

Ford, D. N. and J. D. Sterman (1998). "Dynamic modeling of product development processes." System Dynamics Review **14**(1): 31-68.

Forrester, J. W. (1961). Industrial Dynamics. Cambridge, MA, Productivity Press.

Gold, B. (1987). "Approaches to accelerating product and process development." Journal of Product Innovation Management **4**(2): 81-88.

Griffin, A. (1997). "PDMA Research on New Product Development Practices: Updating Trends and Benchmarking Best Practices." Journal of Product Innovation Management **14**(6): 429-458.

Ha, A. Y. and E. L. Porteous (1995). "Optimal Timing of Reviews in Concurrent Design for Manufacturability." Management Science **41**(9): 1431-1447.

Haddad, C. J. (1996). "Operationalizing the Concept of Concurrent Engineering: A Case Study from the U.S. Auto Industry." IEEE Transactions on Engineering Management **43**(2): 124-132.

Katz, R. and T. Allen (1985). "Project Performance and the Locus of Influence in the R&D Matrix." Academy of Management Journal **28**(1): 67-87.

Kingdon, O. (1973). Matrix Organization: Managing Information Technologies. London, Tavistock.

Krishnan, V., S. D. Eppinger, et al. (1997). "A Model-Based Framework to Overlap Product Development Activities." Management Science **43**(4): 437-451.

Langerak, F. and E. Jan Hultink (2005). "The Impact of NPD Acceleration Approaches on Speed and Profitability - Lessons for Pioneers and Fast Followers." IEEE Transactions on Engineering Management **52**(1): 30-42.

Loch, C. H. and C. Terwiesch (1998). "Communication and Uncertainty in Concurrent Engineering." Management Science **44**(8): 1032-1048.

Marquis, D. (1969). "Ways of Organizing Projects." Innovation **5**(7): 25-33.

Marquis, D. and D. Straight (1965). Organizational Factors in Project Performance, MIT Sloan School of Management.

McGrath, M. E., Ed. (1996). Setting the PACE in Product Development: A Guide to Product and Cycle-Time Excellence. Boston, MA, Butterworth-Heinemann.

Millson, M. R., S. P. Raj, et al. (1992). "Survey of Major Approaches for Accelerating New Product Development." Journal of Product Innovation Management **9**(1): 53-69.

Repenning, N. P. (2001). "Understanding fire fighting in new product development." The Journal of Product Innovation Management **18**: 285-300.

Repenning, N. P., P. Goncalves, et al. (2001). "Past the Tipping Point: The Persistence of Firefighting in New Product Development." California Management Review **43**(4): 44-63.

Rosenblatt, A. and G. F. Watson (1991). "Special Report: Concurrent Engineering." IEEE Spectrum **28**(7): 22-44.

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

Smith, R. P. (1992). Development and Verification of Engineering Design Iteration Models. Cambridge, MA, MIT. **PhD**.

Smith, R. P. and S. D. Eppinger (1997a). "Identifying Controlling Features of Engineering Design Iteration." Management Science **43**(3): 276-293.

Smith, R. P. and S. D. Eppinger (1997b). "A Predictive Model of Sequential Iteration in Engineering Design." Management Science **43**(8): 1104-1120.

Stalk, G. (1988). "Time - The Next Source of Competitive Advantage." Harvard Business Review **66**(4): 41-51.

Sterman, J. D. (2000). Business Dynamics: Systems Thinking and Modeling for a Complex World. Chicago, Irwin-McGraw Hill.

Symonds, W. (1991). Pushing Design to Dizzying Speeds. Business Week**:** 64-68.

Takeuchi, H. and I. Nonaka (1986). "The New New Product Development Game: Stop Running the Relay Race and Take up Rugby." Harvard Business Review(January-February): 137-146.

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

Whiting, R. (1991). "Product Development as a Process: Electronic Industry Executives are Learning that Product Development is as Important as Quality of Manufacturing Prowess." Electronic Business **30**.

INDEX

APPENDIX A: MODEL DOCUMENTATION

This appendix provides full documentation of the simulation model used in this research. It is intended to provide the reader with a more detailed description of the various sectors in the model introduced in Chapter 4. The detailed description provided here along with the full listing of the simulation model's equations which can be found in Appendix B should provide the reader with the necessary information to replicate the simulation model and the associated analysis.

MODEL OVERVIEW

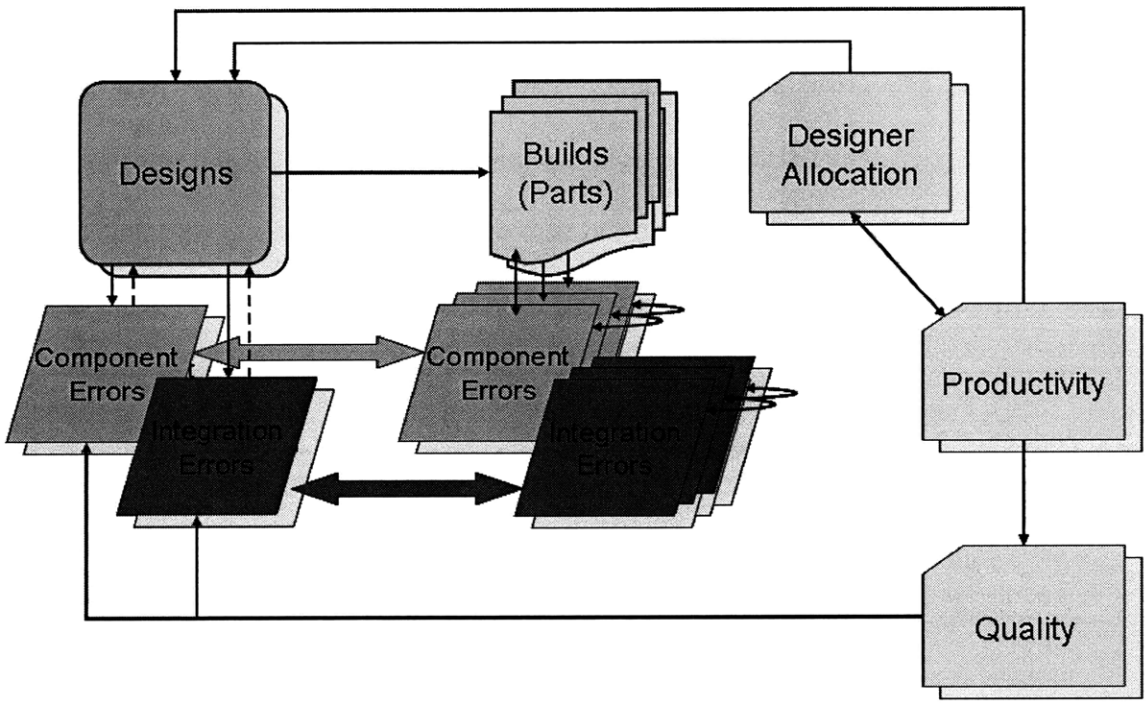


Figure 95 High Level Model Structure

At a very high level, the NPD model can be viewed as a set of paper designs that are constantly evolving and a set (or sets) of parts that correspond to the state of the paper designs at the time they were prototyped for a build event. Along with the set of paper designs are a corresponding set of component errors and integration errors. These errors are created and fixed as part of the ongoing design and redesign of the paper designs. Similarly, there are sets of component and integration errors that correspond to the set of build parts. These errors reflect the errors that existed in the paper designs at the time the parts were prototyped. As component and integration errors are discovered in parts through vehicle testing their corresponding paper designs are sent for rework. Because it is possible to overlap the design-build and test cycles, there can be multiple versions of build parts undergoing vehicle testing. This means there can be multiple versions of component and integration errors associated with those parts. Some of these errors are common across versions of the build parts. In addition to the two main sectors (designs and parts with their corresponding errors) of the NPD Model, there are three other significant sectors which control the flow of designs, parts and errors in the model. The first is the Productivity sector which determines the productivity rates for the design engineers in conducting design and redesign work accounting for the schedule pressure inherent in the new product development environment. The Quality sector is used to model the quality of design and redesign work done by the design engineers as they respond to schedule pressure. This sector obviously affects the rate at which errors are created and fixed in the evolution of paper designs. The Designer Allocation sector models the process by which designers allocate their time between working on design, redesign and coordination tasks. Given that the NPD model simulates two coupled components being designed in parallel, these main sectors as well as the other sectors of the model are replicated for each of the components.

In total there are 13 sectors of the model (listed here) which will be described in detail below:

New Product Development Model Sectors

1. Designs
2. Builds
3. Vehicle Testing
4. Labor
5. Designer Allocation
6. Productivity
7. Quality
8. Bench Test Fraction
9. Coordination Fraction
10. External Rework
11. Phantom Work
12. Build Switches
13. Speed Factor

Each sector of the NPD model is designed to capture a specific physical process or decision rule that is found in the development process at the client company. In this chapter, a section is provided for each of these sectors that details the model structure and provides the rationale for its use. Each section also includes a listing and description of the model equations as formulated in the model. In some cases, the model structure and the formulation of specific processes cross between sectors in the model. In those cases, the detailed description of the model structure will be

included in the section of the chapter that is most relevant. However, in the other sections of the chapter where the model structure and/or variable is used enough information will be included to provide context and understanding to the reader.

Technical documentation of the model is included in Appendix B. It includes a full equation listing that provides a brief description of each model variable, the units of measure, a cross-referenced listing of input variables for a given variable, as well as a listing of the variables for which it serves as an input variable.

In order to make the model documentation provided in this chapter more readable, *italics* are used in the text when specifically referring to a model variable. Additionally, references to functions used in the model equations that are available in the VENSIM simulation software are written in ALL CAPS.

DESIGN SECTOR

Design Sector Overview

The design sector of the NPD Model is structured to account for the current status of the set of designs for a new product development project during its detailed design phase. This sector includes the set of designs (paper drawings, CAD drawings, or prototyped parts) corresponding to the individual parts that make up each component of the new product under development. These designs reflect the latest version (revolution) of the design for each given part. The design sector also accounts for the component errors and the integration errors associated with these designs. Figure 96 provides a high level depiction of the design sector of the NPD Model.

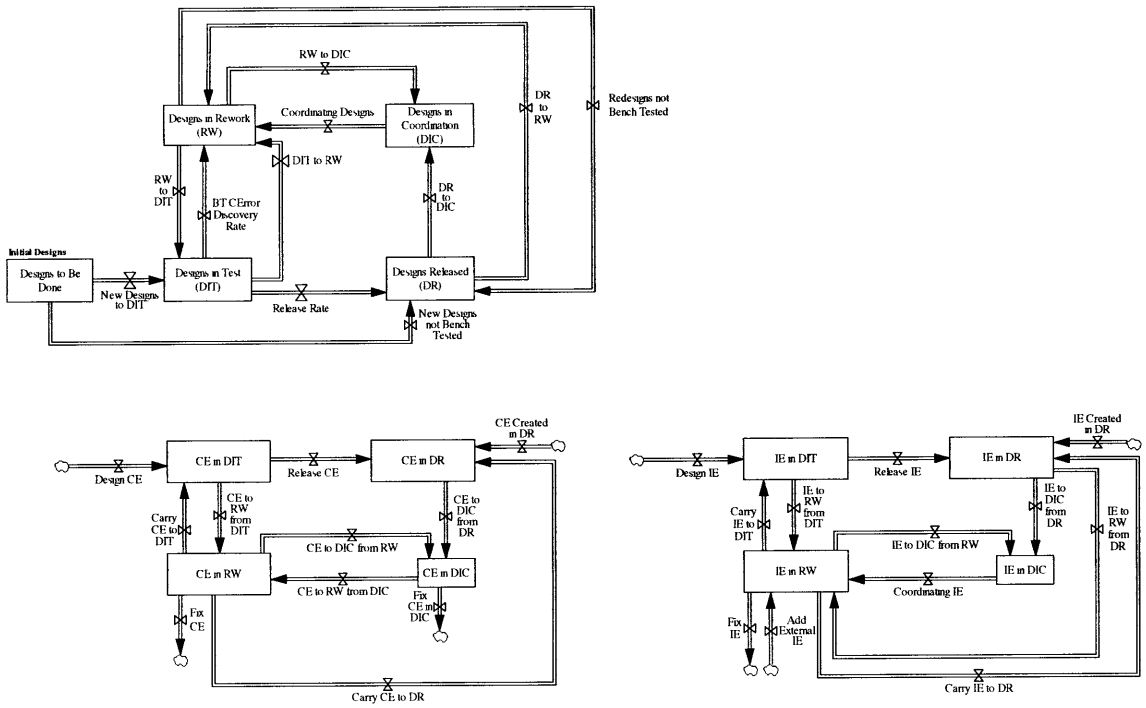


Figure 96 Design Sector

The basic structure of the design sector consists of three stock and flow chains. The main stock and flow chain accounts for the set of designs as they are developed during the detailed design phase of the new product development project. This stock and flow chain is the in the upper left of Figure 96. There are 5 stocks used in the main stock and flow chain: *Designs to Be Done, Designs in Test, Designs in Rework, Designs in Coordination* and *Designs Released.* These stocks are depicted by boxes with the variable names inside of the box and are used to represent the different states that a given design can be in at any given time. For example: a design that has not yet been worked on would reside in the stock of *Designs to Be Done.* As the designs are worked on and evolve over time, they can transition between the different states. For example, a design from the stock of *Designs to Be Done* that has been completed can be sent for bench testing and would transition or flow into the stock of *Designs in Test.* The flow of designs from one stock to another

is depicted by the arrows that connect the boxes with the flow assumed to be in the direction of the arrow. These arrows can be thought of as pipes with control valves. The variable names that control the rates of flow can be found next to the hourglass shaped symbol on each of the arrows.

Co-flows are stock and flow chains that correspond to a main stock and flow chain in a system dynamics model. Co-flows are used to account for properties of the items in the main stock and flow chain. In the case of the NPD Model, two co-flows are used in the design sector to account for the errors, both component and integration, associated with the designs in the main chain. These stock and flow chains are nearly identical to each other and are depicted at the bottom of Figure 96. With the exception of the stock of *Designs to Be Done*, there is a corresponding stock of errors for each of the stocks of designs in the main stock and flow chain. Because there are no errors in a design that has not yet been worked on, there is no need to maintain a corresponding stock of errors for the stock of *Designs to Be Done*. Additionally, there is a flow between each of the stocks in the co-flow structure of the component errors (CE) and integration errors (IE) corresponding to the flows between the stocks in the main stock and flow chain of designs.

A more detailed description of the Design Sector (and each of the other sectors of the NPD Model) follows. A complete listing of the variables and associated equations in the model is included in Appendix B. The appendix includes a brief description of each variable and a cross-referenced list of those variables that serve as an input to the equation for a given variable or use the given variable as an input to their own.

The NPD Model is developed for a new product that has two main components – component A and component B. As such, there are actually two identical sets of stock and flow chains used in the design sector and indeed throughout much of the NPD Model. The simulation software, VENSIM, handles multiple instances of stocks, flows and auxiliary variables using subscripting.

The use of subscripts is indicated in a given equation by the inclusion of a subscript name or a particular instance of the subscript enclosed in brackets. For example: In the NPD model, the differential equation that describes the changing state of the stock of designs to be done is written as follows:

Designs to Be Done[Component] = INTEG(- New Designs to DIT[Component] - New Designs not Bench Tested[Component] , Initial Designs[Component])

(2)

In the VENSIM simulation language, equation (2) is a differential equation that can be read to mean that the state of the stock of *Designs to Be Done[Component]* at any given point in time is the integral of the outflows *New Designs to DIT[Component]* and New *Designs not Bench Tested[Component]* given an initial value of *Designs to Be Done[Component]* equal to *Initial Designs[Component]*. The use of *[Component]* after a given variable in the equation indicates that the variable is subscripted for the set of components, in this case components A and B.

On the other hand, consider the model equations for external rework:

External RW from IE[a] = Coordinating Designs[b] * External RW Probability

(3)

External RW from IE[b] = Coordinating Designs[a] * External RW Probability

(4)

In this case, the use of subscripts in the equations actually invokes a particular component. For example, the variable *External RW from IE[a]* refers to the external rework generated from integration errors for component A and it is based on the rate of *Coordinating Designs* for

component B. You'll note that there is a second equation for *External RW from IE[b]* which is indeed different from the equation for component A because it uses the rate of coordinating designs for component A[13]. The fact that the variable *External RW Probability* does not have a subscript in these equations indicates that the variable is the same for both components in the model.

The use of subscripts is prevalent throughout the NPD model. There are two main uses for subscripting. The first, as mentioned, is to distinguish between the two components simulated in the model. Indeed, almost all of the variables in the model are subscripted for the different components – A and B. This is certainly true for the stocks and flows of designs as well as their related build parts which will be discussed in the Build Sector section of this chapter. In most cases the auxiliary variables are also subscripted to account for differences between the two components. There are some cases (e.g. *External RW Probability* mentioned above) where the variable will be the same for both components and thus is not subscripted. The second use of subscripts in the model is to distinguish between multiple vehicle builds. The stocks and flows of the build parts are subscripted for this purpose. In this case, these variables are subscripted for both the components and the builds. An example of a variable that is thusly subscripted is the stock of *Build Parts [Component, Build]* where the use of the *[Component, Build]* indicates multiple subscripts.

For ease of exposition, except where necessary subscripts will not be included in the description of the NPD Model nor in the equations provided as part of the text. As mentioned earlier, technical documentation of the model is included as an appendix to this report. It includes

[13] Note: For ease of reading this report, the two different components (A and B) used in the simulation model will be referred to in the text with capital letters. However, in those equations and variables where a specific component is referred to, the subscript will appear in lower case.

all of the variables and equations in the model with full subscripting included. Where it is not obvious in the discussion, the reader should refer to this appendix to verify the use of subscripting.

Designs

The main stock and flow chain in the Design Sector of the NPD Model accounts for the set of designs as they are developed during the detailed design phase of a new product development project. Figure 97 depicts the main stock and flow chain for a set of designs for a single component or subsystem. As the model includes two components, the stocks and flows are subscripted to account for the set of designs for each component. The designs are tracked through the various stages of development as they are initially worked on by designers, as they are tested either through bench testing or vehicle testing, as they are released awaiting a build event, as they are coordinated when found to have an integration error, and as they are redesigned in an effort to fix known errors.

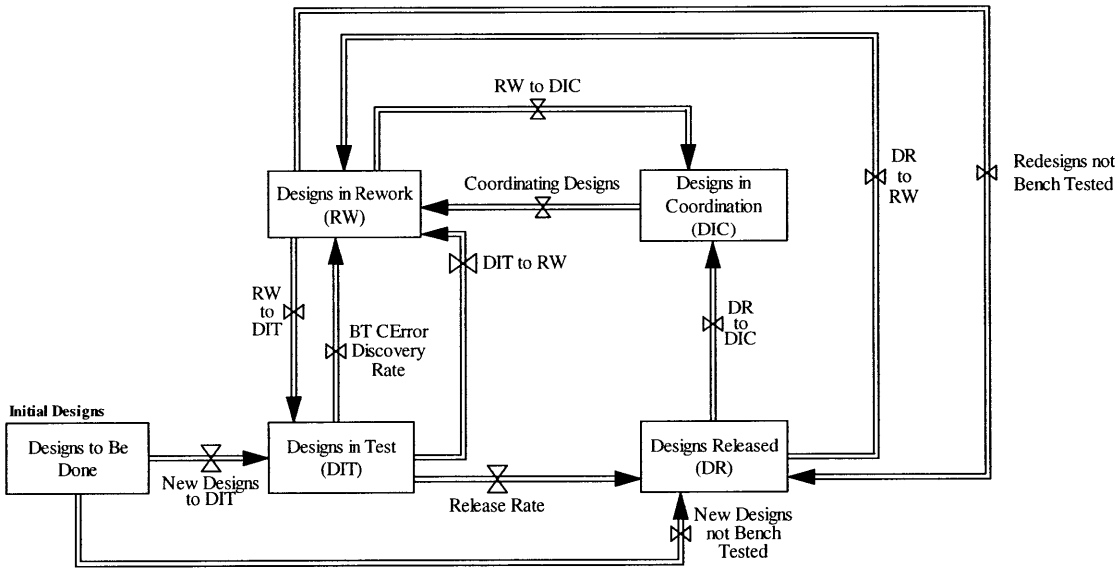


Figure 97 Design Stock and Flow Diagram

Consider a typical component (or sub-system) such as the frame of a motorcycle in the detailed design phase of a new product development project. There is an initial set of designs that must be completed for the motorcycle frame as part of the project. These required designs all begin in the stock of *Designs to Be Done.* As new designs are completed, they can either be sent for bench testing (represented by the flow *New Designs to DIT* in Figure 97) or they can be released without bench testing (represented by the flow *New Designs not Bench Tested*) awaiting the full vehicle build event and subsequent vehicle testing. The stock *Designs in Test* represents those designs that are in the queue for bench testing. As a result of bench testing, a given design might be found to contain a component error and is thus sent to the stock of *Designs in Rework.* The rate at which these component errors are discovered is represented by the flow *BT CError Discovery Rate.* Those designs that are bench tested where no component error is discovered are sent to the stock of *Designs Released* to await the next build event. This is depicted as the flow *Release Rate* in Figure 97. Those designs in the stock of *Designs in Rework* are in the queue awaiting redesign. Just as in the case of new designs, once a redesign is complete the design can either be sent for bench testing (represented by the flow *RW to DIT*) or they can be released without bench testing (represented by the flow *Redesigns not Bench Tested*) to await the next build event.

During a build event, all of the designs that are in progress are physically built as parts and the full vehicle, including all components, is assembled[14]. Once the full vehicles are assembled they are sent for vehicle testing where both component errors and integration errors can be discovered. The model structure for vehicle builds and vehicle testing will be described in greater detail in the Build Sector and Vehicle Testing sections of this chapter.

[14] Designs in progress are those designs that, at a minimum, have a completed initial design. This includes those designs in the stocks of Designs in Test, Designs in Rework, Designs Released, and Designs in Coordination.

As a result of vehicle testing, designs will be found to have either component errors, integrations errors, or both. Depending on the type of error(s) discovered, the designs with errors will be sent either directly to the stock of *Designs in Rework* or to the stock of *Designs in Coordination*. The stock of *Designs in Coordination* is the queue for those designs that have been found to have an integration error and require coordination between designers from the two components involved in the integration error before it can be redesigned and the integration error fixed. As designs are coordinated they are sent to the stock of *Designs in Rework* to await redesign. This process is represented by the flow *Coordinating Designs* in Figure 97.

While those designs found to have only a component error will be sent directly for redesign (*Designs in Rework*) and those found to have only an integration error will be sent for coordination (*Designs in Coordination*), those designs that have both types of errors may be sent for coordination or directly for redesign. The decision about where designs with both types of errors will be sent upon discovery will be discussed in detail below. As an example, in Figure 97 the flow *DR to DIC* from *Designs Released* to *Designs in Coordination* represents the rate at which designs released are sent for coordination as they are discovered to have an integration error in vehicle testing.

You should note that even designs in the queue for redesign (*Designs in Rework*) may be sent for coordination (*RW to DIC*). This is possible because at the time of a vehicle build all of the designs are built and assembled as part of the vehicle – regardless of their current state. This means that in addition to the stock of *Designs Released*, the designs that are in the queue for bench testing (*Designs in Test*), in the queue for coordination (*Designs in Coordination*), and those in the queue for redesign (*Designs in Rework*) are built and subsequently vehicle tested. As a result, a design in the queue for rework with a component error, perhaps discovered in bench testing, may

also contain an integration error that is subsequently found in vehicle testing. As a result of the discovery of the integration error, the design may be sent for coordination prior to being redesigned (represented by the flow *RW to DIC*) or it may remain in the stock of *Designs in Rework* to be redesigned without coordination.

Given this brief overview of the main stock and flow chain of designs, we will now consider each of the stocks of designs and their associated inflows and outflows in greater detail. There is a subsection included for each of the 5 main stocks: *Designs to Be Done*, *Designs in Test*, *Designs in Rework*, *Designs Released* and *Designs in Coordination.*

Designs to Be Done

The first set of designs in the NPD model to be considered is the stock of *Designs to Be Done.* This stock and its associated flows can be seen in Figure 98.

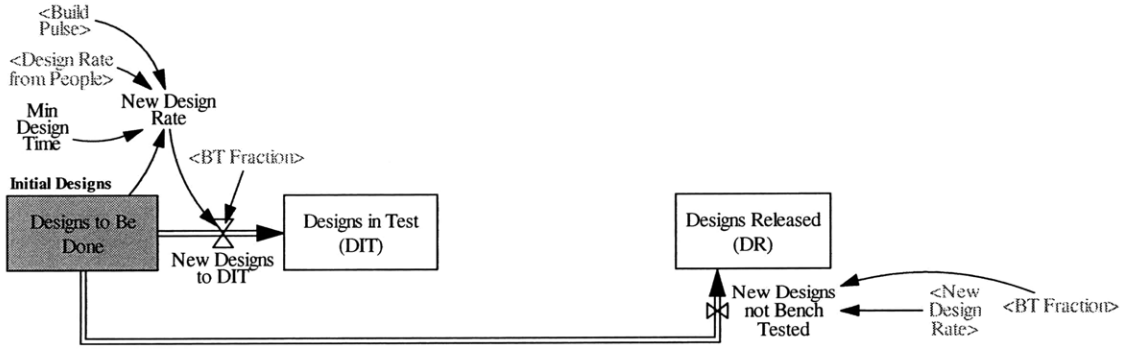


Figure 98 Designs to Be Done

This stock represents the set of paper designs that still need to be completed at any point in time given an initial set of required designs and the amount of new designs that have been completed to date. The equation for this stock is as follows:

Designs to Be Done = INTEG(- New Designs to DIT - New Designs not Bench Tested , Initial Designs)

(5)

The stock of *Designs to Be Done* for each component has an initial value based on the variable *Initial Designs*, which is set to 1000 designs for both components A and B. It also has two outflows: *New Designs to DIT* and *New Designs not Bench Tested.* The equations for these variables are included below:

Initial Designs = 1000

(6)

New Designs to DIT = New Design Rate * BT Fraction

(7)

New Designs not Bench Tested = New Design Rate * (1 - BT Fraction)

(8)

Both of the outflows from the stock of *Designs to Be Done* are based on the rate at which new designs are completed (*New Design Rate*). The variable *BT Fraction* determines the fraction of new designs that are sent for bench testing upon completion with the complement (*1-BT Fraction*) of new designs being released to await the next build event without bench testing. The fraction of

designs that are sent for bench testing is based in large part on the amount of time remaining until the next build event with the fraction decreasing as the amount of time decreases. While the formulation for *BT Fraction* will be discussed in greater detail later in this chapter, the equation for *New Design Rate* and a discussion of its formulation follows:

New Design Rate = (1 - Build Pulse) * min (Design Rate from People , Designs to Be Done / Min Design Time)

(9)

The new design rate for each component is set at the minimum of the design rate that is possible based on constrained resources and the productivity of designers (*Design Rate from People*) and the design rate that is possible based on the amount of work available (*Designs to Be Done*) and the minimum time it takes to complete a design (*Min Design Time*). The minimum design time is set at one week for both components and represents the minimum time required to complete a design without any resource constraints. The variable *Design Rate from People* will be discussed in detail in the Productivity Sector section of this chapter.

As a means for shutting off the flow of new designs at the instant of a build event, the term (*1-Build Pulse*) is multiplied by the minimum value calculated in the equation for *New Design Rate* above. The variable *Build Pulse* is valued at 0 except at the instant in time of each build event[15]. While shutting down this flow at each build event allows the simulation model to handle the accounting of transferring paper designs into build parts, it also reflects the reality of designers ceasing all other work in order to actively participate in and/or monitor the build event.

[15] This same method of turning off the rate of new design completion at the instant of a build event is used to turn off other rates of flow in the NEW PRODUCT DEVELOPMENT model including bench testing, redesigning, coordinating designs, working ahead and vehicle testing.

Designs in Test

The next set of designs in the main stock and flow chain in the Design Sector NPD model to be considered is the stock of *Designs in Test.* This stock and its associated inflows and outflows are depicted in Figure 99.

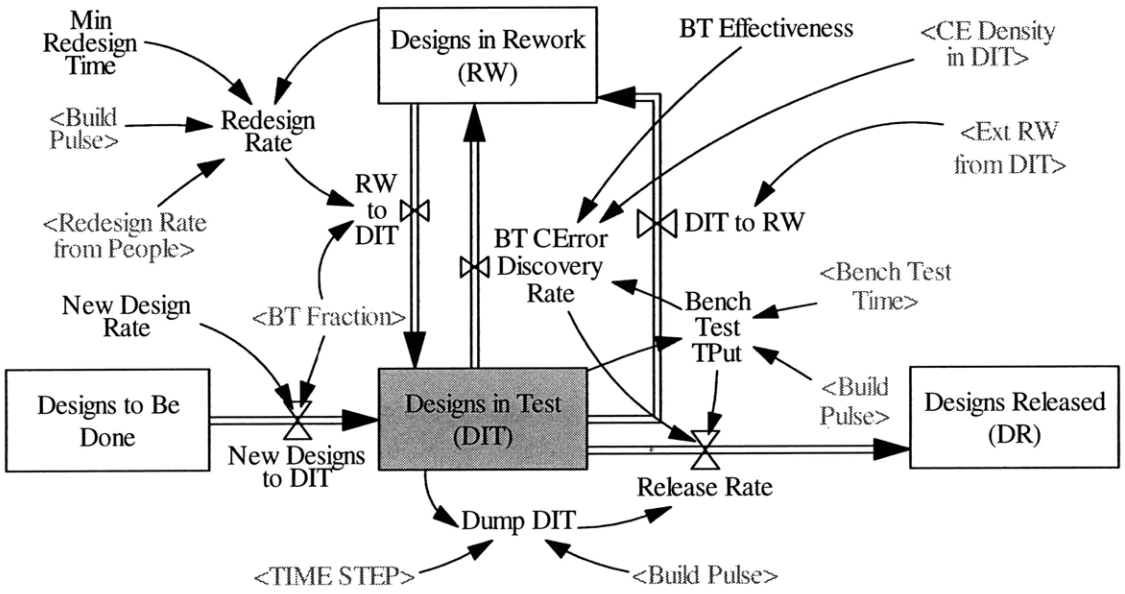


Figure 99 Designs in Test (DIT)

The stock of Designs in Test (DIT) represents those designs that are in the queue for bench testing. The model equation for this stock is as follows:

Designs in Test (DIT) = INTEG(New Designs to DIT + RW to DIT - BT CError Discovery Rate - Release Rate - DIT to RW , 0)

(10)

The stock of *Designs in Test* begins with an initial value of zero[16] and integrates two inflows (*New Designs to DIT* and *RW to DIT*) as well as three outflows (*BT CError Discovery Rate*,

[16] With the exception of *Designs to Be Done* and *New Designers* (in the Labor Sector), all stocks in the NEW PRODUCT DEVELOPMENT model begin the simulation with an initial value of zero.

Release Rate, and *DIT to RW*). The equation for the inflow of *New Designs to DIT* was discussed as an outflow of the stock of *Designs to Be Done* and therefore won't be repeated here. The equations for the other flows follow:

RW to DIT = Redesign Rate * BT Fraction

(11)

BT CError Discovery Rate = CE Density in DIT * Bench Test TPut * BT Effectiveness

(12)

Release Rate = Bench Test TPut - BT CError Discovery Rate + Dump DIT

(13)

DIT to RW = Ext RW from DIT

(14)

As discussed earlier, the inflow *New Designs to DIT* represents the rate at which new designs are completed and sent for bench testing. In the same manner, the inflow *RW to DIT* represents the rate at which completed redesigns from the stock of *Designs in Rework (RW)* are sent for bench testing as determined by the redesign rate and the bench test fraction. Just as in the case of the new design rate, the variable *Redesign Rate* is set at the minimum of the redesign rate that is possible based on constrained resources and designer productivity (*Redesign Rate from People*) and the redesign rate that is possible based on the amount of work available (*Designs in Rework*) and the minimum time it takes to complete a design (*Min Redesign Time*). The minimum redesign time is set at one week for both components and again represents the minimum time required to complete

a redesign without any resource constraints. The variable *Redesign Rate from People* will be discussed in detail in the Productivity Sector section of this chapter.

Redesign Rate = (1 - Build Pulse) * MIN (Redesign Rate from People , Designs in Rework (RW) / Min Redesign Time)

(15)

The two main outflows from the stock of *Designs in Test* are *BT CError Discovery Rate* and *Release Rate*. Both of these flows are the result of bench testing and represent the designs in which a component error is discovered in bench testing and sent for rework and those designs that are released because no error is found. The primary driver of these flows is the variable *Bench Test TPut* which represents the throughput rate at which designs are bench tested. The throughput is modeled as a simple first-order delay using an average bench test time which may be different for the two components. The difference in the average bench test times is one of the sources for the different design iteration speeds between components. The other differences that account for the potential mismatch of design iteration speeds that are accounted for in the NPD model are the design and redesign rates as well as the average vehicle test time. The differences among these speeds will be discussed in the Speed Factor section of this chapter. As in the case of the designing and redesigning, bench testing is shut down at the instant of a vehicle build using (*1 – Build Pulse*) in the equation for *Bench Test TPut*.

Bench Test TPut = (1 - Build Pulse) * Designs in Test (DIT) / Bench Test Time

(16)

Bench Test Time[a] = BTTa

(17)

Bench Test Time[b] = BTTb (18)

In addition to the throughput, there are two main factors that determine the rate at which component errors are discovered in bench testing. The first is the fraction of the stock of *Designs in Test* that actually has a component error. This fraction is represented by *CE Density in DIT* in the model. The second factor is the effectiveness of bench testing at identifying component errors and is captured by the variable *BT Effectiveness* in the model.

CE Density in DIT = ZIDZ (CE in DIT , Designs in Test (DIT)) (19)

BT Effectiveness = 0.5 (20)

The fraction of designs in bench testing that actually has a component error is calculated by dividing the number of component errors associated with the designs in test (*CE in DIT*) by the number of designs in the queue for bench testing (*Designs in Test*). The ***ZIDZ*** expression in the equation for *CE Density in DIT* is a function in VENSIM that divides the first variable in parentheses by the second but sets the value of the expression to zero if the denominator has a value of zero. Simply put, the ***ZIDZ*** function sets the value at **Z**ero **I**f **D**ivided by **Z**ero. The second factor is the effectiveness of bench testing at identifying component errors. This factor is captured by the variable *BT Effectiveness* in the model and is set at a constant value of 0.5 as indicated in equation (20). In effect, this means that only half of the component errors that exist in

the designs that undergo bench testing are identified with their associated designs being sent for rework.

Those designs that undergo bench testing where no component error is discovered are released to the stock of *Designs Released* to await the next build event. As indicated in equation (13), the release rate of these designs is simply the bench test throughput minus the rate of discovering component errors. An additional component of the overall *Release Rate* is the dumping of those designs in the queue for bench testing into the stock of *Designs Released* at the time of the build. This is done to reflect the standard practice in the client company where design engineers remove their designs from the bench testing queue once a build occurs because they expect more complete and reliable test results from vehicle testing. The equation for *Dump DIT* simply takes on the value of the number of designs remaining in the stock of DIT divided by the simulation *TIME STEP* at the instant when a build event occurs (i.e. *Build Pulse* = 1). This value is then used by the simulation, which multiplies the value by the *TIME STEP*, to calculate the flow of releasing designs. In effect the simulation "dumps" the entire stock of *Designs in Test* to stock of *Designs Released* in a single *TIME STEP* at the time of a build.

Dump DIT = Designs in Test (DIT) * Build Pulse / TIME STEP

(21)

The third outflow from the stock of *Designs in Test* is *DIT to RW*. This flow represents the rate at which designs in the stock of *Designs in Test* are sent for rework because they are paired with a design from a coupled component that was found to have an integration error and in the process of coordinating the fix for the integration error, it was determined that the designs from both components would need to be reworked. This type of rework is referred to in the model as external rework and will be discussed in greater detail in the External Rework section of this chapter.

Designs Released

The next set of designs in the main stock and flow chain in the Design Sector of the NPD model is the stock of *Designs Released (DR).* This stock and its associated inflows and outflows are depicted in Figure 100.

This stock represents the set of designs that have been released by the designers awaiting the next vehicle build event. As you can see in Figure 100, there are three inflows (*Release Rate*, *New Designs not Bench Tested*, and *Redesigns not Bench Tes*ted) and two outflows (*DR to DIC* and *DR to RW*) to this stock. The equation for *Designs Released* is as follows:

Designs Released (DR) = INTEG(New Designs not Bench Tested + Release Rate - DR to DIC - DR to RW + Redesigns not Bench Tested , 0)

(22)

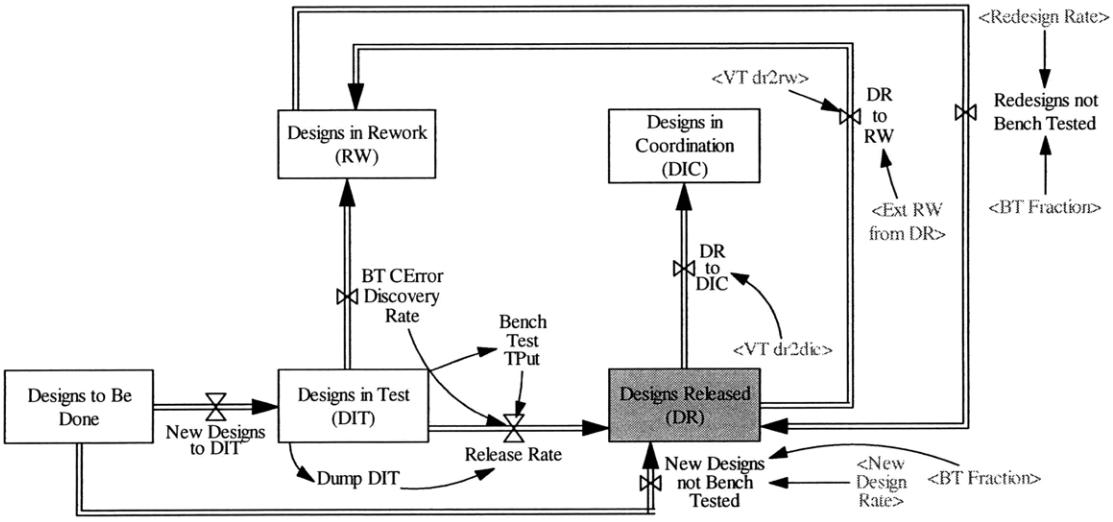


Figure 100 Designs Released (DR)

Two of the inflows - *Release Rate* and *New Designs not Bench Tested* were discussed previously and won't be repeated here. The inflow *Redesigns not Bench Tested* to the stock of

Designs Released represents the rate at which designs are reworked and subsequently released without being sent for bench testing. This inflow is based on the rate at which designs in rework are completed (*Redesign Rate*) and the variable *BT Fraction* which determines the fraction of redesigns that are sent for bench testing upon completion. As is the case with new designs, the fraction of redesigns that are sent for bench testing is based in large part on the amount of time remaining until the next build event with the fraction decreasing as the amount of time decreases. The formulation for *BT Fraction* will be discussed in greater detail later in this chapter.

Redesigns not Bench Tested = Redesign Rate * (1 - BT Fraction)

(23)

The two outflows from the stock of Designs Released – *DR to DIC* and *DR to RW* – represent the rates at which designs that have been previously released are sent for coordination and/or rework as they are discovered to have errors. The primary means for the discovery of errors in the designs that have been released is the vehicle testing that follows each build event. Vehicle testing is able to identify both component and integration errors in designs and, unlike bench testing, vehicle testing is presumed to be perfect (i.e. all errors are discovered) in the NPD model. As a result of discovering component and/or integration errors, designs are sent either directly to rework (DR to RW) or for coordination first (DR to DIC). A discussion of vehicle testing and the decision on whether to send designs with integration errors for coordination (VT dr2dic) or directly for rework (VT dr2rw) will be discussed in detail in later sections of this chapter. The equations for these outflows follow:

DR to DIC = VT dr2dic

(24)

DR to RW = VT dr2rw + Ext RW from DR

(25)

The second component to the flow of designs from the stock of *Designs Released* to the stock of *Designs in RW* is the external rework generated by coordinating designs with integration errors from a coupled component as discussed earlier. This component is represented by the variable *Ext RW from DR* in the equation for *DR to RW.*

Designs in Rework

The next set of designs in the main stock and flow chain in the Design Sector of the NPD model is the stock of *Designs in Rework (RW).* This stock accumulates those designs that have been designated for rework and represents the queue of those waiting for redesign. The stock of *Designs in Rework* and its associated inflows and outflows are depicted in Figure 101.

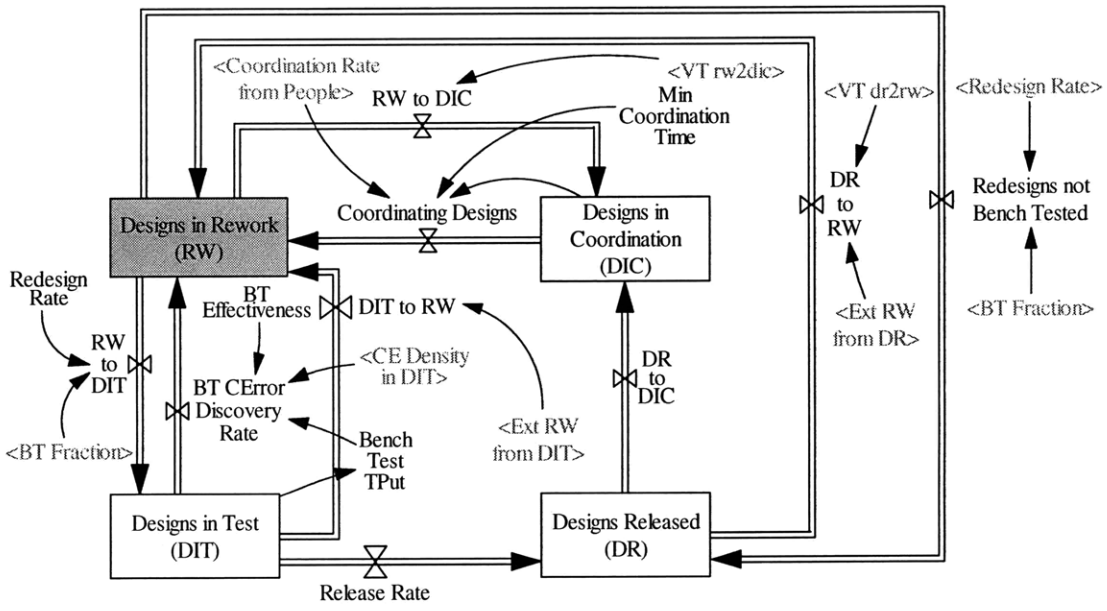


Figure 101 Designs in Rework (RW)

Altogether there are 4 inflows to the stock of *Designs in Rework – BT CError Discovery Rate, DIT to RW, DR to RW* and *Coordinating Designs*. All but the flow of *Coordinating Designs* have been discussed previously and won't be repeated here. There are 3 outflows to the stock of *Designs in Rework (RW) – Redesigns not Bench Tested, RW to DIT* and *RW to DIC*. The first two have been discussed previously and won't be repeated here. The equation for Designs in RW is as follows:

Designs in Rework (RW) = INTEG(BT CError Discovery Rate - RW to DIT + DIT to RW + Coordinating Designs + DR to RW - Redesigns not Bench Tested - RW to DIC , 0)

(26)

The inflow of Coordinating Designs is simply the inflow of designs from the stock of Designs in Coordination as they are coordinated and sent for rework. As defined, an integration error is a

mismatch between two designs from coupled components. By convention, only one of the designs (either from component A or B) has the integration error tied to it and as this design is coordinated it is necessarily sent for rework. This process is represented by the flow Coordinating Designs in the model. As mentioned earlier, with some likelihood the design from the coupled component will also be sent for rework. This is referred to as external rework and will be discussed later in this chapter.

Coordinating Designs = MIN (Coordination Rate from People , Designs in Coordination (DIC) / Min Coordination Time) * (1 - Build Pulse)

(27)

The rate of *Coordinating Designs*, much like the new design and redesign rates, is set at the minimum of the coordination rate that is possible based on constrained resources (*Coordination Rate from People*) and the coordination rate that is possible based on the amount of work available (*Designs in Coordination (DIC)*) and the minimum time it takes to coordinate the fixes required for an integration error (*Min Coordination Time*). The minimum coordination time is set at one week for both components and represents the minimum time required to coordinate a design without any resource constraints. The variable *Coordination Rate from People* will be discussed in detail in the Productivity Sector section of this chapter. As in the case of designing and redesigning, the rate of coordinating designs falls to zero at the instant of a build event using the (1 – Build Pulse) term as a multiplier in the equation for *Coordinating Designs.*

There are 3 outflows to the stock of *Designs in Rework (RW) – Redesigns not Bench Tested, RW to DIT* and *RW to DIC.* The first two have been discussed previously and won't be repeated here. The third outflow *RW to DIC* is much like the outflow *DR to DIC* from *Designs Released* in

that it represents the rate at which designs are found to have an integration error while undergoing vehicle testing and are subsequently sent for coordination. Given that the designs in this case are flowing from *Designs in Rework* they are already identified as requiring rework for having a component error and/or an integration error that has yet to be coordinated. The equation for *RW to DIC* is listed below. The variable *VT rw2dic* will be discussed in detail in the vehicle testing section of this chapter.

RW to DIC = VT rw2dic

(28)

Designs in Coordination

The last set of designs in the main stock and flow chain in the design sector of the NPD model is the stock of *Designs in Coordination (DIC).* This stock accumulates the designs with an identified integration error that are in the queue for coordination between designers from coupled components. The purpose of this coordination is to identify the source of the integration error and the required rework to fix it. The stock of *Designs in Coordination* and its associated inflows and outflows are depicted in Figure 102.

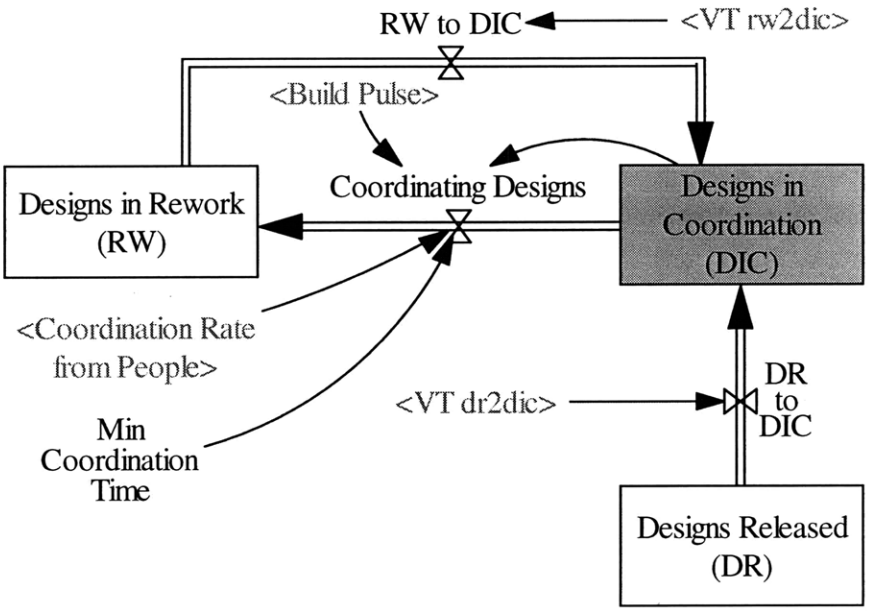


Figure 102 Designs in Coordination (DIC)

There are two inflows to the stock of *Designs in Coordination – DR to DIC* and *RW to DIC*. Both of these flows and their equations – equations (24) and (28) - have already been discussed in detail above. They represent the discovery of designs with integration errors through vehicle testing and the subsequent decision to add these designs to the queue for coordination. The outflow to the stock is *Coordinating Designs* which was also discussed above. As designs are coordinated they are added to the stock of rework and are added to the queue awaiting redesign.

Error Co-Flow Overview

In addition to the main stock and flow chain which accounts for the set of designs for a given component, the NPD model uses a standard co-flow structure to account for the errors associated with the set of designs. In Figure 96 these co-flows are depicted below the main stock and flow

chain. You'll note that there are two co-flow chains used – one for each type of error, component and integration, associated with the designs in the main chain.

Component errors are those errors that arise in a particular component and in the case of the NPD Model to a particular design. Component errors can be of a wide variety but at a basic level can be thought of as a design (or part) that has an error in its "form" or "function" – either it doesn't look the way it should (e.g. in size, shape or style) or it doesn't do its job. Component errors are an inevitable outcome of design work but their likelihood of occurring increases as individual designers are rushed for time and take shortcuts in their design work. By definition, a component error in the NPD model is associated with a single design and each design can have only a single component error at any given time.

Integration errors, on the other hand, are those errors that involve a mismatch between coupled designs. These mismatches, or errors, can be for any number of reasons but on a very basic level they involve two component parts that must either "fit" or "function" together. An integration error occurs when either one of the coupled designs introduces a mismatch by failing to fully integrate its design work with that of its coupled design. Like component errors, integration errors are an inevitable outcome of design work especially in highly complex and coupled systems. However, the likelihood of introducing integration errors dramatically increases as component designers conduct more and more of their design work in isolation with limited interaction with designers working on coupled designs. The tendency for designers to work in isolation and focus solely on their component design increases significantly as designers are rushed for time and get overwhelmed by their work.

By definition, an integration error requires two designs (or parts) to exist. In the NPD model there are two components, A and B, each with a set of designs to be completed. Because these two components are coupled, integration errors can exist between pairs of designs from the different components. That is: an integration error can exist between a design from component A and a design from component B[17]. In the NPD model, each design from a given component can be paired up with at most a single design from the coupled component and each "design pair" can have at most a single integration error between them at any given time. For accounting purposes, the NPD model assigns an integration error, if one exists, to only one of the two coupled designs.

A design can have a component error, an integration error, or both errors and the process for fixing each of these types of errors is different. As such, it is necessary to track the component errors and the integration errors associated with the designs in the main stock and flow chain using a separate co-flow structure for each. These stock and flow chains are nearly identical to each other. The sections below will discuss the structure of the co-flows in detail beginning with the co-flow for component errors. In describing the co-flow structure for integration errors, I will limit the discussion of the structure that is identical to that of the component errors and focus instead on those areas where the structure is different.

The overall structure of both of the error co-flows is very similar to that of the main stock and flow chain of designs. With the exception of the stock of *Designs to Be Done*, there is a corresponding stock of errors for each of the stocks of designs in the main stock and flow chain. As mentioned, there is no need to maintain a corresponding stock of errors for the stock of *Designs*

[17] For the purposes of this model and analysis, integration errors between designs within a single component (e.g. two designs from component A that have a mismatch between them) are not considered.

to Be Done because there aren't any errors in a design that has not been worked on yet. Additionally, there is a flow between each of the stocks in the co-flow structures of the component and integration errors corresponding to the flows in the main stock and flow chain between the stocks of designs. As is the case for the main stock and flow chain for designs, the co-flows for component and integration errors is subscripted in the model to account for both components A and B.

Component Errors

As mentioned, a component error is one that is isolated to a particular design and can be thought of as an error in "form" or "function". By definition, a component error in the NPD model is associated with a single design and each design can have only a single component error at any given time. A high level view of the co-flow structure that is used to account for the component errors associated with the set of designs is depicted in Figure 103.

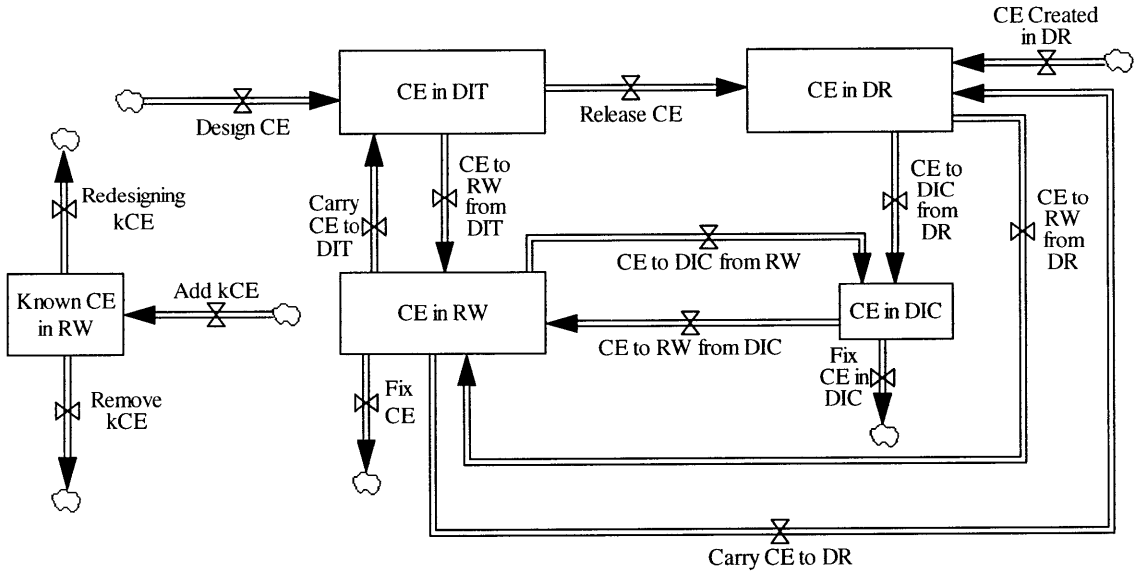


Figure 103 Component Error Co-Flow Structure

Using a co-flow structure, the component errors are tracked along with their associate designs as they progress through the various stages of development. This includes accounting for the generation or creation of component errors as designs are initially worked on and the fixing of component errors as they are redesigned. The NPD model also accounts for component errors that are generated in the process of rework. The co-flow structure for component errors has four main stocks. These include: the stock of component errors associated with the designs in the stock of *Designs in Testing* (*CE in DIT*), the stock of component errors associated with the designs in the stock of *Designs Released* (*CE in DR*), the stock of component errors associated with the designs in the stock of *Designs in Rework* (*CE in RW*) and the stock of component errors associated with the designs in the stock of *Designs in Coordination* (*CE in DIC*). There are inflows and outflows for each of these component error stocks that correspond to the flows between the stocks of designs.

Unlike the main stock and flow chain, which conserves the number of designs at its initial level, the co-flow structure allows errors to be added to and taken out of the system. This is indicated in the depiction of the co-flow structure with arrows coming out of or going into small clouds. For example, in the upper left of Figure 103 the arrow representing the variable *Design CE* emanates from a small cloud indicating that component errors can be added to the system through this flow. On the other hand, the arrow representing the variable *Fix CE* in the bottom center of Figure 103 is pointing into a cloud indicating that component errors can be removed from the system through this flow.

Additionally, the co-flow structure for component errors includes a stock of known component errors in rework (*Known CE in RW*). This stock of component errors does not have a

corresponding stock of designs in the main stock and flow chain. Indeed, it is a co-flow structure of sorts for the stock of *CE in RW*. This stock is needed to account for the known component errors in rework. As will be discussed in detail below, not all component errors in the stock of rework will be known. As the model presumes that only known errors can be fixed, it is important to account for the density of known component errors.

We will now consider each of the stocks of component errors and their associated inflows and outflows in greater detail. There is a subsection included for each of the 5 CE stocks: *CE in DIT*, *CE in DR*, *CE in RW*, *CE in DIC* and *Known CE in RW*. Again the technical documentation for the model is included in Appendix B.

Component Errors in Designs in Test

The first set of component errors in the co-flow structure to be considered is the stock of *CE in DIT*. It represents the stock of component errors associated with the stock of *Designs in Test* (those designs in the queue for bench testing) for a given component. The stock of *CE in DIT* and its associated inflows and outflows are depicted in Figure 104 below.

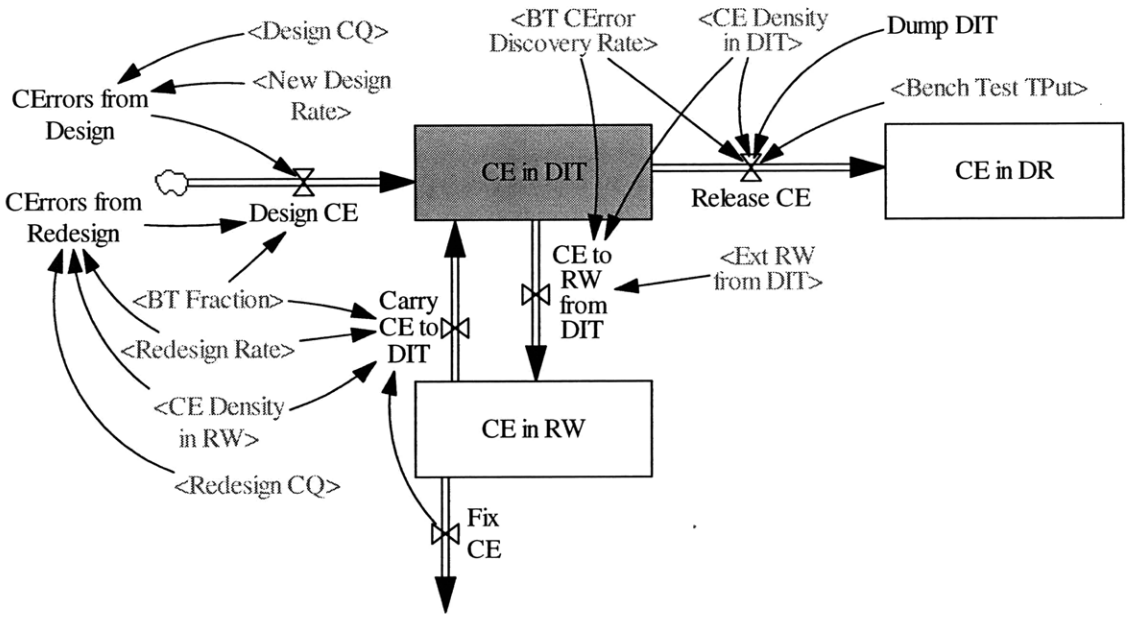


Figure 104 Component Errors in Designs in Test (CE in DIT)

There are two inflows to the stock of *CE in DIT – Design CE* and *Carry CE to DIT* – and two outflows – *Release CE* and *CE to RW from DIT*. The equation for CE in DIT is as follows:

CE in DIT = INTEG(Design CE + Carry CE to DIT - CE to RW from DIT - Release CE , 0)

(29)

The main inflow to the stock of *CE in DIT* is *Design CE*. This variable represents the addition of component errors to the stock of *CE in DIT* as they are created as designs are either initially designed (*CErrors from Design*) or redesigned (*CErrors from Redesign*).

Design CE = (CErrors from Design + CErrors from Redesign) * BT Fraction

(30)

The rate at which component errors are generated in design or redesign is dependent on the rate of design (or redesign) and the associated component quality of design (*Design CQ* and *Redesign CQ* respectively).

CErrors from Design = New Design Rate * (1 - Design CQ)

(31)

CErrors from Redesign = Redesign Rate * (1 - Redesign CQ) * (1 - CE Density in RW)

(32)

The equations for *New Design Rate* and *Redesign rate* have been previously discussed and so won't be repeated here. The *Design CQ* is a variable that represents the component quality of design. The formulation for Design CQ (and Redesign CQ) will be discussed in detail in the Quality section of this chapter. Valued between 0 and 1, Design CQ represents the fraction of designs completed without introducing a component error. Thus the complement (*1- Design CQ*) when multiplied by the *New Design Rate* generates the number of component errors introduced while completing new designs. The formulation for *CErrors from Redesign* is equivalent to the one for new designs except that the term (*1-CE Density in RW*) is added. In the case of redesign, new component errors can only be introduced in those designs in rework that do not have a component error. The NPD model assumes that each design can only have one component error. The variable *CE Density in RW* represents the density of component errors in the stock of *Designs in Rework* thus the complement (*1-CE Density in RW*) is the fraction of designs in rework that do not have a component error and are eligible to have one created in the process of redesign.

Not all of the component errors created in design and redesign are added to the stock of *CE in DIT* because not all initial designs and redesigns are sent for bench testing. Thus, we include *BT Fraction* in the equation for *Design CE* effectively adding a proportionate amount of component errors to the stock of *CE in DIT* to the fraction of designs and redesigns that are sent for bench testing.

The second source of component errors being added to the stock of *CE in DIT* is "carrying over" component errors from the stock of *CE in RW* (those associated with the stock of designs in rework) represented by the variable *Carry CE to DIT* in the model. "Carrying over" an error effectively means that a design with an error is redesigned without fixing it. This can occur in two ways. The first occurs when a design in the stock of rework has an unknown error. Again, the model presumes that only known errors can be fixed and thus a design with an unknown component error will see the component error carried over. The second way that an error is carried over is when a known error is redesigned but not fixed due to poor quality. Thus the equation for *Carry CE to DIT* is as follows:

Carry CE to DIT= (Redesign Rate * CE Density in RW - Fix CE) * BT Fraction

(33)

To calculate the rate at which component errors are carried over from the stock of CE in RW, it is necessary to determine the rate at which component errors are being redesigned (*Redesign Rate * CE Density in RW*) and subtract out the number of component errors that are fixed (*Fix CE*). The equation for *Redesign Rate* was previously discussed and the formulation of *CE Density in RW* is

analogous to that of *CE Density in DIT*. The formulation for fixing component errors will be discussed in the CE in RW section that follows.

CE Density in RW = ZIDZ (CE in RW , Designs in Rework (RW))

(34)

As in the case of creating new component errors through redesign, not all component errors that are carried over are added to the stock of *CE in DIT*. The inclusion of the variable *BT Fraction* as a multiplier in equation (33) ensures that the amount of carried over CE added to the stock of *CE in DIT* is proportionate to the fraction of redesigns that are sent for bench testing.

The first outflow from the stock of *CE in DIT* to be considered is the releasing of component errors to the stock of *CE in DR* (*Release CE* in the model). This occurs in two ways. The first is when designs in the stock of *Designs in Test* that have component errors are bench tested but the component error is not discovered and thus the designs and their associated errors are released. The rate of releasing component errors in this manner can be simply calculated by multiplying the throughput rate of bench testing by the fraction of designs in *DIT* that have component errors (*CE Density in DIT*) and subtracting out the rate at which component errors are being discovered through bench testing.

Release CE = Bench Test TPut * CE Density in DIT - BT CError Discovery Rate + Dump DIT * CE Density in DIT

(35)

The second way in which component errors are released to the stock of *CE in DR* is in conjunction with dumping the designs in the bench test queue upon a build event. This is calculated simply by multiplying the rate of dumping designs (*Dump DIT*) by the density of component errors in the stock of *Designs in Test* (*CE Density in DIT*).

The second outflow from the stock of *CE in DIT* is the rate at which component errors are sent to the stock of *CE in RW* as their associated designs are sent for rework. This rate is represented by the variable *CE to RW from DIT* in the model and its equation is as follows:

CE to RW from DIT = BT CError Discovery Rate + Ext RW from DIT * CE Density in DIT

(36)

There are two processes by which component errors and their associated designs are sent for rework. The first is when designs are discovered to have component errors in bench testing (*BT CError Discovery Rate*) which was discussed earlier. The result is that a component error is moved to the stock of *CE in RW* from the stock of *CE in DIT* for every design that is discovered to have a component error in bench testing and is subsequently sent for rework. The second way that a component error is sent to the stock of *CE in RW* is when its associated design is sent to rework as a result of coordinating an integration error in a design from a coupled component (*Ext RW from DIT*). The formulation for Ext RW from DIT will be discussed in the External RW section of this chapter. In this case, the number of component errors that are sent along with the designs is proportionate to the fraction of designs in *DIT* that have component errors (*CE Density in DIT*).

Component Errors in Designs Released

The next set of component errors in the co-flow structure is the stock of *CE in DR*. It represents the stock of component errors associated with the stock of *Designs Released* (those designs released awaiting the next build event) for a given component. The stock of *CE in DR* and its associated inflows and outflows are depicted in Figure 105.

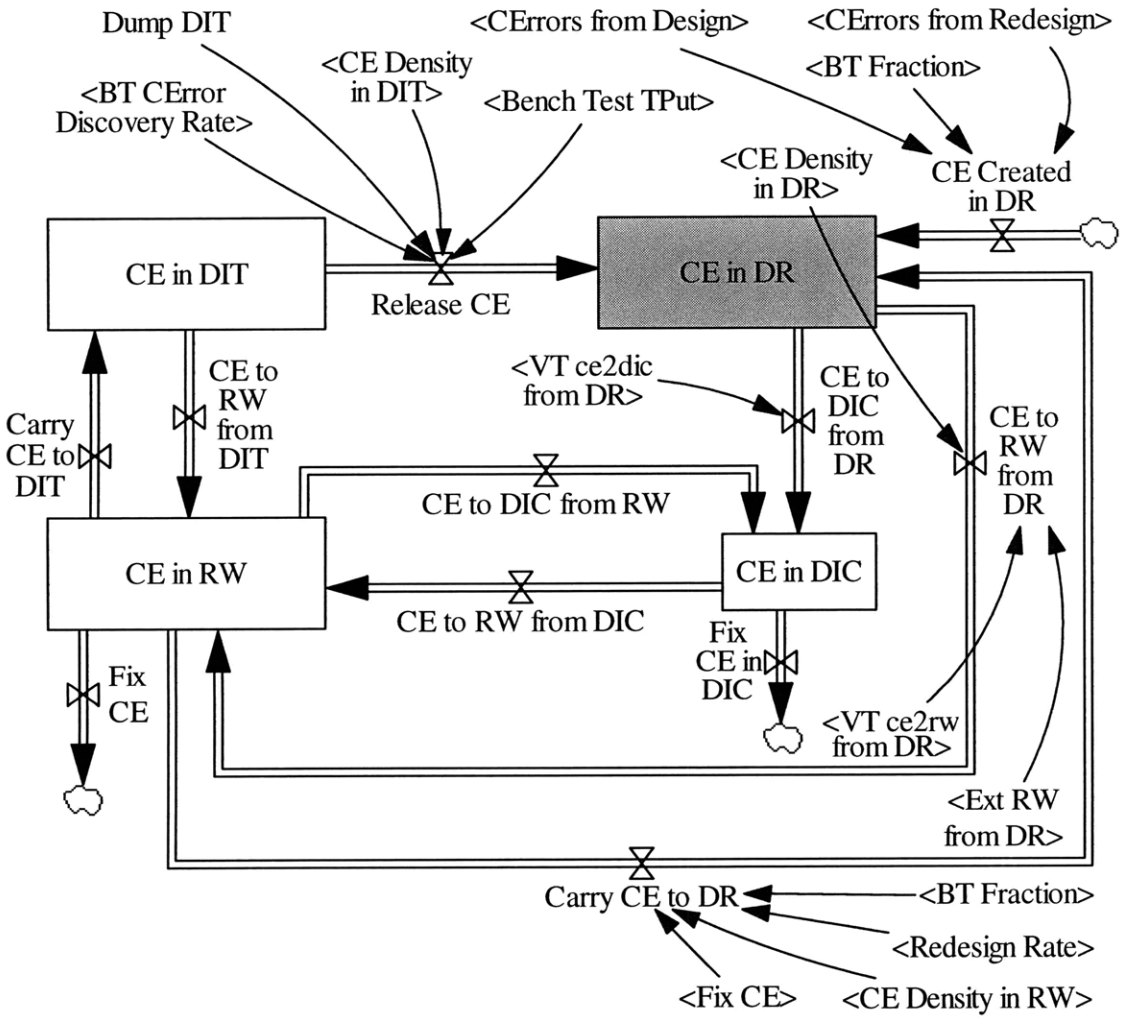


Figure 105 Component Errors in Designs Released (CE in DR)

There are three inflows to the stock of *CE in DR – Release CE, CE Created in DR* and *Carry CE to DR* – and two outflows – *CE to DIC from DR* and *CE to RW from DR*. The equation for *CE in DR* is as follows:

CE in DR = INTEG(Release CE - CE to RW from DR + Carry CE to DR + CE Created in DR - CE to DIC from DR , 0)

(37)

The formulation of the inflow *Release CE* was discussed earlier and won't be repeated here. It represents the flow of component errors that are released along with their associated designs when component errors go undiscovered in bench testing and when designs are "dumped" from the bench test queue at the time of a build. The inflow *CE Created in DR* is analogous to the inflow of *Design CE* to the stock of *CE in DIT* except the component errors in this flow are those associated with the designs that are not sent for bench testing upon initial design or redesign. Thus in the equation for *CE Created in DR* the term *(1-BT Fraction)* is used to add component errors in the same proportion as the fraction of designs and redesigns that are not sent for bench testing.

CE Created in DR = (CErrors from Design + CErrors from Redesign) * (1 - BT Fraction)

(38)

The third inflow to *CE in DR* is *Carry CE to DR* and represents the "carrying over" of component errors from the stock of *CE in RW*. The formulation is analogous to that of *Carry CE to DIT* in equation (33) except that the component errors that are carried over (i.e. not fixed) are associated with the designs that are not sent for bench testing. Again the use of the term (*1-BT Fraction*) in equation (39) provides this accounting.

Carry CE to DR = (Redesign Rate * CE Density in RW - Fix CE) * (1 - BT Fraction)
(39)

The primary reason that component errors flow out of the stock of *CE in DR* is the discovery of these errors through vehicle testing. When component errors are discovered in vehicle testing, the errors and their associated designs are either sent directly for redesign (*CE to RW from DR*) or in some cases they are sent for coordination prior to redesign (*CE to DIC from DR*) if they are found to have both a component error and an integration error. The component errors sent to the stock of *CE in RW* also include those component errors associated with the designs from the stock of *Designs Released* that are identified for rework based on the coordination of an integration error in a coupled design.

CE to RW from DR = VT ce2rw from DR + Ext RW from DR * CE Density in DR
(40)

CE to DIC from DR = VT ce2dic from DR
(41)

Vehicle testing and the variables *VT ce2rw from DR* and *VT ce2dic from DR* will be discussed in detail in the Vehicle Testing section of this chapter. You'll notice that the equation for *CE to RW from DR* also accounts for the component errors sent to *CE in RW* based on the discovery of external rework. This is done simply by multiplying the rate at which released designs are sent for external rework (*Ext RW from DR*) by the fraction of released designs that have a component error (*CE Density in DR*).

CE Density in DR = ZIDZ (CE in DR , Designs Released (DR))

(42)

Component Errors in Designs in Rework

The next set of component errors in the co-flow structure is the stock of component errors that are associated with the designs in the queue for redesign (*Designs in Rework*). The stock of *CE in RW* and its associated flows is depicted in Figure 106.

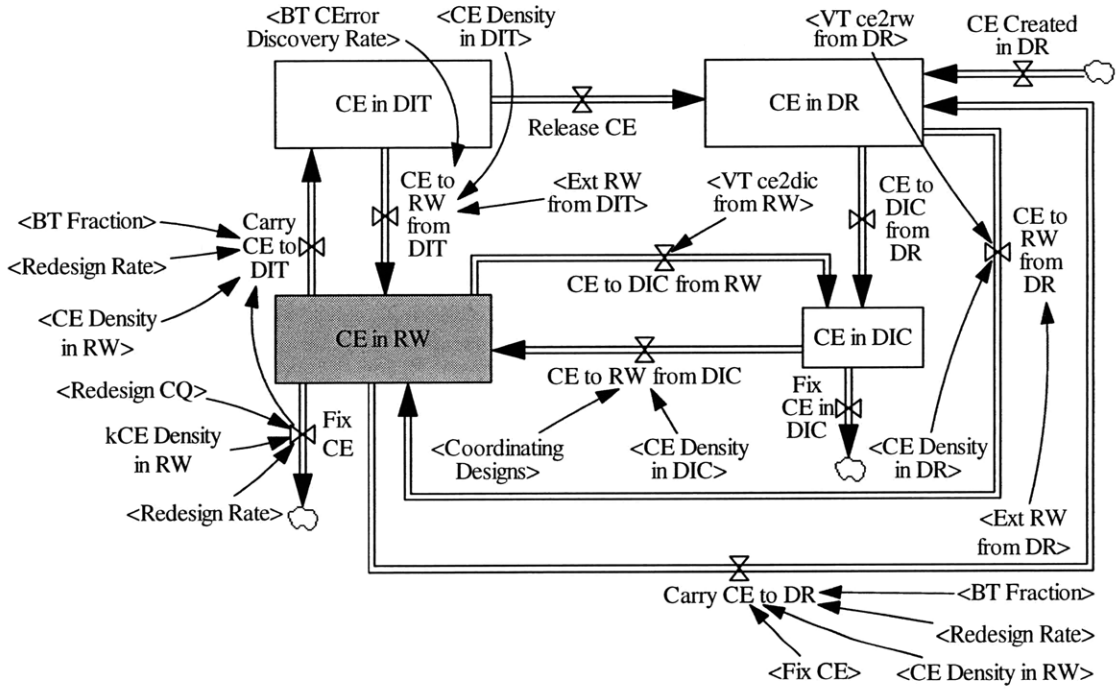


Figure 106 Component Errors in Designs Rework (CE in RW)

There are three inflows to the stock of *CE in RW – CE to RW from DIT, CE to RW from DR* and *CE to RW from DIC* – and four outflows –. *Fix CE, CE to DIC from RW*, C*arry CE to DIT,* and *Carry CE to DR.* The equation for *CE in RW* is as follows:

CE in RW = INTEG(CE to RW from DIT + CE to RW from DR - Carry CE to DIT - Fix CE - Carry CE to DR - CE to DIC from RW + CE to RW from DIC , 0)

(43)

The primary inflow of component errors to the stock of *CE in RW* is the result of bench testing and is represented by the flow *CE to RW from DIT* in the model. The formulation for this flow was discussed previously. You'll remember that this flow also includes those component errors associated with designs identified for external rework. The second inflow of component errors, *CE to RW from DR*, was discussed in the previous section and includes component errors discovered in vehicle testing as well as component errors associated with released designs that are identified for external rework. The third inflow to the stock of *CE in RW* comes from component errors associated with designs that are coordinated prior to being sent for rework and is represented by the flow *CE to RW from DIC* in the model. The formulation for this flow is as follows:

CE to RW from DIC = Coordinating Designs * CE Density in DIC

(44)

CE Density in DIC = ZIDZ (CE in DIC , Designs in Coordination (DIC))

(45)

Simply multiplying the rate at which designs are being coordinated (*Coordinating Designs*) by the fraction of designs in coordination that contain a component error (*CE Density in DIC*) yields the rate at which component errors are added to the stock of *CE in RW* from coordination.

The primary outflow of component errors from the stock of *CE in RW* is the result of fixing component errors in the process of redesign. This process is captured by the flow *Fix CE* in the

model. Prior to discussing in detail the formulation for fixing component errors, it is important to note that those component errors that are not fixed while their associated designs are redesigned are carried over. This process results in the outflow of component errors from the stock of *CE in RW* to either the stock of *CE in DIT* (if the associated design is sent for bench testing after redesign) or the stock of *CE in DR* (if the associated design is not bench tested). The formulation for these two flows was discussed previously. The third outflow of component errors from the stock of *CE in RW* is the result of discovering integration errors through vehicle testing and the subsequent decision to send the associated designs for coordination. This process is represented by the flow *CE to DIC from RW* in the model and its equation follows:

CE to DIC from RW = VT ce2dic from RW

(46)

The formulation for the variable *VT ce2dic from RW* will be discussed in detail in the Vehicle Testing section of this chapter.

The equation for fixing component errors (Fix CE) is as follows:

Fix CE = Redesign Rate * kCE Density in RW * Redesign CQ

(47)

The variables Redesign Rate and Redesign CQ have been discussed previously. The variable *kCE Density in RW* represents the fraction of designs in the stock of Designs in RW that have known component errors. By multiplying this fraction by the *Redesign Rate*, we can calculate the rate at which designs with known component errors are being redesigned. By further multiplying

by the component quality of redesign (*Redesign CQ*) we can calculate the rate at which component errors are being fixed.

kCE Density in RW = ZIDZ (Known CE in RW , Designs in Rework (RW))

(48)

The formulation of *kCE Density in RW* is analogous to all of the other component error densities in the co-flow structure. However, this formulation requires us to distinguish the known component errors - those discovered through bench testing and/or vehicle testing – from the unknown component errors associated with the set of designs in rework. Since the stock of *CE in RW* includes both known and unknown component errors we need to maintain a separate co-flow for the known component errors associated with the stock of *Designs in Rework.* This separate co-flow structure will be discussed in the next section.

Known Component Errors in Designs in Rework

In order to be able to distinguish between the known and unknown component errors associated with the stock of Designs in Rework, we need to maintain a separate co-flow structure for the known component errors. This is done with the stock of *Known CE in RW* and its associated inflows and outflows, a picture of which is included in Figure 107.

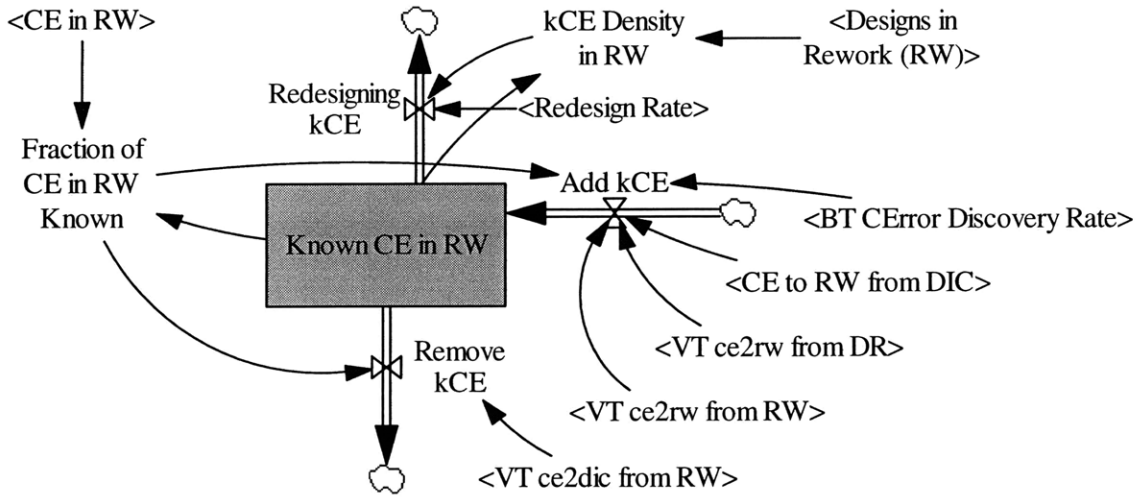


Figure 107 Known Component Errors in Designs in Rework (Known CE in RW)

There is one inflow to the stock of *Known CE in RW – Add kCE* which represents the addition of component errors to the stock of known component errors in rework upon discovery either through bench testing or vehicle testing. There are two outflows to the stock of *Known CE in RW*. The main outflow, *Redesigning kCE*, as the name implies represents removing component errors from the stock of known CE as designs in rework are redesigned. The second outflow, *Remove kCE*, is the result of discovering an integration error in a design with a known component error and sending it for coordination. The equation for *Known CE in RW* is included below and the formulation of each of the flows will be discussed in more detail.

Known CE in RW = INTEG(Add kCE - Remove kCE - Redesigning kCE , 0)

(49)

Adding component errors to the stock of *Known CE in RW* is done as component errors are discovered either in bench testing or vehicle testing and as designs with known component errors and integration errors are coordinated. As such, the equation for the flow *Add kCE* is as follows:

Add kCE = BT CError Discovery Rate + CE to RW from DIC + VT ce2rw from DR + VT ce2rw from RW * (1 - Fraction of CE in RW Known)

(50)

Fraction of CE in RW Known = ZIDZ (Known CE in RW , CE in RW)

(51)

The first three variables included in this equation have been discussed previously. *BT CError Discovery Rate* represents the rate at which component errors are discovered in bench testing and their associated designs are sent for rework; thus each component error discovered in this manner is added to the stock of *Known CE in RW*. The variable *CE to RW from DIC* represents the component errors that are sent to the stock of *CE in RW* as designs are coordinated. The NPD model assumes perfect vehicle testing which means that all component and/or integration errors are discovered in vehicle testing. It also assumes that a design that has both types of error will have both errors discovered at the same time. As such, for any design that has a known integration error – the reason for coordination – and a component error, we can assume that the component error is also known. Therefore, every component error that is moved to the stock of *CE in RW* as its associated design is coordinated (via *CE to RW from DIC*) can be added to the stock of *Known CE in RW*. The third variable, *VT ce2rw from DR*, represents those component errors associated with released designs that are discovered through vehicle testing and sent for rework along with their associated design.

The last term [*VT ce2rw from RW * (1 – Fraction of CE in RW Known)*] in the equation for adding known component errors to the stock of *Known CE in RW* represents the discovery of previously unknown component errors in the designs in the stock of rework. The way that a design makes its way into the stock of *Designs in RW* with an unknown component error is through the process of external rework. As designs are designated for rework as a result of coordinating integration errors in designs from a coupled component, they take with them their associated component errors. These component errors may or may not be known. One can imagine a design with a component error that was released without bench testing and thus the component error is unknown. If a corresponding design from a coupled component has an integration error that is discovered, the resulting coordination may require both designs to be reworked. Thus the released design with the unknown component error would be sent for rework and its component error would remain unknown until it is discovered in subsequent testing.

The method for accounting for the discovery of these previously unknown component errors that are in the stock of *CE in RW* is to take the rate of discovery of component errors in rework (*VT ce2rw from RW*) and multiply it by the fraction of component errors that are unknown (*1- Fraction of CE in RW Known*) and add it to the stock of *Known CE in RW*. The discovery of previously known component errors does not add to the stock of *Known CE in RW*.

The formulation for *Fraction of CE in RW Known* as shown in equation (51) is straightforward and is computed much like the component error densities discussed previously. In this case we determine the fraction of all component errors in the stock of *CE in RW* that are known.

The main outflow from the stock of *Known CE in RW* is *Redesigning kCE*. As the name implies, this occurs when designs with known component errors are redesigned and thus leave the stock of Designs in Rework along with their associated errors. The formulation for this rate of flow is simply to multiply the rate at which designs are being reworked (*Redesign Rate*) by the fraction of designs in rework that have known component errors (*kCE Density in RW*).

Redesigning kCE = Redesign Rate * kCE Density in RW

(52)

The second outflow from the stock of *Known CE in RW* is *Remove kCE* and represents the rate at which known component errors are sent along with their designs for coordination upon discovery of an integration error in vehicle testing. The variable *VT ce2dic from RW* included in the equation for *Remove kCE* below represents the rate at which designs in rework with both types of errors – component and integration – are discovered in vehicle testing and subsequently sent for coordination. However, only some of the component errors discovered in the designs in rework were previously known and thus the rate at which these component errors are discovered and sent along with their designs sent for coordination must be multiplied by the fraction of component errors in rework that are known (*Fraction of CE in RW Known*) in order to determine how many should be removed from the stock of *Known CE in RW*. The formulation for *VT ce2dic from RW* will be discussed in the Vehicle Testing section of this chapter.

Remove kCE = VT ce2dic from RW * Fraction of CE in RW Known

(53)

Component Errors in Designs in Coordination

The last stock in the component error co-flow structure is that of component errors in designs in coordination (CE in DIC). The stock and its associated flows are depicted in Figure 108.

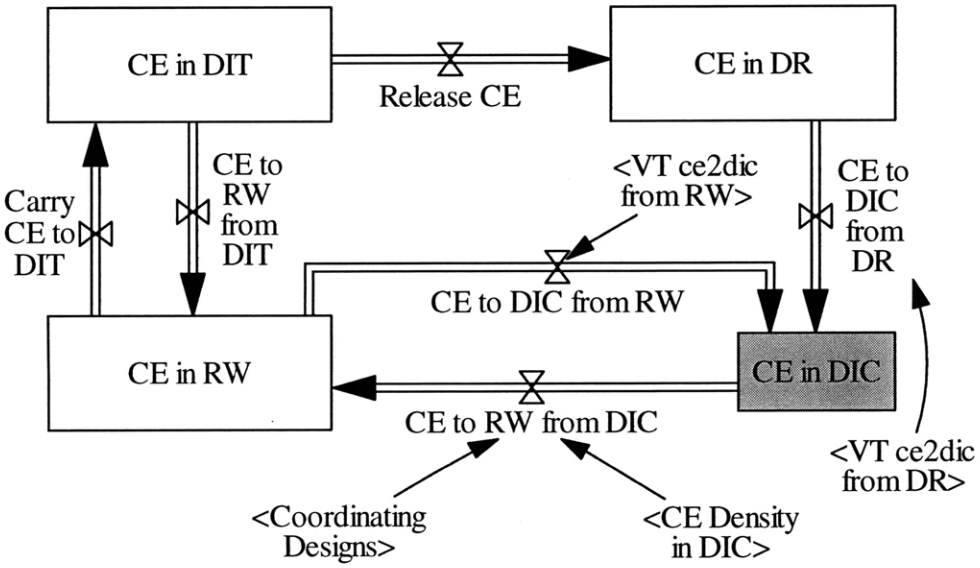


Figure 108 Component Errors in Designs in Coordination (CE in DIC)

The stock of component errors in designs in coordination has two main inflows – *CE to DIC from DR* and *CE to DIC from RW*. These represent the discovery of integration errors through vehicle testing in designs that also have component errors. The formulation for these flows was already discussed. The only outflow from the stock of *CE in DIC* is the flow of component errors to the stock of rework as designs with both types of errors are coordinated represented by the variable *CE to RW from DIC* in the model. The formulation of this variable has also been previously discussed.

Integration Errors

As mentioned, integration errors are those errors that involve a mismatch between coupled designs that must either "fit" or "function" together. An integration error occurs when either one of the coupled designs introduces a mismatch due to poor integration quality. In the model, each design from a given component can be paired up with at most a single design from the coupled component and each "design pair" can have at most a single integration error between them at any given time. The NPD model assigns an integration error, if one exists, to only one of the two coupled designs. The formulation for how this accounting is done will be discussed below.

A design can have a component error, an integration error, or both errors and the processes for identifying and fixing each of these types of errors are different. As such it is necessary to track the integration errors associated with the designs in the main stock and flow chain using a separate co-flow structure from the component error co-flow just discussed. A high level view of the co-flow structure that is used to account for the integration errors is depicted in Figure 109.

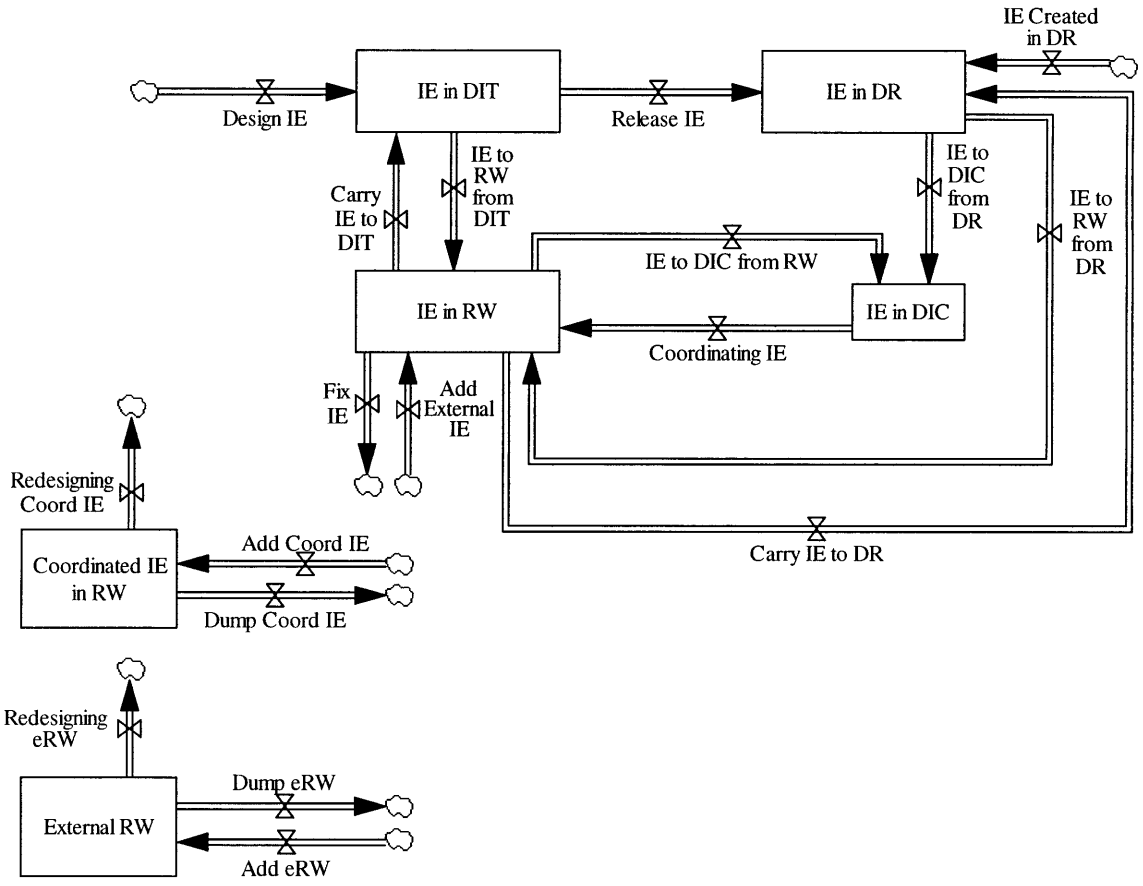


Figure 109 Integration Error Co-Flow Structure

You should note that the co-flow structure used to account for the integration errors associated with the designs in the main stock and flow chain is almost identical to the co-flow structure used for components errors as depicted in Figure 103. There are 4 stocks of integration errors (IE) that correspond to the stocks in the main chain of the designs in progress. The stock of *Coordinated IE* is analogous to the stock of *Known CE in RW* from the component error co-flow structure. Like known component errors, only coordinated integration errors can be fixed through redesign. The additional stock of *External RW* is needed to account for the additional errors and rework (i.e.

Phantom Work) that are created when components iterating at different speeds don't get all of the necessary rework done prior to a build event.

We will now consider each of the stocks of integration errors and their associated inflows and outflows in greater detail. There is a subsection included for each of the 6 IE stocks: *IE in DIT*, *IE in DR*, *IE in RW*, *IE in DIC*, *Coordinated IE in RW*, and *External RW*. Again the technical documentation for the model is included in Appendix B.

Integration Errors in Designs in Test

Analogous to the stock of CE in DIT, the stock of integration errors in the designs in test (*IE in DIT*) represents the stock of integration errors associated with the stock of *Designs in Test* (those designs in the queue for bench testing) for a given component. The stock of *IE in DIT* and its associated inflows and outflows are depicted in Figure 110. There are two inflows to the stock of *IE in DIT – Design IE* and *Carry IE to DIT* – and two outflows – *Release IE* and *IE to RW from DIT*. The equation for IE in DIT is as follows:

IE in DIT = INTEG(Design IE + Carry IE to DIT - IE to RW from DIT - Release IE , 0)

(54)

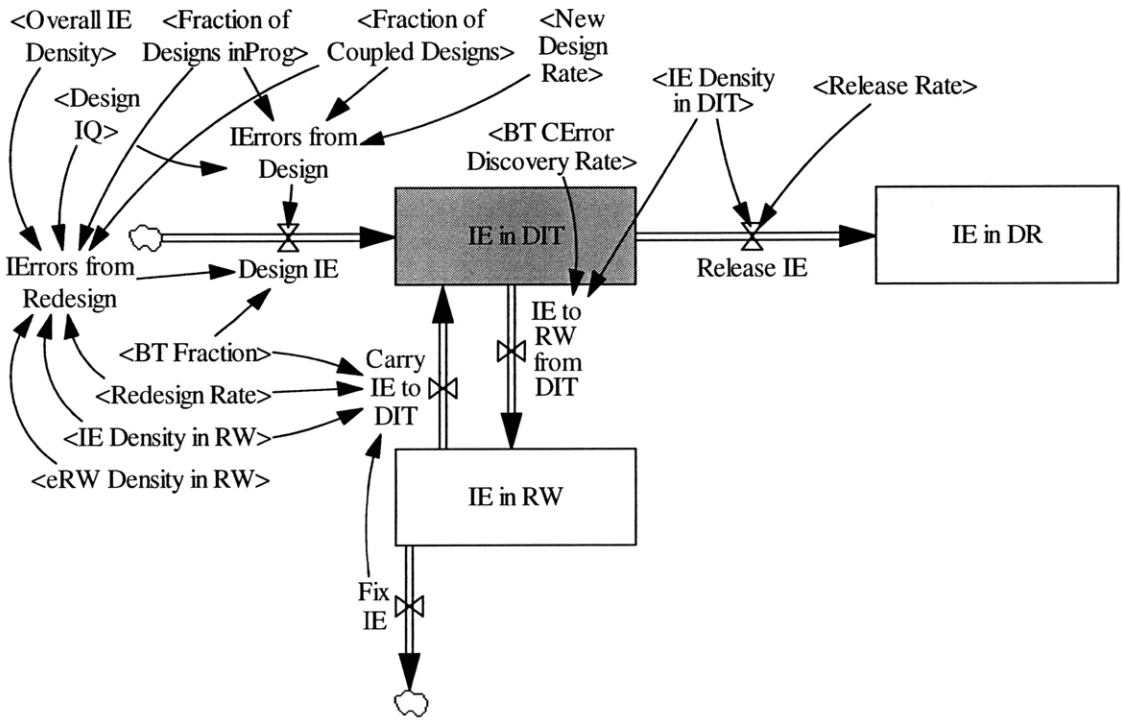


Figure 110 Integration Errors in Designs in Test (IE in DIT)

While the inflows and outflows to the stock of IE in DIT are analogous to those for CE in DIT, their formulations are different. Each will be discussed below.

The equation for the inflow *Design IE* is exactly analogous to that of *Design CE* and is made up of integration errors from design and redesign that are associated with the designs that are sent for bench testing.

Design IE = (IErrors from Design + IErrors from Redesign) * BT Fraction

(55)

The driving processes behind the generation of integration errors in design and redesign are the New Design Rate and the Redesign Rate respectively, and a term (*1-Design IQ*) used to account for the level of integration quality used during design and redesign activity. The formulation for the design and redesign rates has been discussed and the formulation of integration quality is completely analogous to that of component quality. You'll note that there are separate equations required for the generation of integration errors for components A and B from both design and redesign. While the equations are similar to those in the component error co-flow there are important differences. These will be discussed below.

IErrors from Design[a] = New Design Rate[a] * (1 - Design IQ[a]) * Fraction of Coupled Designs[a] * Fraction of Designs inProg[b]

(56)

IErrors from Design[b] = New Design Rate[b] * (1 - Design IQ[b]) * Fraction of Coupled Designs[b] * Fraction of Designs inProg[a]

(57)

In calculating the rate of generating integration errors from designing, we use the *Fraction of Coupled Designs* to the equation to account for the probability that a given design being completed is in fact coupled with a design from the other component and thus able to generate an integration error. The *Fraction of Coupled Designs* for each component is a fixed number between 0 and 1

that represents the fraction of designs from a given component that are coupled with a design from the other component.

Additionally, because an integration error by definition requires two coupled designs to exist we use the fraction of designs in progress (Fraction of Designs inProg) of the OTHER component to account for the probability that a given design is coupled with a design from the other component that has at least been initially designed. This means that of two coupled designs, the first one to be completed CANNOT, by definition and formulation, generate an integration error. It is not until the second of the two coupled designs is completed that an integration error can be generated.

The formulation for integration errors from redesigning for components A and B is similar to that from designing above. Each of the equations includes the rate of redesign, a term to account for the integration quality, the fraction of coupled designs, and the fraction of designs from the coupled component that are in progress. Analogous to the equation for component errors from redesign, there is a term included (*1-IE Density*) to account for the fact that there can only be one integration error per design and some of the designs that are being redesigned might already have an integration error.

IErrors from Redesign[a] = Redesign Rate[a] * (1 - Design IQ[a]) * (1 - IE Density in RW[a]) * (1 - eRW Density in RW[a]) * Fraction of Coupled Designs[a] * Fraction of Designs inProg[b] * (1 - Overall IE Density)

(58)

IErrors from Redesign[b] = Redesign Rate[b] * (1 - Design IQ[b]) * (1 - IE Density in RW[b]) * (1 - eRW Density in RW[b]) * Fraction of Coupled Designs[b] * Fraction of Designs inProg[a] * (1 - Overall IE Density)

(59)

The first new term that is included in the formulation of integration errors from redesign is (*1- eRW Density in RW*). This term is used to prevent the generation of an integration error in the redesign of a design that is in rework due external rework. You'll recall that external rework, by definition, means that the given design is coupled with a design from the other component that already has an integration error attached to it. Therefore, a new integration error cannot be generated for the given design.

The second new term included in this formulation is the last term in each of the equations (58) and (59). This term (*1- Overall IE Density*) ensures that integration errors throughout all of the stock do not exceed the total number possible integration errors (i.e. the number of coupled design pairs). The variable *Overall IE Density* calculates the fraction of possible integration errors that currently exist. Therefore the complement of this fraction adds the probability of being able to introduce a new integration error into the formulation of *IErrors from Redesign.*

The inflow *Carry IE to DIT* is the rate at which integration errors are "carried over" to *IE in DIT*. This is the rate at which designs with existing integration errors are redesigned where the integration errors are not fixed. The formulation is completely analogous to the corresponding inflow for component errors which was discussed previously.

Carry IE to DIT = (Redesign Rate * IE Density in RW - Fix IE) * BT Fraction

(60)

The outflow *Release IE* is the rate at which integration errors are released to the stock of integration errors in designs released (*IE in DR*). This occurs as designs with integration errors are released after bench testing or the designs are released without bench testing upon a build event (*Dump DIT*). The formulation is straightforward in that it assumes that integration errors will be released along with designs in proportion to the density of integration errors in the stock of designs in test (*IE Density in DIT*).

Release IE = Release Rate * IE Density in DIT

(61)

The outflow *IE to RW from DIT* is the rate at which integration errors in designs in testing (*IE in DIT*) are moved to the stock of integration errors in rework (*IE in RW*). This occurs when designs with integration errors also have component errors that are discovered in bench testing and are moved to rework. Like releasing integration errors, the formulation assumes that integration errors will be sent to rework along with designs discovered to have component errors through bench testing (*BT CError Discovery Rate*) in proportion to the density of integration errors in the stock of designs in test (*IE Density in DIT*).

IE to RW from DIT = BT CError Discovery Rate * IE Density in DIT

(62)

Integration Errors in Designs Released

Those integration errors associated with the set of designs released awaiting the next build event are represent by the stock of integration errors in designs released (IE in DR) which is

depicted along with its inflows and outflows in Figure 111. The structure is analogous to that of the stock of CE in DR discussed previously.

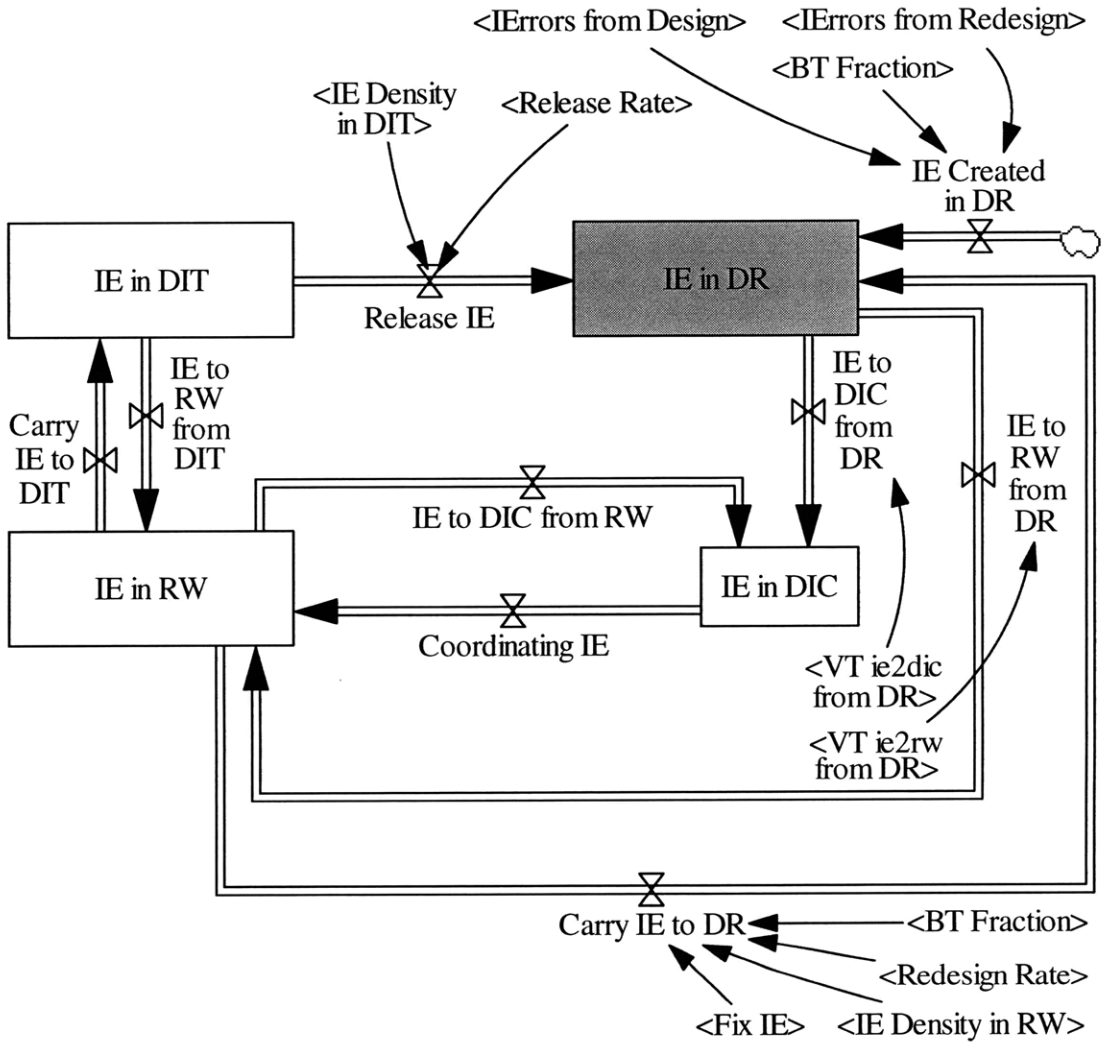


Figure 111 Integration Errors in Designs Released (IE in DR)

The differential equation for the stock of IE in DR is as follows:

IE in DR = INTEG(Release IE - IE to DIC from DR + IE Created in DR + Carry IE to DR - IE to RW from DR , 0)

(63)

The first of three inflows in the equation for *IE in DR*, *Release IE* was just discussed and its equation (61) is provided above. The second inflow, *IE Created in DR*, is analogous to the corresponding equation from the component error co-flow. It represents the creation of integration errors through design and redesign in those designs that are not sent for bench testing. The third inflow to *IE in DR* is represented by the variable *Carry IE to DR* which again is analogous to the corresponding flow in the component co-flow structure.

IE Created in DR = (IErrors from Design + IErrors from Redesign) * (1 - BT Fraction)

(64)

Carry IE to DR = (Redesign Rate * IE Density in RW - Fix IE) * (1 - BT Fraction)

(65)

There are two outflows to the stock of *IE in DR – IE to DIC from DR* and *IE to RW from DR*. Both of these outflows are analogous to the corresponding flows from the component co-flow and their formulations are analogous as well. They represent the rate at which integration errors are discovered in vehicle testing and their corresponding designs are sent for either coordination or directly to rework. The formulation for *VT ie2dic from DR* and *VT ie2rw from DR* will be discussed in the Vehicle Testing section of this chapter. You'll recall that designs that are found to have both an integration error and a component error may be sent directly for rework without coordination based on the time available until the next build and available resources. The formulation for this decision process will be discussed as part of the Coordination Fraction section of this chapter.

IE to DIC from DR= VT ie2dic from DR

(66)

IE to RW from DR = VT ie2rw from DR

(67)

Integration Errors in Designs in Coordination

The next set of integration errors to be considered are those that are associated with the set of designs in coordination and are represented by the stock of *IE in DIC*. This stock of integration errors and its associated inflows and outflow is depicted in Figure.

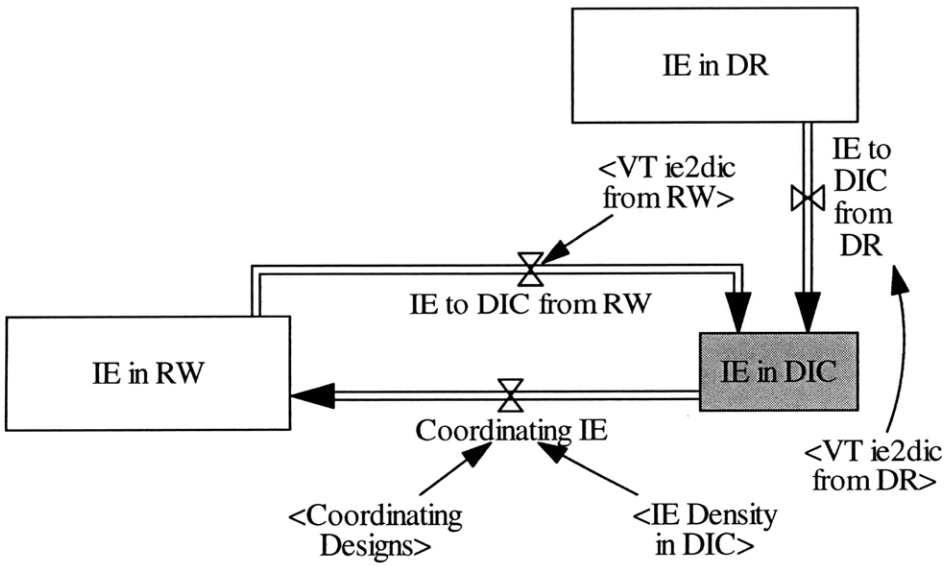


Figure 112 Integration Errors in Designs in Coordination (IE in DIC)

There are two inflows to the stock of *IE in DIC – IE to DIC from DR* and *IE to DIC from RW* – the first of which was discussed in the previous section. There is one outflow to the stock, *Coordinating IE*, which sends integration errors along with their associated designs as they are coordinated. The differential equation that describes the state of *IE in DIC* at any point in time is as follows:

IE in DIC = INTEG(IE to DIC from DR + IE to DIC from RW - Coordinating IE , 0)

(68)

As mentioned, the formulation for *IE to DIC from DR* has already been discussed. The second inflow, *IE to DIC from RW*, is similar in that it too represents the inflow of integration errors to the stock of *IE in DIC* based on the discovery of integration errors in designs through vehicle testing and the subsequent decision to send these designs for coordination. The formulation for *VT ie2dic from RW* as included in equation (69) will be discussed in detail in the Vehicle Testing section of this chapter.

IE to DIC from RW = VT ie2dic from RW

(69)

The only outflow to the stock of *IE in DIC* is *Coordinating IE* which represents the transfer of integration errors to the stock of *IE in RW* as their associated designs are coordinated. The formulation ensures that integration errors are transferred along with the coordinated designs in

proportion to the fraction of designs in coordination (DIC) that have integration errors (*IE Density in DIC*)[18].

Coordinating IE = Coordinating Designs * IE Density in DIC

(70)

It is important to remember that the process of coordinating designs, while required to fix an integration error, does not in and of itself fix the error. Coordinating designs simply involves two designers, one from each coupled component involved in the integration error, determining the root cause of the integration error and the necessary rework of one or both of the designs. By convention in the NPD model, the component design to which the integration error is attached will always require rework. With some fixed probability the coupled design will also require rework. It is not until the required rework (whether of one or both designs) is complete that the integration error is fixed. The formulation for ultimately fixing integration errors will be discussed in the next section.

Integration Errors in Designs in Rework

The final set of integration errors in the co-flow structure is the stock of integration errors that are associated with the designs in the queue for redesign (*Designs in Rework*). The stock of *IE in RW* and its associated flows is depicted in Figure 113.

[18] Nominally, every design that is in coordination must have an integration error and this density should remain equal to 1. Indeed, in the simulation it does. The inclusion of this term in the formulation is done for both consistency and robustness.

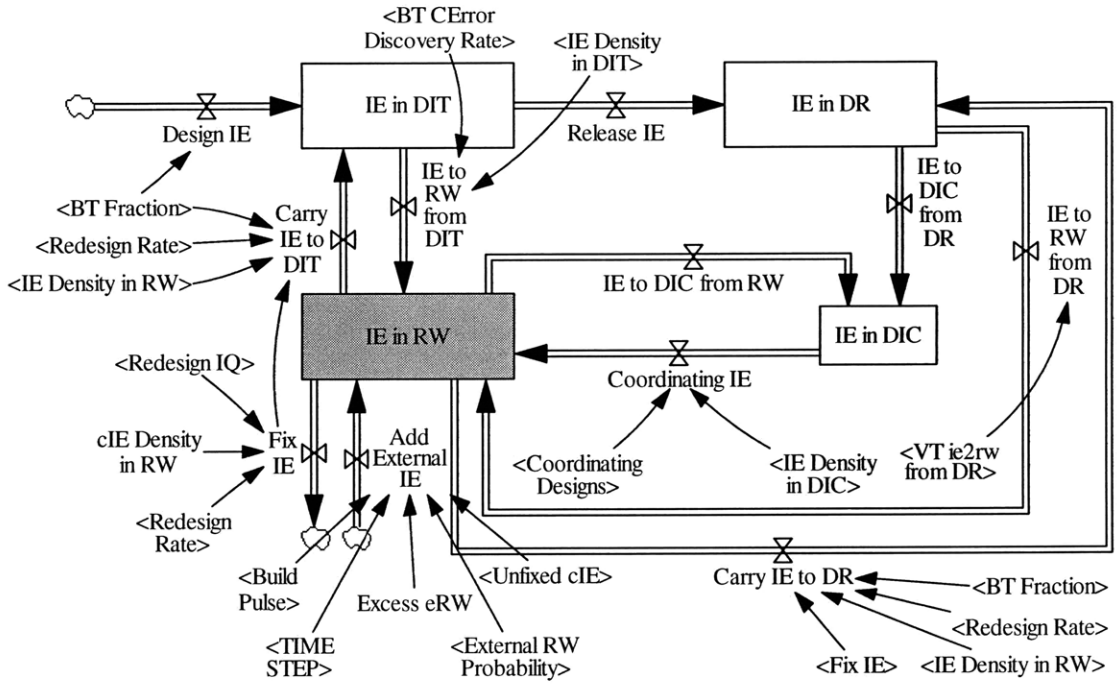


Figure 113 Integration Errors in Designs in Rework (IE in RW)

There are four inflows to the stock of *IE in RW – IE to RW from DIT, IE to RW from DR*, *Coordinating IE*, and *Add External IE*. Each of these inflows, except the last (*Add External IE*) has a corresponding flow in the component error co-flow structure. There are also four outflows to this stock – *Fix IE, IE to DIC from RW*, *Carry IE to DIT,* and *Carry IE to DR* – each of which has a corresponding flow in the co-flow structure for component errors. The equation for *IE in RW* is as follows:

IE in RW = INTEG(IE to RW from DIT - Carry IE to DIT - Carry IE to DR + IE to RW from DR + Coordinating IE - Fix IE - IE to DIC from RW + Add External IE , 0)

(71)

Of the four inflows to the stock of *IE in RW*, each flow except the last (*Add External IE*) is analogous to a corresponding flow in the component error co-flow. The formulation of the equations for these flows and a discussion of each have been provided above. On the other hand, the last inflow – *Add External IE* – does not have an analogous flow in the component error co-flow and requires a fair bit of explanation.

The last flow, *Add External IE*, is the rate at which integration errors are added to the stock of *IE in RW* and represents the transfer of an integration error from one design in a coupled design pair to the other. This occurs when the coordination of an integration error requires both of the coupled designs to conduct rework and the design with the integration error attached to it completes its redesign and fixes its share of the of the integration error but the coupled design does not complete its rework. In this case the integration error is not considered fixed until both designs have completed their rework. Therefore, if the design with the attached integration error completes its required rework the integration error will still exist if the coupled component design has not yet completed its rework - in which case, the model transfers the error to the incomplete design awaiting rework. For accounting purposes in the model, this transfer of integration errors between coupled designs occurs at the time of vehicle builds.

Add External IE[a] = MAX (Excess eRW[a] - Unfixed cIE[b] * External RW Probability , 0) / TIME STEP * Build Pulse

(72)

Add External IE[b] = MAX (Excess eRW[b] - Unfixed cIE[a] * External RW Probability , 0) / TIME STEP * Build Pulse

(73)

Excess eRW = MAX (External RW - Indicated eRW , 0)

(74)

The variable *Excess eRW* in equations (72) through (74) represents the number of designs in external rework that are over and above that which would be expected based on the number designs the coupled component has left in rework that have integration errors that have been coordinated. The number of excess designs that are in this situation can be determined by subtracting the indicated amount of external rework (*Indicated eRW*) from the stock of external rework (*External RW*) remaining for a given design. As indicated in equation (74), the amount of excess work is restricted to be a positive amount.

Indicated eRW[a] = Coordinated IE in RW[b] * External RW Probability

(75)

Indicated eRW[b] = Coordinated IE in RW[a] * External RW Probability

(76)

External RW Probability = 0.5

(77)

The indicated external rework is the amount of external rework a given design should expect to have based on the number of coordinated designs that its coupled component still has in rework (*Coordinated IE in RW*) and the probability of a coordinated design requiring rework from both coupled designs (*External RW Probability*). In the NPD model, the *External RW Probability* is set at 0.5 meaning that half of the integration errors that are coordinated with require both designs from the coupled "design pair" to conduct rework to fix the integration error. The formulation for

the stocks of *External RW* and *Coordinated IE in RW* and their associated flows will be discussed in more detail below.

A component that has more external rework than what is indicated has not completed its share of required redesigns based on previously coordinated integration errors. Therefore, the integration errors associated with the excess external rework will not have been fixed. In this case it is appropriate to attach these integration errors to the slower component (i.e. the component with excess external rework) and designate them as new uncoordinated integration errors in the stock of IE in RW. However, not all of the excess external rework will be transferred to the slower component as new integration errors because the faster component (the one that completed its share of required redesigns) may not have fixed the integration errors attached to it during redesign. Remember that the integration quality of redesign is not perfect and some integration errors, even those that have been coordinated, will not be fixed in redesign and thus will be carried over. In the event this happens, the integration error is still attached to the original component and need not be transferred to the slower component. In equations (72) and (73) above, these unfixed errors are referred to as *Unfixed cIE* because in this case we are only concerned about coordinated integration errors, which drive external rework, that weren't fixed during redesign. The formulation of *Unfixed cIE* will be discussed below. Not all of these unfixed errors are associated with external rework. Indeed we would expect only half of the unfixed errors to be associated with external rework based on an *External RW Probability* of 0.5. Therefore, we subtract half of the unfixed errors from the excess amount of external rework before adding them to the stock of *IE in RW*. Again the transfer of these integration errors between coupled components happens at the time of each build and we use the *MAX* function in VENSIM to ensure that we don't add a

negative amount of integration errors through this process in the event that there are more unfixed errors associated with external rework than the number of excess designs remaining in external rework to be dumped. In equations (72) and (73), dividing the resulting value of the term *MAX (Excess eRW – Unfixed cIE *External RW Probability)* by *TIME STEP* and multiplying it by the *Build Pulse* ensures that the requisite integration errors are all dumped (i.e. transferred) in a single time step at the time of each build event.

It should be noted that these "new" integration errors are added to the stock of *IE in RW* but they are not added to the stock of *Coordinated IE in RW* which will be discussed below. Because the NPD model assumes that integration errors must be coordinated in order to be fixed, by not adding these integration errors to the stock of *Coordinated IE in RW* effectively means that they can not be fixed unless and until they are discovered in subsequent vehicle testing and coordinated again.

Lastly, a component that has less external rework than what is indicated has done more than its share of required redesigns (as compared to the rework done by its coupled component) and therefore will not have any integration errors transferred to it. This feature of the formulation is accomplished using the MAX function in VENSIM which chooses the maximum of two values as in the *MAX (External RW – Indicated eRW, 0)* term in equation (74).

External Rework

The stock of external rework represents the set of designs in rework that require redesign because of an integration error that exists between the given designs and a corresponding set of designs from a coupled component. The rework is classified as external because it is identified in

the process of coordinating the required rework to fix an integration error that is attached to a design from a coupled component and not the given component. The stock of External RW and its associated flows is depicted in Figure 114.

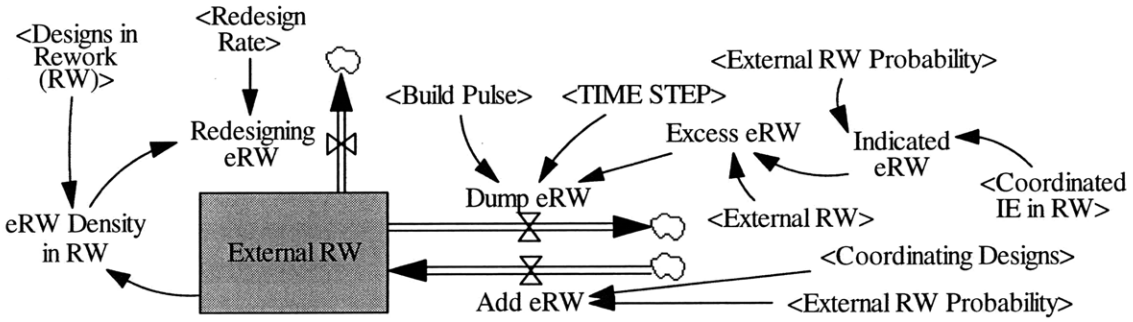


Figure 114 External Rework (External RW)

There is one flow into the stock of *External RW – Add eRW* and two outflows – *Redesigning eRW* and *Dump eRW*. The differential equation describing the state of the stock of External RW follows:

External RW = INTEG(Add eRW - Dump eRW - Redesigning eRW , 0)

(78)

The inflow, *Add eRW*, represents the addition of external rework to the stock of *External RW* for a given component as designs from the OTHER (i.e. coupled) component that have integrations errors are coordinated. The formulation ensures that designs are added to the stock of *External RW* as the coupled designs from the other component are coordinated (*Coordinating Designs*) with the

probability that both designs will require rework (*External RW Probability*). As such, a separate equation is required for each component.

Add eRW[a] = Coordinating Designs[b] * External RW Probability

(79)

Add eRW[b] = Coordinating Designs[a] * External RW Probability

(80)

The first outflow, *Redesigning eRW*, represents taking out designs considered to be external rework from the stock as they are redesigned. The formulation in equation (81) is straightforward ensuring that designs are removed as they are reworked (*Redesign Rate*) in proportion to the fraction of designs in rework that can be considered as external rework (*eRW Density in RW*). This formulation assumes that designers choose the designs to rework in a random fashion.

Redesigning eRW = Redesign Rate * eRW Density in RW

(81)

eRW Density in RW = ZIDZ (External RW , Designs in Rework (RW))

(82)

Assuming randomness in redesign work means that a designer for a given component doesn't distinguish between designs in the stock of rework when choosing which design to redesign next. For example, a designer would not necessarily choose to rework a design with only a component or an integration error before choosing one with both types of errors. The likelihood of a designer reworking a design with a particular error profile (CE, IE or both) is proportional to the densities of

designs with the given error profiles in the stock of *Designs in Rework*. In the case of external rework, randomness in redesign also means that the designer is no more likely to choose to rework a design designated as external rework as he is to choose a design that is not except as supported by the density of external rework in the stock of designs in rework (*eRW Density in RW*).

The second outflow, *Dump eRW*, is part of the process of transferring integration errors between coupled components as outlined above. In addition to the excess external rework designs that are added as new integration errors to the stock of *IE in RW*, it is necessary to remove all of the excess external rework from the stock of external rework. This is true because the balance of the excess rework that was not added to the stock of *IE in RW* is associated with redesigned but unfixed integration errors in the coupled component. Therefore, the design's original designation as external rework affiliated with a coordinated integration error is no longer valid. Mind you, the design itself will remain in the stock of rework and thus may be redesigned somewhat unnecessarily. This notion of unnecessary rework is the basis for some of the phantom work captured by the model and will be discussed in more detail.

The formulation for the outflow *Dump eRW* is straightforward in that it takes the excess external rework (*Excess eRW*) which was described earlier and dumps it in its entirety at the time of each build event.

Dump eRW = (Excess eRW / TIME STEP) * Build Pulse

(83)

Unfixed Coordinated Integration Errors

The stock of unfixed coordinated integration errors (*Unfixed cIE*) represents the set of coordinated integration errors whose associated designs have been redesigned since the last build event where the integration error was not fixed. As such the integration error was carried over. In order to track these unfixed errors, the following stock and flow structure is used:

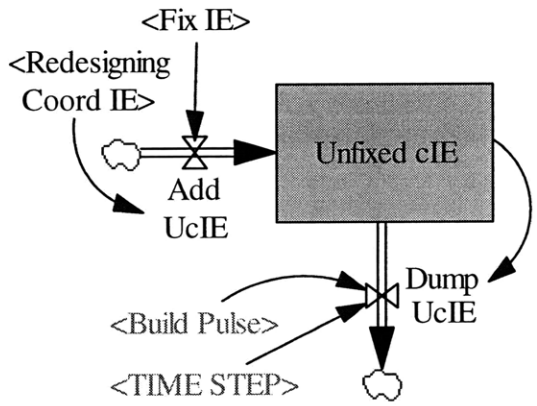


Figure 115 Unfixed Coordinated Integration Errors (Unfixed cIE)

There is a single stock representing coordinated integration errors which have been redesigned but not fixed since the last build event. There is one inflow which adds to the stock of *Unfixed cIE* as coordinated integration errors are redesigned but not fixed. There is one outflow that empties the stock of *Unfixed cIE* at the time of each build event. The model equations are included below.

Unfixed cIE = INTEG(Add UcIE - Dump UcIE , 0)

(84)

Add UcIE = Redesigning Coord IE - Fix IE

(85)

Dump UcIE = (Unfixed cIE / TIME STEP) * Build Pulse

(86)

Coordinated Integration Errors in Rework

The stock of coordinated integration errors in the designs in rework (*Coordinated IE in RW*) represents the set of designs awaiting rework that have associated integration errors that have been coordinated. Similar to the stock of known component errors in the stock of designs in rework (*Known CE in RW*), it is necessary to account for the stock of *Coordinated IE in RW* in order to determine the rate at which integration errors can be fixed through redesign. This is based on the assumption that integration errors that have not been coordinated can not be fixed. Additionally, it is necessary to keep track of these coordinated integration errors to determine the rate of transfer of integration errors between coupled designs as outlined above.

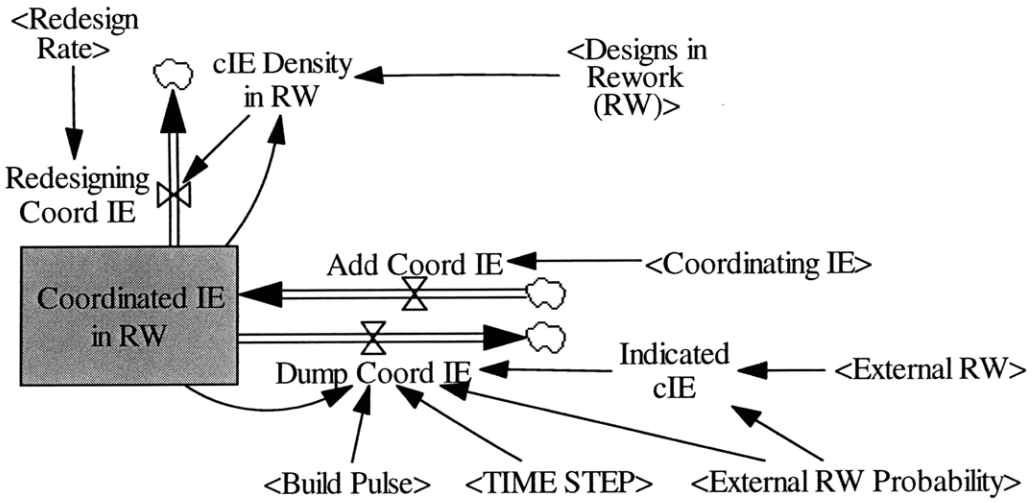


Figure 116 Coordinated IE in RW

There is one inflow to the stock of *Coordinated IE in RW – Add Coord IE* – and two outflows – *Redesigning Coord IE* and *Dump Coord IE*. The differential equation for the stock of Coordinated IE in RW is:

Coordinated IE in RW = INTEG(Add Coord IE - Redesigning Coord IE - Dump Coord IE , 0)

(87)

The inflow, *Add Coord IE*, represents the addition of integration errors to the stock as designs with integration errors are coordinated and sent for rework. The rate at which integration errors are added to the stock is determined by the rate of *Coordinating IE* which was discussed previously.

Add Coord IE = Coordinating IE

(88)

The first outflow, *Redesigning Coord IE*, represents taking out integration errors from the stock of *Coordinated IE in RW* as designs with coordinated integration errors are redesigned. The formulation in equation (89) is straightforward ensuring that integration errors are removed as designs are reworked (*Redesign Rate*) in proportion to the fraction of designs in RW that have coordinated integration errors associated with them (*cIE Density in RW*).

Redesigning Coord IE = Redesign Rate * cIE Density in RW

(89)

cIE Density in RW = ZIDZ (Coordinated IE in RW , Designs in Rework (RW))

(90)

The second outflow, *Dump Coord IE*, is formulated in a manner similar to that of the flow *Dump eRW* discussed in the previous section. It compares the number of coordinated integration errors in rework for a given component to the indicated number of coordinated integration errors it should have based on the amount of external rework its coupled component has yet to complete.

Dump Coord IE = (MAX ((Coordinated IE in RW - Indicated cIE) * External RW Probability , 0) / TIME STEP) * Build Pulse

(91)

Indicated cIE[a] = ZIDZ (External RW[b] , External RW Probability)

(92)

Indicated cIE[b] = ZIDZ (External RW[a] , External RW Probability)

(93)

The variable *Indicated cIE* for each component can be determined by dividing the amount of external rework remaining (*External RW*) for the OTHER component by the fixed probability of both designs requiring rework upon coordination of an integration error (*External RW Probability*). For example, if component B has 10 designs left in its stock of *External RW*, given an *External RW Probability* of 0.5 we would expect component A to have 20 integration errors left in its stock of coordinated integration errors in rework (*Coordinated IE in RW*) that correspond to the incomplete designs in external rework for component B. If component A has more than 20, than the excess coordinated integration errors may correspond to completed external rework from component B. Indeed, we would expect that the excess coordinated IE would correspond to the completed external rework from component B with a probability of 0.5 - the same probability of a coordinated design requiring both component designs to conduct rework (*External RW Probability*) - assuming that component A completes its redesign work randomly.

Assuming randomness in redesign work means that a designer for a given component doesn't distinguish between designs in the stock of rework when choosing a design for rework. For example, a designer would not necessarily choose to rework a design with only a component or an integration error before choosing one with both types of errors. The likelihood of a designer reworking a design with a particular error profile (CE, IE or both) is proportional to the densities of designs with the given profiles in the stock of designs in rework. In the case of integration errors, randomness in redesign also means that the designer is no more likely to choose to rework a design that has a coordinated integration error requiring rework from both coupled components than a design with a coordinated IE only requiring rework from one component, except as supported by the densities of each type of integration error in the stock of designs in rework.

Using the example above, if component A's stock of *Coordinated IE in RW* is at 30 designs we would expect that 5 designs (half of the 10 designs in excess of the indicated 20 coordinated integration errors) would correspond to completed external rework from component B.

If at the time of a build one component has made the required changes to its design to fix the integration error and the other has not as described above, the result is a "new" integration error that must be discovered through subsequent vehicle testing and require additional coordination. As such, the excess coordinated integration errors are dumped from the stock of *Coordinated IE in RW* at the time of each build lowering the density of coordinated integration errors in rework. This effectively means that the designs associated with the dumped integration errors will not be fixed in redesign unless and until they are discovered and coordinated again.

BUILD SECTOR

Build Sector Overview

The build sector of the NPD model is structured to account for the current status of the set of build parts for a new product development project during vehicle testing. This sector includes the set of parts (also referred to as build parts) which correspond to the designs that make up each component of the new product under development. While the paper designs modeled in the design sector reflect the latest version (revolution) of the design for each given part, the parts in the build sector reflect the state of the designs at various points in time – i.e. at the time of each build event. While the set of designs continue to evolve, the parts for each build are like a snapshot in time – they don't change during vehicle testing. In addition to the set of parts, the build sector also

accounts for the component errors and the integration errors associated with these parts using a co-flow structure as in the design sector. Figure 117 provides a high level depiction of the build sector of the NPD model.

The structure of the build sector is fairly straightforward. There is a single stock of *Build Parts* and its associated flows. There are two additional stocks, one for component errors (*Build CE*) and another for integration errors (*Build IE*) that are associated with the build parts, along with their associated flows. Just as is the case of the design sector, the build sector is subscripted for the different components, which means there is a different set of build parts and its associated errors for both component A and B. Additionally, since there may be multiple build events during the course of a new product development project, the build sector is also subscripted for the different builds. This means that there are different stocks of parts and errors not only for each component but for each build as well.

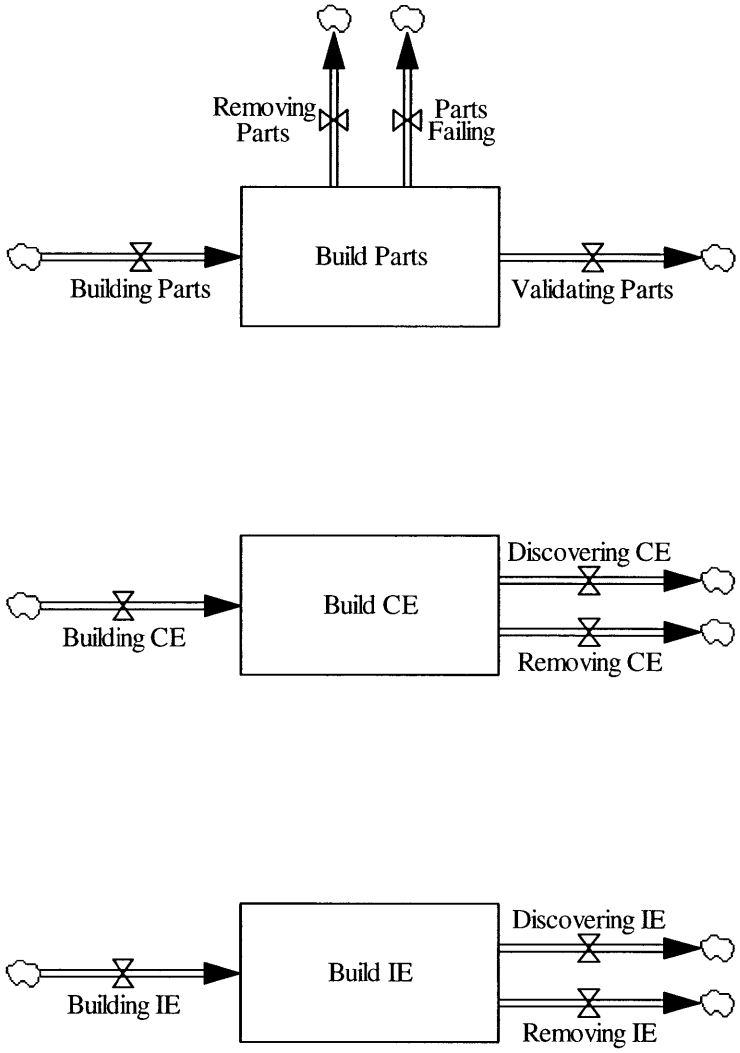


Figure 117 Build Sector

In the build sector, each stock has a single inflow that represents the building of parts and their corresponding errors. Each stock has an outflow that represents the discovery of errors through vehicle testing. In the stock of build parts this is referred to as *Parts Failing* while in the error stocks it is referred to as *Discovering CE/IE.* Additionally, there is an outflow to the stock of *Build Parts* which validates parts as they are tested and found to be error free. There is no corresponding outflow to *Validating Parts* from the error stocks because vehicle testing is assumed to be perfect

(i.e. it finds all true errors and no false ones) and therefore it wouldn't make sense to "validate" an error.

Finally, there is also an outflow from each of the stocks that represents the removal of build parts and errors. Parts and their associated errors are removed from the stocks for a given build when the same part with the same error(s) fails vehicle testing in another build. This can occur because it is possible (and indeed common in the client company) to overlap build events and subsequent vehicle testing. This means that the vehicle testing from a given build is still ongoing when the next build event occurs and the testing of the subsequent version of the vehicle begins.

The NPD model by convention allows no more than two vehicle builds and testing to be ongoing simultaneously. This reflects the true state of vehicle builds and testing in the client company. While it is common to have two successive versions (revolutions) of a full vehicle in testing at the same time, historically the time between builds is sufficiently long enough that vehicle testing is completed for a given build before the second successive build occurs. For example, vehicle testing on Version 1 is normally completed before Version 3 is built. Additionally, on those rare occasions when vehicle testing is not completed as described, vehicle testing is cut off on the version of the vehicle from two generations ago. For example, when Version 3 is built the vehicle testing on Version 1 would be cut off or any future test results (i.e. errors discovered) from Version 1 would be ignored.

As mentioned, throughout the course of a new product development project there may be multiple build events. The NPD model allows for this by using subscripting in the build sector. In addition to the stocks, flows and auxiliary variables in the build sector being subscripted for the

different components (A and B), they are also subscripted for the different builds. In the NPD model, the basic builds include a Prototype build (referred to as *Proto*), a Design Intent build (*DIB*), and a First Production Event build (*FPE*). While these are the pre-planned builds, the model also allows for additional builds to be scheduled as necessary. This often occurs in the client company if a given build does not go well (i.e. too many errors are discovered). These additional builds are referred to in the model as Proto2, Proto3...DIB2, DIB3...and FPE2, FPE3...

While the simulation model accounts for the different builds using subscripting, it is necessary for the model to be able to distinguish between the builds in terms of their generation or vehicle version. As mentioned, only two vehicle builds and testing are allowed to be ongoing (or active) at any given time. Of these two vehicle builds, it is necessary to distinguish between the current build (i.e. the latest version of vehicle parts and errors) and the previous build (the prior generation or version of vehicle parts). While a description of how the model handles this distinction will be reserved for the technical documentation, the use of this convention will be necessary in the sections that follow.

Build Parts

The main stock of concern in the build sector is that of the *Build Parts*. These parts correspond to the set of designs in the main stock and flow chain in the design sector. At the time of a build event (for example, the prototype build) a set of parts that reflect the state of the designs for each component is built. These parts are then assembled into a full vehicle and vehicle testing begins. While the build process includes assembly into a full vehicle, in the model each component maintains its own separate stock of *Build Parts* for each build event. Through the process of vehicle testing, component and/or integration errors are discovered in the set of build parts and

they fail testing. Other parts are found to be without error and are thus validated. As parts fail in vehicle testing, their corresponding designs are sent for coordination and/or rework depending on their error profile.

The stock of *Build Parts* and its associate inflow and outflows is depicted in Figure 118. The differential equation for the stock of *Build Parts* and the formulation for each of the flows are included below.

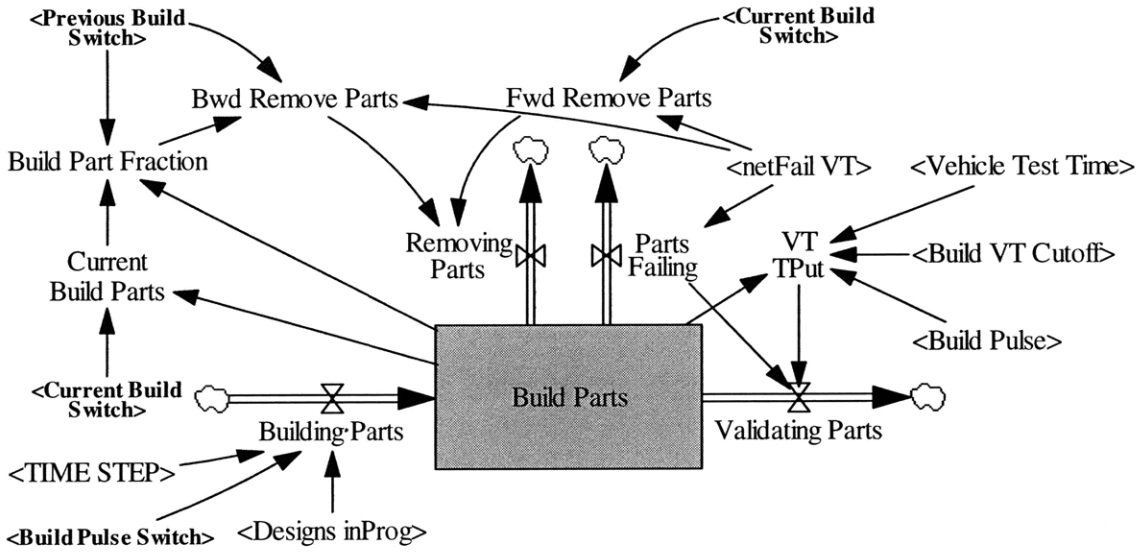


Figure 118 Build Parts

There is one inflow – *Building Parts* – and three outflows – *Parts Failing, Validating Parts,* and *Removing Parts* – to the stock of *Build Parts*. The differential equation describing the state of the stock at any given point in time is as follows:

Build Parts[Component,Build] = INTEG(Building Parts[Component,Build] - Parts Failing [Component,Build] - Removing Parts[Component,Build] - Validating Parts[Component,Build] , 0)

(94)

From equation (94) you can see that the stock of *Build Parts* at any given point in time is the integral of the inflows minus the outflows given an initial value of zero. Indeed, each of the stocks (*Build Parts*, *Build CE*, and *Build IE*) has an initial value of zero for each of the components and each of the builds. You'll notice that the variables in the equation for *Build Parts* are subscripted for the different components and builds as indicated by the use of **[Component, Build]** after each. This is true of nearly all of the variables in the build sector. In the discussion and equations that follows, the subscripts will be left out except as necessary for clarification.

Building Parts is the only inflow to the stock of build parts. It represents the process of converting designs into parts and assembling the parts into full vehicles in preparation for vehicle testing. For the purposes of this model, this process is greatly simplified and it is assumed that the parts and vehicles are built in a single simulation time step at the time of a build event.

Building Parts = Build Pulse Switch * Designs inProg / TIME STEP

(95)

Designs inProg = Designs in Rework (RW) + Designs in Test (DIT) + Designs Released (DR) + Designs in Coordination (DIC)

(96)

As formulated, the flow of building parts takes the set of designs that are in progress for a given component (*Designs inProg*) and, by dividing by *TIME STEP*, adds them all to the stock of

Build Parts in a single time step when the *Build Pulse Switch* equals 1. The *Build Pulse Switch* is a dimensionless switch which pulses on (i.e. equals 1) at the instant when a given build event occurs and it stays on for a single simulation time step. The set of parts corresponding to the designs in progress are built for all components at the time of each build with each component maintaining a separate stock of build parts and the associated error co-flows.

The two major outflows for the stock of Build Parts are *Parts Failing* and *Validating Parts*. These outflows are the result of vehicle testing which is modeled as a simple first order delay where vehicle testing throughput (*VT TPut*) represents the rate at which build parts are tested.

Parts Failing = netFail VT

(97)

Equation (97) shows the formulation for the flow of *Parts Failing*. It is based on the variable *netFail VT* which is an auxiliary variable representing the net rate at which build parts fail in vehicle testing because a component error, an integration error or both were discovered in a given part. The formulation for this variable will be discussed in detail in the vehicle testing section of this chapter.

Validating Parts = VT TPut - Parts Failing

(98)

VT TPut = (Build Parts / Vehicle Test Time) * Build VT Cutoff* (1 - Build Pulse)

(99)

Build VT Cutoff = IF THEN ELSE (Build Parts < 0.005, 0, 1) * Build Switch

(100)

The rate of validating parts is simply the difference between the rate at which build parts are tested (*VT TPut*) and the rate at which parts fail (*Parts Failing*) as outlined in equation (98). Again, this formulation is based on the assumption that vehicle testing is perfect. If a part does not fail then it is validated. Like bench testing in the design sector, vehicle testing is modeled as a simple first order delay where *Vehicle Test Time*, which may be different for the different components, represents the average time it takes to conduct vehicle testing on a part for a given component. Therefore, the rate at which build parts are tested (*VT TPut*) can be formulated by taking the number of build parts remaining and dividing it by the average test time as in equation (99). You'll notice in the equation that this rate is multiplied by the variable *Build VT Cutoff* which is a switch used to cut off vehicle testing when the stock of build parts falls below a minimum threshold. The cutoff switch also ensures that vehicle testing only occurs on active builds via the variable *Build Switch.* There will be a section included in the technical documentation that describes the formulation for these and the other switches used in the model. Just as with the rates of flow in the design sector, the last term in equation for *VT TPut, (1-Build Pulse)*, is used to pause vehicle testing at the instant in time when a build event occurs.

The third outflow to the stock of *Build Parts* is *Removing Parts.* When a part fails in vehicle testing during any build, if the same part with the same errors exists in another active build, it is removed from that build. Removing the failing parts (and their associated errors as you will see in the co-flow description later in this chapter) prevents the model from reacting twice to the same error. This is akin to a test engineer finding a duplicate error in testing and, recognizing it as such, not writing it up as a new error but a repeat of a previously identified error.

Removing Parts = Bwd Remove Parts + Fwd Remove Parts

(101)

There are two mechanisms by which removing parts takes place in the NPD model. The first is called forward removing parts in the model (*Fwd Remove Parts*) and refers to the case where parts are removed from the stock of build parts for a given build, when it is considered to be the current build, because identical parts fail in a previously ongoing build. The second is called backward removing parts (*Bwd Remove Parts*) and refers to the case where parts are removed from the stock of build parts for a given build, when it is considered to be the previous build, because identical parts fail in a more recent (i.e. the current) build. The formulation for each of these mechanisms will be discussed below.

Fwd Remove Parts[Component,Build] = (SUM (netFail VT[Component,Build!]) - netFail VT[Component,Build]) * Current Build Switch[Build]

(102)

The first of the two mechanisms by which the removal of parts from the stock of build parts occurs is forward removing parts. In short, this is the process of removing a part from the current set of build parts when the same part fails testing on a previous version of the vehicle. The formulation of the rate at which these parts are removed is shown in equation (102). The first term on the right hand side of the equation, *SUM (netFail VT[Component, Build!])* represents the sum of the parts failing across all vehicle builds.[19] From this sum, the equation removes the rate of failing parts of the given build as indicated by the term *netFail VT[Component, Build]*. This leaves

[19] The SUM function in VENSIM calculates the sum across a vector of variables. In this case, the vector is the set of builds as indicated by the exclamation point "!" following the subscript Build in equation (102).

the rate at which parts are failing in the OTHER active build. By multiplying this difference by the *Current Build Switch*, which is only turned on when the given build is the most recent build, equation (102) results in the forward removal of parts for a given build only when it is the current build and at the rate at which parts are failing in the previous build.

Essentially this formulation means that every part that fails in ongoing testing from a previous build will be removed as a duplicate error in the current build. This assumes that every part with an error that remains in testing on a previous version of the vehicle will also reside in the current version of the vehicle. This assumption is possible because of the way that errors are handled in the co-flow structure. As you will see, both component and integration errors are forward and backward removed. Additionally, as parts are redesigned in the design sector their associated errors are also removed from the pre-existing stocks of errors in build parts. While removing these errors means that they won't be discovered in the model as they would be on the actual vehicle, this is akin to a designer ignoring errors found on parts that have already been redesigned. Ultimately, the handling of errors in the co-flow structure ensures that every exiting error in an ongoing previous build also exists in the current build which allows failing parts (and newly discovered errors) from the previous build to be removed as duplicates from the current build.

Bwd Remove Parts[Component,Build] = (SUM (netFail VT[Component,Build!]) - netFail VT[Component,Build]) * Build Part Fraction[Component,Build] * Previous Build Switch[Build]

(103)

The second mechanism through which parts are removed from the stock of build parts is referred to as forward removing parts. In short, this is the process of removing a part from the

previous set of build parts (the older version) when the same part fails testing on the current version of the vehicle. The formulation of the rate at which these parts are backward removed is shown in equation (103). The formulation is very similar to that of forward removing parts found in equation (102). The first part of the equation *(SUM (netFail VT [Component, Build!]) - netFail VT[Component, Build])* is duplicated from equation (102) and calculates for a given build the rate of parts failing in the other active build. In this case, however, the term is multiplied by the variable Previous Build Switch which is turned on only when the given build is considered the older of the two active builds. This results in the backward removal of parts for a given build only when it is the previous (or older) build.

Build Part Fraction[Component,Build] = ZIDZ (Build Parts[Component,Build] , SUM (Current Build Parts[Component,Build!])) * Previous Build Switch[Build]

(104)

Unlike forward removal of parts, this backward removal does not occur at the same rate at which parts are failing in the current build. This is true because not all of the parts and their associated errors in the current build exist in the remaining parts and errors in the ongoing previous build. Undoubtedly some new parts have been designed since the previous build and many more have been redesigned all with the possibility of introducing new errors. Therefore, it is necessary to calculate the fraction of parts in the current build that are the same as those from the previous build. The variable *Build Part Fraction* as formulated in equation (104) represents this fraction. As discussed earlier, we know that all of the remaining parts in the previous build are duplicates of the parts in the current build. Therefore, this fraction can be calculated simply by taking the number of build parts for a given build and dividing it by the current build parts. Multiplying by

the *Previous Build Switch* ensures that the *Build Part Fraction* is zero except when the given build is the previous build.[20]

Current Build Parts= Build Parts* Current Build Switch

(105)

The variable *Current Build Parts* is equal to the set of build parts that is associated with the current build and is easily calculated by using the *Current Build Switch* to designate the build parts from a given build as the Current Build Parts when indeed the given build is designated as the current build. While there is only one set of *Current Build Parts* at any one time, the use of the summation function in equation (104) is necessary in the model to allow the capture the value of the *Current Build Parts* across any of the subscripted builds.

Parts Error Co-Flow Overview

Much like the co-flow structure used in the design sector, the build sector uses two co-flows to track the errors (component and integration) associated with the set of build parts. These co-flow structures are depicted below the stock and flows for the set of build parts in Figure 117. There are two stocks, one each for the component and integration errors associated with the stock of build parts. There is a corresponding inflow in both of the co-flow structures to the one inflow of *Building Parts* to the main stock of *Build Parts*. There are outflows of discovering component and integration errors that correspond to failing parts. There are also outflows for removing component and integration errors that correspond to the removal of parts as described earlier. As mentioned, there are no outflows in the component and integration error co-flows that correspond to the

[20] Without the use of the *Previous Build Switch* in equation (104), when the given build is also the current build, the *Build Parts Fraction* would be equal to 1.

outflow of *Validating Parts* from the main stock of Build Parts. The formulation for each of the corresponding stocks and flows in the co-flow structure is nearly identical to that of the main stock and flow chain for *Build Parts*. Where appropriate I will minimize the description of formulations that have been previously discussed.

Component Errors in Build Parts

The stock of component errors in build parts (*Build CE*) maintains the component errors that are associated with the stock of build parts discussed previously. There is one inflow – *Building CE* – which adds component errors to the stock at the time of a build event. There are two outflows – *Discovering CE* and *Removing CE* – that correspond to the outflows of *Parts Failing* and *Removing Parts* from the stock of *Build Parts*. A depiction of the co-flow for component errors in build parts can be found in Figure 119.

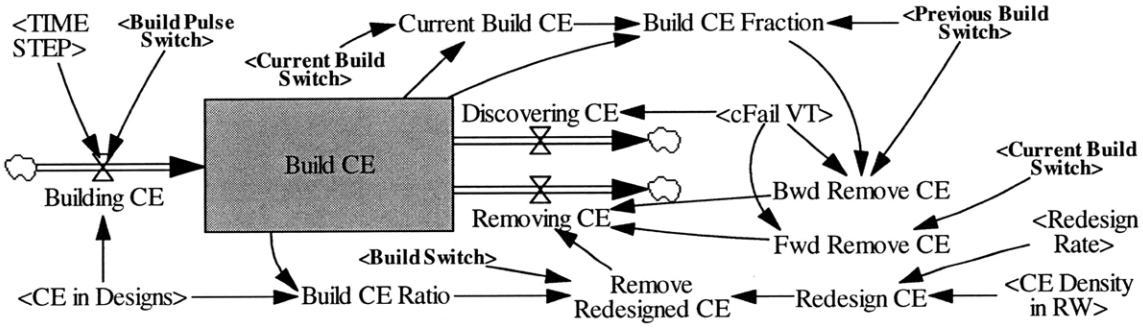


Figure 119 Component Errors in Build Parts (Build CE)

The differential equation describing the state of the stock of Build CE is as follows:

Build CE = INTEG(Building CE - Discovering CE - Removing CE, 0)

(106)

The formulation for the inflow *Building CE* is analogous to that of Building Parts except that instead of building the set of designs in progress the inflow adds the number of component errors that are associated with those designs (*CE in Designs*). Again, these errors are added in a single simulation time step at the time of the build event.

Building CE = Build Pulse Switch * CE in Designs / TIME STEP

(107)

CE in Designs = CE in DIT + CE in RW + CE in DR + CE in DIC

(108)

The outflow *Discovering CE* represents the rate at which component errors are removed from the stock of *Build CE* for a given build as they are identified in vehicle testing. The formulation is straightforward in that it relies on the rate of discovering component errors in vehicle testing (*cFail VT*) which will be discussed in further detail in the Vehicle Testing section of this chapter.

Discovering CE = cFail VT

(109)

The outflow *Removing CE* represents the rate at which component errors are removed from the stock of *Build CE* in vehicle testing from a given build because they have been discovered in an ongoing previous build, subsequent build or the part with which the error is associated has been redesigned. While the formulation for *Removing CE* is analogous to *Removing Parts* as it both forward removes and backward removes component errors, it varies in that it also removes component errors when the design for the associated part is redesigned.

Removing CE = Fwd Remove CE + Bwd Remove CE + Remove Redesigned CE

(110)

Forward removing component errors (*Fwd Remove CE*) represents the rate at which component errors are removed from the stock of *Build CE* for a given build (when it is considered to be the current build) because identical component errors have been identified in an ongoing previous build. The formulation is completely analogous to that of *FWD Remove Parts* in equation (102) above. Again, the subscripts are shown in equation (111) for clarity.

Fwd Remove CE[Component,Build] = (SUM (cFail VT[Component,Build!]) - cFail VT [Component,Build]) * Current Build Switch[Build]

(111)

Backward removing component errors (*Bwd Remove CE*) represents the rate at which component errors are removed from the stock of *Build CE* for a given build (when it is considered to be the previous build) because identical component errors have been identified in a more recent (i.e. the current) ongoing build. Here, too, the formulation is completely analogous to the corresponding removal of parts (*Bwd Remove Parts*) discussed in equations (103) and (104) above.

Bwd Remove CE= (SUM(cFail VT) - cFail VT) * Build CE Fraction * Previous Build Switch

(112)

Build CE Fraction = ZIDZ (Build CE, SUM(Current Build CE)) * Previous Build Switch

(113)

Removing redesigned component errors (*Remove Redesigned CE*) represents the rate at which component errors are removed from the stock of *Build CE* during vehicle testing because the part with which the error is associated has been redesigned since the build event occurred.. Because the design / part has been redesigned since the beginning of vehicle testing, the component error may or may not have been fixed during the course of its redesign. Thus, the component error associated with the previous version of the part will be ignored when it is discovered in vehicle testing. In order to account for this in the model, the original component errors associated with parts prior to their redesign parts are removed from the stock of *Build CE* and thus will not be discovered.

Remove Redesigned CE = Redesign CE * Build CE Ratio * Build Switch

(114)

Redesign CE = Redesign Rate * CE Density in RW

(115)

Build CE Ratio[Component,Build] = ZIDZ (Build CE[Component,Build] , CE in Designs[Component])

(116)

Because only component errors that are known by the designer can be fixed in redesign, only known component errors that are redesigned are removed from the stock of *Build CE*. This is accomplished in the formulation for *Remove Redesigned CE* by multiplying the redesign rate by the density of known component errors in rework (*kCE Density in RW*) in equation (114).

However, not all of the component errors that are redesigned should be removed from the stock of *Build CE* because not all of the component errors that exist in the current set of designs were

necessarily included in the build event. Remember, while the set of designs continues to evolve throughout the course of the new product development project the build parts and their associated errors represent a snapshot in time and do not change except from build to build. For example: One might imagine a part that was designed or redesigned after the build event occurred and vehicle testing began. In the course of designing or redesigning the part a component error may have been introduced that is subsequently discovered in bench testing and sent for rework. In this case, the component error would not have been included as part of the stock of component errors in build parts (*Build CE*) since the error wasn't generated until after the build event occurred. Therefore, this error shouldn't be removed from the stock of *Build CE* as its associated design is redesigned. The formulation for removing redesigned component errors handles this likelihood by multiplying the rate of redesigning component errors (*Redesign CE*) by the variable *Build CE Ratio* which maintains the fraction of component errors in the current set of designs that exist in a given build (*Build CE*). As designs with component errors are redesigned, the probability that they are component errors that are also in the stock of *Build CE* is equal to this fraction and thus component errors should be removed proportionally.

Integration Errors in Build Parts

The stock of integration errors in build parts (*Build IE*) maintains the set of integration errors that are associated with the stock of build parts discussed previously. Completely analogous to the stock of component errors in build parts (*Build CE*), there is one inflow – *Building IE* – which adds component errors to the stock at the time of a build event and there are two outflows – *Discovering IE* and *Removing IE* – that correspond to the outflows of *Parts Failing* and *Removing Parts* from

the main stock of *Build Parts*. Figure 120 is a depiction of the co-flow structure for integration errors in build parts.

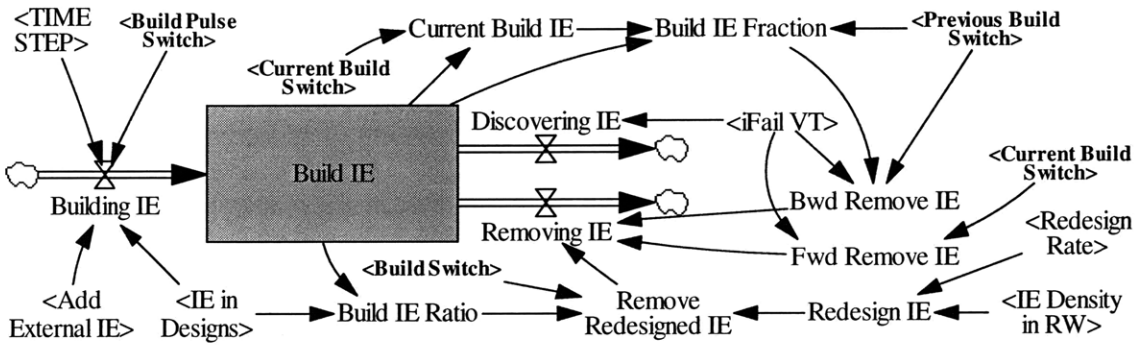


Figure 120 Integration Errors in Build Parts (Build IE)

The differential equation for the stock of *Build IE* can be found in below. Again it is completely analogous to the formulation for *Build CE*. Indeed, with the exception of the formulation for the one inflow, *Building IE*, the formulation and equations describing the stock and flow structure here are analogous to those of from the stock of *Build CE* discussed in the previous section.

Build IE = INTEG(Building IE - Discovering IE - Removing IE, 0)

(117)

As mentioned, the formulation for *Building IE* is slightly different from that of *Building CE* in equation (107). In addition to building the integration errors that exist in the designs in progress (*IE in Designs*) at the time of the build event, in the case of integration errors it is also necessary to include the "new" integration errors that are created from the imbalance in external rework

discussed previously (*Add External IE*). Doing so allows these new integration errors to be discovered in vehicle testing.

Building IE= Build Pulse Switch * (IE in Designs / TIME STEP + Add External IE)

(118)

The formulations for the outflows *Discovering IE* and *Removing IE* (both forward and backward removal) are the same as those for the component error co-flow and so they won't be repeated here. The equations for these flows can be found in the technical documentation in Appendix B.

VEHICLE TESTING SECTOR

Vehicle Testing Overview

Vehicle testing is the process by which errors are discovered in full vehicles after they have been assembled as part of a build event. Vehicle testing typically refers to the process of rigging up a vehicle with the necessary test equipment and then operating it in a variety of operational conditions and recording any failures or concerns that might reflect an error in the design. Often times the vehicle is ridden on a test track and miles are accumulated in an effort to determine where and when components may break down. This is generally referred to as high mileage testing. However, many errors are found up front in the process of building the vehicle itself. For example, it is not uncommon to find two components that don't quite fit together properly during the assembly process. This would be an example of an integration error as defined in the NPD

model. For the purposes of this model, vehicle testing will include the testing done and errors discovered in both the build event and the subsequent road testing.

Unlike bench testing, which is assumed to only be able to find component errors, vehicle testing is formulated to be able to discover both component and integration errors. Additionally, for simplification purposes and without loss of generalization, vehicle testing is considered to be perfect in that as vehicle parts are tested any associated errors will be discovered.

The vehicle testing sector of the NPD model consists of the vehicle testing itself – which discovers the errors in build parts – and the necessary accounting to determine how to move the designs and their associated errors between the various stages of design (i.e. the stocks of *Designs in Test*, *Designs Released*, *Designs in Coordination* and *Designs in Rework*). Each will be discussed in turn in the sections that follow.

Discovering Errors

The main driver behind the rate at which errors are discovered in vehicle testing is the throughput rate at which build parts are tested. The variable *VT TPut* as formulated in equation (99) determines the rate at which parts are tested. As mentioned earlier, vehicle testing is modeled as a simple first order delay where *Vehicle Test Time*, represents the average time it takes to conduct vehicle testing on a part for a given component. The average test times may be different for the different components which can result in one component identifying its errors sooner than the other giving its designers more time to do root cause analysis and redesign as necessary.

The rate at which build parts are tested (*VT TPut*) can be formulated by taking the number of build parts remaining and dividing it by the average test time as in equation (99). The use of a first order delay as the driver of vehicle testing results in a rate of vehicle testing for each component with profiles like those in Figure 121. In this simple case, both components begin testing at time 0 with 1000 build parts and component A has an average vehicle test time of 3 months while component B's test time is set at 6 months.

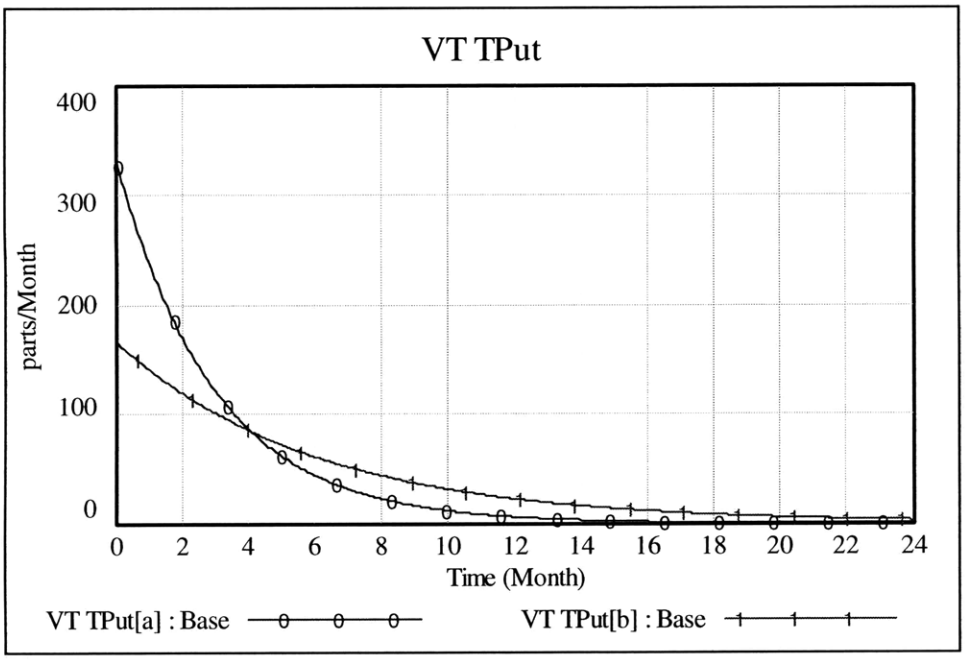


Figure 121 Vehicle Test Throughput

As you can see in the graph of *VT TPut*, the result for both components is a high throughput rate early in the vehicle testing that decays exponentially over time. The practical result is that many parts are tested early in vehicle testing and, as you will see, their associated errors are discovered early as well. However, some parts aren't tested for a much longer time which delays

discovery of their associated errors. This testing profile, in particular the resulting error discovery profile, reflects the testing results experienced in the client company.

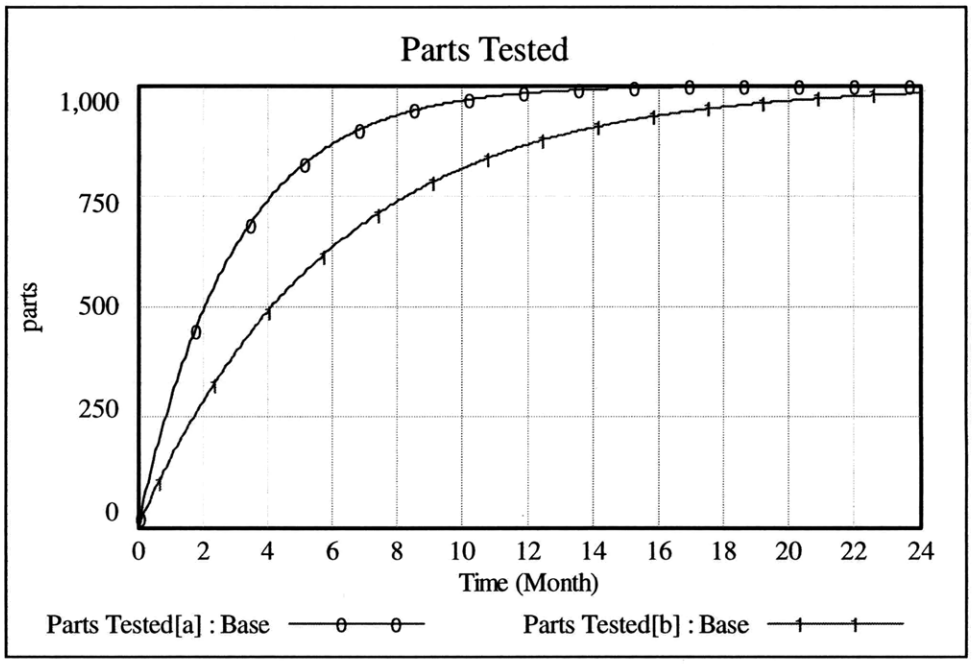


Figure 122 Parts Tested

Integrating the rate of vehicle testing (VT TPut) captures the number of parts tested cumulatively over time and perhaps gives a more intuitive sense for the rate at which parts are tested. Some parts only require a short time to be tested and their associated errors discovered. Others take a much longer time. Using a previous example, an integration error that causes two parts to not fit together properly will be discovered during the assembly process as part of the build event itself. A vibration problem that causes undue stress on the vehicle frame over time resulting in a fracture might not be discovered until thousands of miles have been accumulated on the test track.

The use of a first order delay structure for vehicle testing also assumes that the build parts and their associated errors are perfectly mixed. The likelihood of a given part being tested next is no more likely than any other. Also, the parts are assumed to have component and/or integration errors in proportion to the density of component and integration errors in the co-flow structure from the build sector. As a result, as a given part is randomly selected from the stock of build parts for testing it can have a component error, an integration error or both. For accounting purposes in the model, it is necessary to be able to distinguish between each of these cases because designs that correspond to parts with these different error profiles will be handled differently by designers in real life and by the model as well. For example, a part that is found to have a component error only will immediately be sent for rework (i.e. redesign). However, a part that is found to have both a component error and an integration error will either be sent directly for rework or for coordination of the integration error first.

Figure 123 shows a schematic showing the relationship between the significant variables used in the model for the discovery of errors in vehicle testing. Each of the variables and their respective equations will be discussed below.

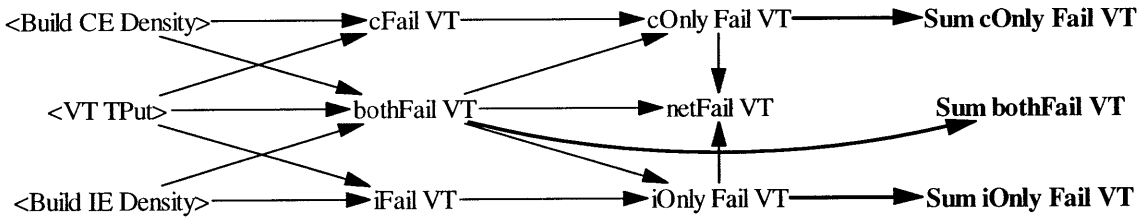


Figure 123 Error Discovery in Vehicle Testing

In addition to the rate at which build parts are being testing (VT TPut) which was discussed above, the other main driver for discovering errors through vehicle testing in the NPD model is the density of the two error types in the build parts. *Build CE Density* and *Build IE Density*, respectively, represent the fraction of build parts with component and integration errors. The formulation for both of these densities has been discussed and their equations can be found in the technical documentation. As mentioned, the formulation for determining the rate at which parts fail for having a particular kind of error is simply to multiply the rate at which parts are being tested (*VT TPut*) by the density of errors in the stock of build parts as outlined in equations (119) and (120) for component failures (*cFail VT*) and integration errors (*iFail VT*) respectively. You'll note that these equations are subscripted for both the different components and the different builds as each component's parts are vehicle tested separately in the model and they are tested during multiple builds. Again, the build events are common across the components.

cFail VT[Component,Build] = VT TPut[Component,Build] * Build CE Density[Component,Build]

(119)

iFail VT[Component,Build] = VT TPut[Component,Build] * Build IE Density[Component,Build]

(120)

The formulation for determining the rate at which parts fail for having both types of errors (*bothFail VT*) is simply the rate at which they are tested multiplied by the joint probability of them having a component and an integration error.

bothFail VT[Component,Build] = VT TPut[Component,Build] * Build CE Density [Component,Build] * Build IE Density[Component,Build]

(121)

From these failure rates, we can determine the rate at which parts fail for having only a component error or only an integration error (*cOnly Fail VT* and *iOnly Fail VT*, respectively). This is done by subtracting the rate at which parts fail for having both types of errors from the rates at which they fail for having a particular type of error as outlined in equations (122) and (123). Additionally, we can determine the net failure rate which represents the rate at which build parts fail for having a component error, an integration error or both. This is done by adding the rate at which parts fail for having both types of error (*bothFail VT*) with the rates of a part failing for having only one type of error just discussed.

cOnly Fail VT[Component,Build] = cFail VT[Component,Build] - bothFail VT[Component,Build]

(122)

iOnly Fail VT[Component,Build] = iFail VT[Component,Build] - bothFail VT[Component,Build]

(123)

netFail VT[Component,Build] = cOnly Fail VT[Component,Build] + iOnly Fail VT [Component,Build] + bothFail VT[Component,Build]

(124)

The failure rates calculated in equations (122) through (124) are what drive the outflows of parts and errors from their respective stocks in the build sector. However, it is also necessary to combine the vehicle testing results across the different builds. Because there can be as many as two different builds (generations or revolutions of the vehicle) and vehicle testing overlapping, it is

necessary to sum up the failure rates from vehicle testing from each build before taking appropriate action in the design sector (e.g. sending a design for coordination and/or rework). This is accomplished using the SUM function in VENSIM as indicated in equations (125) through (127). Again, the exclamation point after *Build* in each of these equations indicates that the summation is occurring across builds.

Sum cOnly Fail VT[Component] = SUM (cOnly Fail VT[Component,Build!])

(125)

Sum iOnly Fail VT[Component] = SUM (iOnly Fail VT[Component,Build!])

(126)

Sum bothFail VT[Component] = SUM (bothFail VT[Component,Build!])

(127)

In adding together the failure rates as outlined above, we can reasonably assume that the same part with the same errors will not be found at the same time in two different builds and thus added together as if they were different parts and/or different errors. We can assume this because of the forward and backward removing of parts and errors between builds as described in the build sector section of this chapter and because we use a sufficiently small time step (*dt*) in the simulation model. In the NPD model, the simulation *Time Step* is set at .0625 of the time unit which is months.

Vehicle Test Accounting

Once we have determined the failure rates for build parts, we must determine how to handle the designs and their newly discovered errors. There are two major components to the framework for handling failed parts in the NPD model. The first is to model the process used to decide what should be done with a design whose part fails in vehicle testing. The second component is determining where the designs corresponding to the failed parts reside (i.e. in which stock of designs from the design sector) at the time of the error discovery. Based on these two processes, the model can then move the designs and their associated errors to either the stocks for rework or coordination as appropriate.

The decision process used to decide what should be done with a design whose part fails in vehicle testing is fairly straightforward. If a part fails that has a component error only, then its corresponding design and the associated component error are sent directly to rework. If a part fails that has an integration error only then its design and the error are sent for coordination prior to being sent for rework. A part that fails for having both a component error and an integration error will either have its design and associated errors sent directly to rework or they will be sent for coordination first. This decision is based on the designers' willingness to delay rework on a known component error until they have a chance to coordinate the required rework for a known integration error. This willingness to delay rework on a design until the integration error has been coordinated is reflected in the variable *Coordination Fraction* which will be discussed in more detail below. Essentially, this variable determines the fraction of designs with both types of errors that are sent for coordination first upon discovery.

The second component to the framework for handling designs with newly discovered errors is to determine where they reside at the time of error discovery. This is necessary because design and redesign work continue during vehicle testing. This means that designs and their associated errors can and do move between the various stocks in the design sector while a static representation of those designs are tested in the vehicle testing sector of the model. While there are 4 stocks of designs that have errors associated with them in the design sector – *Designs in Test*, *Designs Released*, *Designs in Rework* and *Designs in Coordination*, we know that errors found in parts in vehicle testing will not correspond to designs (and their associated errors) that reside in the stock of *Designs in Test (DIT)*. This is true for three reasons. First, at the time of a given build event all of the designs and their associated errors that are in the stock of DIT are dumped into the stock of Designs Released. Secondly, all designs that are redesigned and sent for bench testing (i.e. added to the stock of *Designs in Test*) have their associated errors removed from the stocks of *Build CE* and *Build IE* as described in the build sector. Therefore, none of these errors would be discovered in vehicle testing Lastly, the only other way for a design and its associated errors to move into the stock of DIT is through new design work which would have occurred after the build event Therefore, none of the errors discovered in vehicle testing would correspond to these new designs.

The result of the previous discussion is that we know that errors discovered in vehicle testing must correspond to designs (and their associated errors) that are in the stocks of *Designs Released*, *Designs in Coordination* or *Designs in Rework*. As designs with certain error profiles fail vehicle testing, we pull the designs from the respective stocks just mentioned in proportion to the number of designs with the same error profile that reside in each stock. For example, as parts fail vehicle testing because they have a component error only (*Sum COnly Fail VT*), we would pull designs

from the stock of *Designs Released* in proportion to the number of designs with component errors only that are in the stock released designs *(% CEo in DR)*.

Given the failure rates for parts, the location of designs corresponding to the failing parts, and the decision regarding where to send a design that has failed in vehicle testing, we can formulate the flows of designs and their associated errors between the various stages of development due to failures in vehicle testing. In the following sections, we will use the formulation described above as a basis for discussing the movement of component errors, integration errors and designs in conjunction with vehicle testing.

Component Errors

When a part fails vehicle testing because it has a component error discovered, we need to know how to move the component error that is associated with the design that corresponds to the failed part. As mentioned, there are three stages of development (stocks) that the corresponding design (and thus its associated component error) could be at the time the part fails – *Designs Released*, *Designs in Coordination* and *Designs in Rework.* Additionally, there are two places we could send a failed design and its associated component error – *Designs in Coordination* or *Designs in Rework.* By convention in the NPD model, designs that are in the stock of *Designs in Coordination* will stay in that queue until they are coordinated. This means that even if the failing part has both an integration error and a component error, the corresponding design will not be pulled from the stock of DIC and sent directly for rework. In other words, the original decision to send the design for coordination, perhaps from a previous build, will stand.

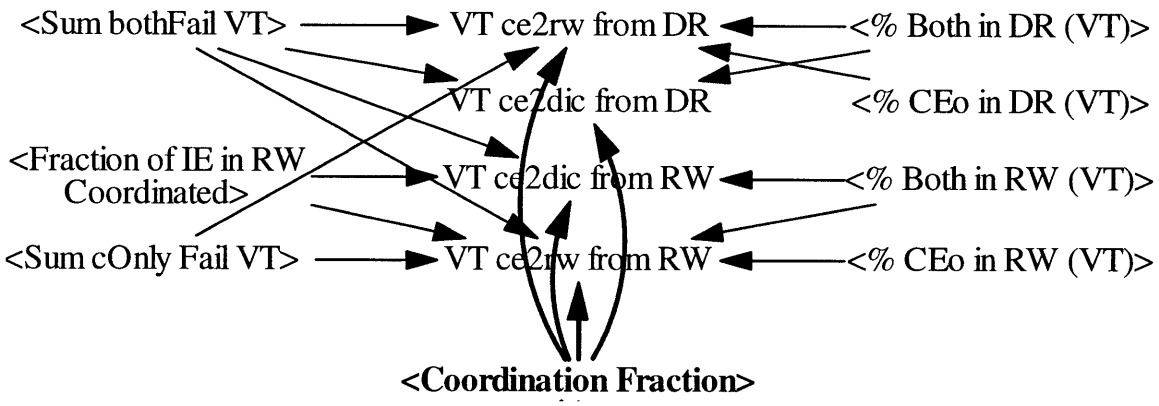


Figure 124 Component Error Accounting in Vehicle Testing

Ultimately, this means that we have to determine the rates of flow of component errors from the stocks of *Designs Released (DR)* and *Designs in Rework (RW)* to the stocks of *Designs in Coordination (DIC)* and *Designs in Rework (RW).* These rates of flow correspond to the variables down the middle of Figure 124. Using the first of these variables, *VT ce2rw from DR*, as an example, the naming convention for a particular variable in vehicle test accounting is as follows:

VT ce2rw from DR

1. The "VT" indicates that the flow is a result of **v**ehicle **t**esting.

2. "ce2rw" means that the flow consists of component errors (**ce**) being sent to (**2**) the stock of rework (**rw**).

3. Lastly, "from DR" indicates that these component errors are being taken **from** the stock of component errors in designs released (CE in **DR**)

The formulation for this variable and the other variables controlling the rates of flow for component errors will be discussed next.

VT ce2dic from DR = Sum bothFail VT * % Both in DR (VT) * Coordination Fraction

(128)

% Both in DR (VT) = ZIDZ (Both Errors in DR , Total BothErrors (VT))

(129)

The variable VT ce2DIC from DR represents the rate at which component errors in designs released (*CE in DR*) are identified through vehicle testing and designated to move with their associated designs to the stock of *CE in DIC*. Because we would only move a design to the stock of *DIC* if it has both types of errors, we only move component errors to the stock of *CE in DIC* when it is associated with a design whose part fails vehicle testing for having both types of errors. We calculate the rate by taking the rate of parts failing in any active build for having both types of errors (*Sum bothFail VT*) and multiplying it by the fraction of designs that have both types of errors in the stock of released designs (*% Both in DR (VT)*). Furthermore, we need to multiply this rate by the fraction of designs with both types of errors that designers choose to send for coordination prior to rework (*Coordination Fraction*). The fraction of designs in the stock of released designs that have both types of errors is simply the number of designs having both types of errors in the stock of released designs (*Both Errors in DR)* divided by the total number of designs having both errors for vehicle testing purposes (*Total BothErrors (VT)*) as in equation (129). Counting designs (and their associated errors) for vehicle testing purposes means that we only consider those designs in the stocks of *Designs Released (DR), Designs in Coordination (DIC),* and *Designs in Rework (RW).*

VT ce2rw from DR= Sum cOnly Fail VT * % CEo in DR (VT) + Sum bothFail VT * % Both in DR (VT) * (1 - Coordination Fraction)

(130)

% CEo in DR (VT) = ZIDZ (CEonly Errors in DR , Total CEo (VT))

(131)

VT ce2rw from DR represents the rate at which component errors in designs released (*CE in DR*) are identified through vehicle testing and designated to move with their associated designs to the stock of *CE in RW*. Since this flow sends component errors (along with their associated designs) directly for rework, we know that all of those component errors associated with designs failing vehicle testing for having a component error only would fall into this category. The first part of equation (130) captures these component errors and their associated designs by taking the rate of parts failing for component only errors (*Sum cOnly Fail VT*) and multiplying it by the percentage of designs with component errors only that reside in the stock of designs released (*% CEo in DR (VT)*).

In addition to these component errors, we also send some portion (*1- Coordination Fraction*) of those component errors associated with designs released that fail vehicle testing for having both types of errors. This rate is calculated using a formulation similar to that of equation (128) except that it uses the complement of the *Coordination Fraction* (*1-Coordination Fraction*) as a multiplier.

VT ce2dic from RW = Sum bothFail VT * % Both in RW (VT) * (1 - Fraction of IE in RW Coordinated) * Coordination Fraction

(132)

% Both in RW (VT) = zidz (Both Errors in RW , Total BothErrors (VT))

(133)

The variable *VT ce2dic from RW* represents the rate at which component errors associated with designs in rework (*CE in RW*) are identified through vehicle testing and designated to move with

their associated designs to the stock of *CE in DIC*. In this case, formulating the flow involves taking the failure rate of parts with both types of errors (*Sum bothFail VT*) and multiplying it by the percent of designs with both types of errors in rework (*% Both in RW (VT)*) and subsequently by the *Coordination Fraction* similar to what was done in equation (128). However, some of the designs in rework with both types of errors have integration errors that have been previously coordinated but not since redesigned. In this case, we would not need to coordinate the design again. For this reason, it is necessary to multiply the failure rate by the term (*1-Fraction of IE in RW Coordinated*) which represents the fraction of integration errors in the stock of *IE in RW* that have not yet been coordinated.

VT ce2rw from RW = Sum cOnly Fail VT * % CEo in RW (VT) + Sum bothFail VT * % Both in RW (VT) - Sum bothFail VT * % Both in RW (VT) * (1 - Fraction of IE in RW Coordinated) * Coordination Fraction

(134)

% CEo in RW (VT) = ZIDZ (CEonly Errors in RW , Total CEo (VT))

(135)

VT ce2rw from RW represents the rate at which component errors in designs in rework (*CE in RW*) are identified through vehicle testing and designated to remain in the stock of *CE in RW*. Clearly those designs in rework that only have a component error will stay in rework as will their associated component errors. This is represented by the first portion of the formulation in equation (134) which multiplies the component error only failure rate (*Sum cOnly Fail VT*) by the percent of component only errors in rework (*% CEo in RW (VT)*).

The second portion of the formulation for *VT ce2rw from RW* in equation (134) simply takes the rate at which designs in rework fail for having both types of errors (*Sum bothFail VT * % Both in RW (VT))* and subtracts the number of component errors associated with these designs that are sent for coordination (*Sum bothFail VT * % Both in RW (VT) * (1 - Fraction of IE in RW Coordinated) * Coordination Fraction)* as formulated in equation (132).

Even though the variable *VT ce2rw from RW* does not result in any movement, or flow, of errors between stocks it is necessary to the model because it helps determine the rate at which known component errors are added to the stock of *Known CE in RW* which drives the rate at which component errors can be fixed through redesign.

Integration Errors

Just as with component errors, when a part fails vehicle testing because it has an integration error, we need to know how to move the integration error that is associated with the design that corresponds to the failed part. We know from the discussion above, that the only place designs corresponding to failing parts in vehicle testing can reside are the stocks of *Designs Released (DR)*, *Designs in Coordination (DIC)* and *Designs in Rework (RW)* and by convention we will not move

any designs that are already in the queue for coordination at the time its corresponding part fails in vehicle testing. Again there are only two places where we send designs (and their associated integration errors) when their corresponding part fails vehicle testing – the queue for coordination (*DIC*) or directly for rework (*RW*). As such there are four (4) flows of integration errors that we must account for as a result of vehicle testing and they can be found down the middle of Figure 125 Integration Error Accounting in Vehicle Testing.

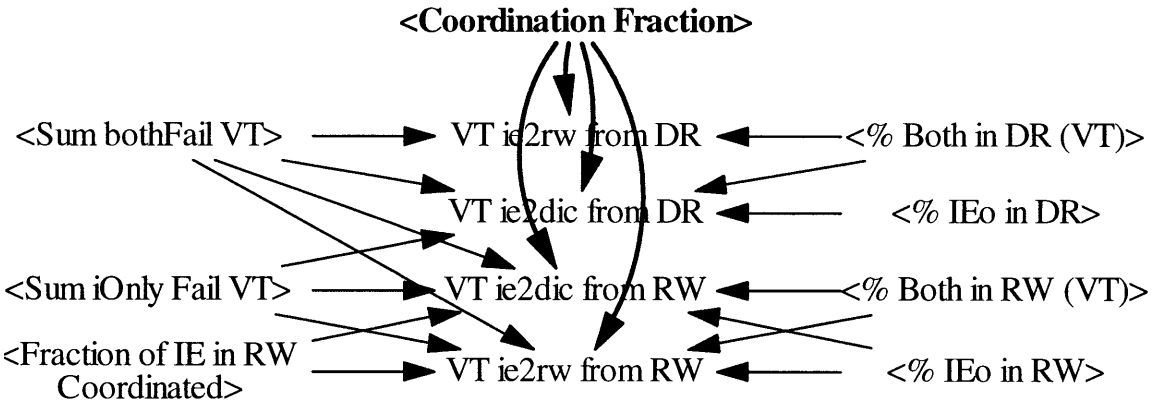


Figure 125 Integration Error Accounting in Vehicle Testing

The formulations for the four flows of integration errors in designs resulting from parts failing in vehicle testing are analogous to the flows of component errors above. As such, the relevant equations are listed below with only brief explanations where required to highlight changes in the formulation.

VT ie2rw from DR = Sum bothFail VT * % Both in DR (VT) * (1 - Coordination Fraction)

(136)

VT ie2rw from DR represents the rate at which integration errors in designs released (*IE in DR*) are identified through vehicle testing and designated to move with their associated designs to the stock of *IE in RW*. It is formulated simply by multiplying the rate at which parts fail for having both types of errors (*Sum bothFail VT*) by the percentage of designs that have both types of errors that are in the stock of designs released (*% Both in DR (VT)*) and multiplying again by the term *(1-Coordination Fraction)* which represents the fraction of designs with both types of errors that will bypass coordination and be sent directly for rework.

VT ie2dic from DR = Sum iOnly Fail VT * % IEo in DR + Sum bothFail VT * % Both in DR (VT) * Coordination Fraction

(137)

% IEo in DR = ZIDZ (IEonly Errors in DR , Total IEo in VT)

(138)

VT ie2dic from DR represents the rate at which integration errors in designs Released (*IE in DR*) are identified through vehicle testing and designated to move with their associated designs to the stock of *IE in DIC*. There are two sources for this rate of flow. First, all released designs whose corresponding parts fail for having an integration error only will be sent for coordination and thus so will their associated integration errors. This is represented by *Sum iOnly Fail VT * % IEo in DR* in equation (137). The second source comes from those released designs whose corresponding parts fail for having both types of errors that are sent for coordination represented by the latter half of equation (137). In this case only some of the designs and their associated integration errors are sent for coordination based on the *Coordination Fraction*.

VT ie2dic from RW = Sum iOnly Fail VT * % IEo in RW * (1 - Fraction of IE in RW Coordinated) + Sum bothFail VT * % Both in RW (VT) * (1 - Fraction of IE in RW Coordinated) * Coordination Fraction

(139)

% IEo in RW = ZIDZ (IEonly Errors in RW , Total IEo in VT)

(140)

VT ie2dic from RW represents the rate at which integration errors in the designs in rework (*IE in RW*) are identified through vehicle testing and designated to move with their associated designs to the stock of *IE in DIC*. The formulation here is analogous to equation (137) – all integration errors from designs whose part fails from having an integration error only will be sent for coordination and some of those failing for having both types of errors will be sent for coordination based on the *Coordination Fraction*. However, an additional term is included in the formulation (*1-Fraction of IE in RW Coordinated*) to ensure that those coordination errors that have already been coordinated but not yet redesigned are not sent for coordination again.

VT ie2rw from RW = Sum iOnly Fail VT * % IEo in RW * Fraction of IE in RW Coordinated + Sum bothFail VT * % Both in RW (VT) - Sum bothFail VT * % Both in RW (VT) * (1 - Fraction of IE in RW Coordinated) * Coordination Fraction

(141)

VT ie2rw from RW represents the rate at which integration errors in designs in rework (*IE in RW*) are identified through vehicle testing and designated to remain in the stock of *IE in RW*. The first source of integration errors remaining in rework are those designs in rework whose corresponding part fails for having an integration error only where the integration error was previously coordinated. This is represented in equation (141) by the term *Sum iOnly Fail VT * % IEo in RW * Fraction of IE in RW Coordinated*. Secondly, there are those designs in rework

whose part fails for having both types of errors where the decision is made to not send the design and its associated integration error for coordination. This can be calculated by taking the rate of these types of failures in the designs in rework and subtract out the designs that are indeed sent for coordination. This calculation is included in the latter half of equation (141).

Designs

The previous two sections outlined the formulation of the variables that control the rate at which component and integration errors are moved between the various stocks, or stages of development, in the co-flows in the design sector as a result of discovering errors in vehicle testing. This section will use these results to formulate the variables necessary to move the designs corresponding to these errors between the stocks in main stock and flow chain in the design sector. A schematic of the relationship between the relevant variables is included in Figure 126.

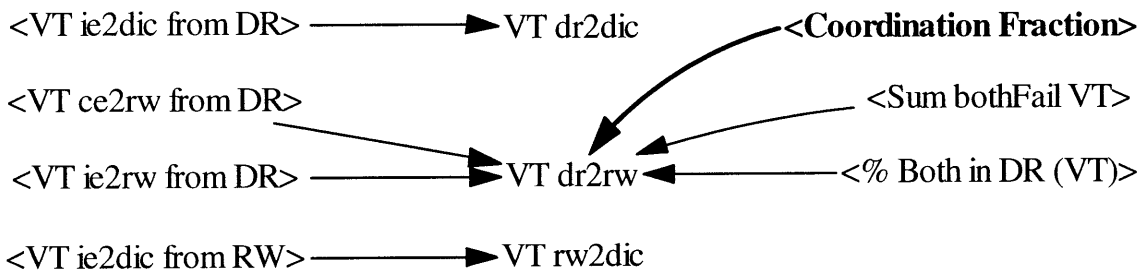


Figure 126 Design Accounting in Vehicle Testing

Just as in the case of discovering and moving errors, there are two places from which we will move designs corresponding to parts failing vehicle testing – *Designs Released (DR)* and *Designs in RW (RW)* – and two places to which we will move them – *Designs in Coordination (DIC)* and *RW*. As a result there are three resulting flows of designs that we will need to calculate as a result

of vehicle testing. The variables that drive these flows are *VT dr2dic*, *VT dr2rw*, and *VT rw2dic*. Their respective equations and a description of their formulations follow.

VT dr2dic represents the rate at which designs released are sent for coordination based on the discovery of integration errors through vehicle testing. Its equation follows:

VT dr2dic = VT ie2dic from DR

(142)

Because only those designs that have an integration error are sent for coordination, the rate at which released designs are sent for coordination is equal to the rate at which integration errors are discovered in vehicle testing and sent to the stock of *IE in DIC* from *IE in DR* as formulated in equation (142).

VT dr2rw represents the rate at which designs released are sent directly to rework based on the discovery of errors (component and/or integration errors) through vehicle testing. Its equation follows:

VT dr2rw = VT ce2rw from DR

(143)

Because a released design will be sent directly for rework only if it has a component error discovered in vehicle testing – either only a component error or in some cases both types of errors – it is easy to determine the rate at which released designs are moved to the stock of *Designs in Rework* based on vehicle testing. It is simply equivalent to the rate at which component errors are

discovered in vehicle testing and sent from the stock of *CE in DR* to the stock of *CE in RW* as formulated in equation (143).

VT rw2dic represents the rate at which designs in rework are sent for coordination based on the discovery of integration errors through vehicle testing. Its equation follows:

VT rw2dic = VT ie2dic from RW

(144)

Because only those designs that have an integration error are sent for coordination, the rate at which designs in rework are sent for coordination is equal to the rate at which integration errors are discovered in vehicle testing and sent to the stock of IE in DIC from IE in RW as formulated in equation (144).

LABOR SECTOR

The labor sector of the NPD model accounts for the number of designers that are available to work on the different activities (design, redesign and coordination) conducted directly by designers in the detailed design phase of a new product development project. As in the execution of a real project, the NPD model provides for the ability to add designers to the project team as work pressure builds. The model also accounts for the effect of adding inexperienced designers to a project.

The basic framework for determining the number of designers available is to calculate the desired number of designers based on the amount of design, redesign, and coordination work to be done and the time remaining until the next deadline (e.g. the next build or product launch) and then, with some delay, add additional designers. In bringing in new designers, the NPD model distinguishes them as "inexperienced" designers with a lower productivity level than experienced designers. Over time, these new designers mature into experienced designers and increase their productivity accordingly. Figure 127 depicts the relationship between the key variables included in the Labor sector of the NPD model.

As in the other sectors of the model, the stocks, flows and auxiliary variables used in the Labor Sector are subscripted to account for the different components (A and B) in the project. In the case of the Labor Sector, this allows the project team to set the number of designers available to work on each component based on the amount of work to be done and the productivity levels of the component designers. The deadlines, however, are project deadlines and are the same for both components.

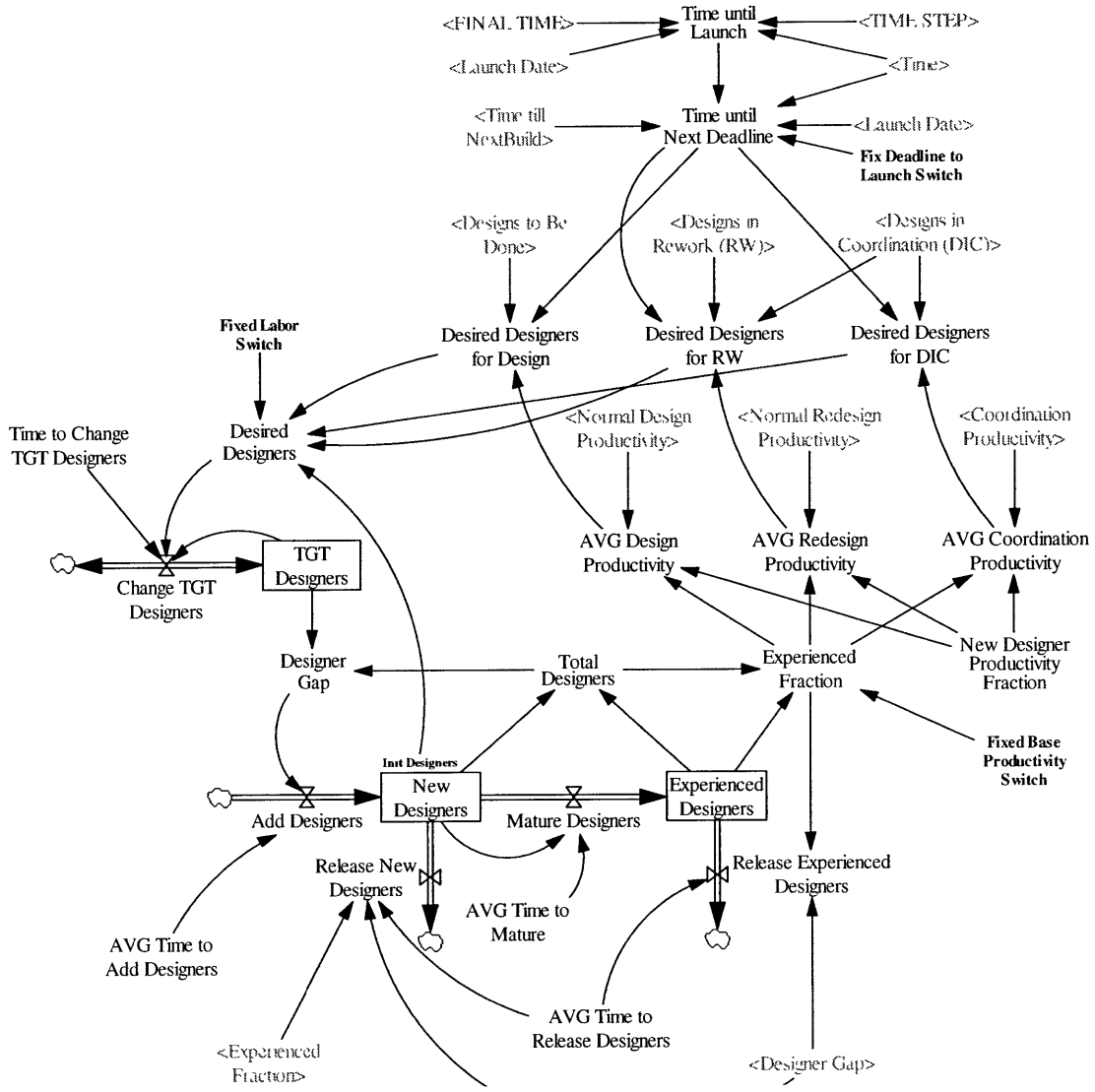


Figure 127. Labor Sector

The Labor Sector can be broken into three main sections. The first section accounts for the number of designers the project team needs (*Desired Designers*) based on the amount of work pressure the project is under. The second section sets the target number of designers that the project team is authorized to have (*TGT Designers*) based on the desired number of designers and some time to change the authorized number. The final section of the Labor Sector determines the

total number of designers available to do project work (*Total Designers*) while accounting for the experience factor of the designers on the project team by adding designers to the team as inexperienced designers (*New Designers*) with lower productivity and, over time, having them mature into *experienced designers*. Each of these sections will be discussed in more detail below. Technical documentation can be found in Appendix B.

Desired Designers

The first section of the Labor Sector determines the number of designers the project team needs (*Desired Designers*) based on the amount of work pressure the project is under. The number of designers that are needed is based on the amount of design, redesign and coordination work that needs to be done before the next perceived hard deadline. The model structure that is used to determine the number of desired designers is depicted in Figure 128.

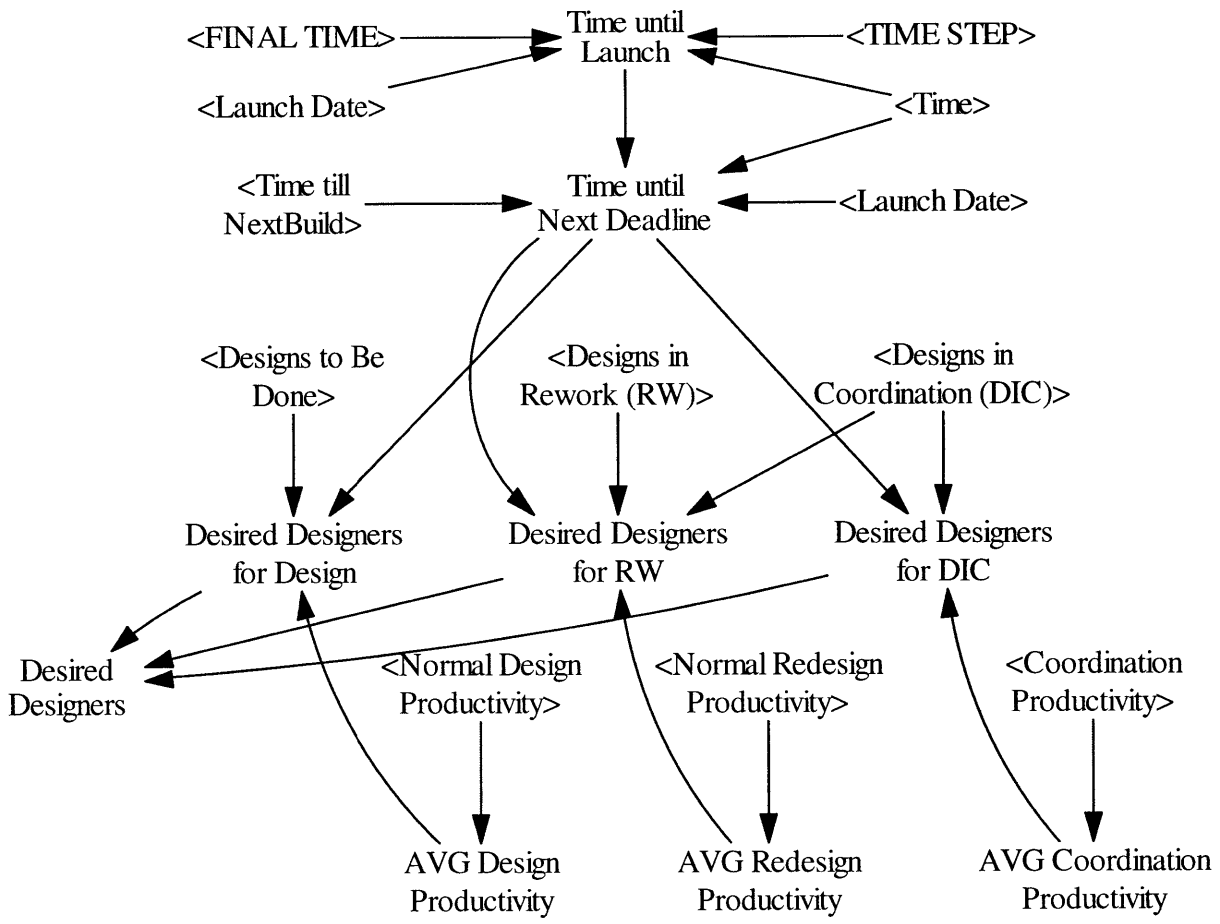


Figure 128 Desired Designers

In order to determine the total number of designers desired for the project team it is necessary to determine the number of designers desired for design, redesign and coordination. Each of these is determined in the same way. For example, the Desired Designers for Design is determined by first dividing the number of designs in the stock of Designs to Be Done by the Time until Next Deadline. This determines the rate at which designs must be completed in order to be finished with the designs by the next deadline. By further dividing this rate by the average design productivity of our designers (AVG Design Productivity) we can determine the number of designers needed in order to complete the designs by the deadline.

The average productivity for designers in each of the activities (design, redesign and coordination) is set at a percentage of the normal productivity based on the fraction of designers who are experienced versus new. This fraction will be discussed in further detail below.

In the base model, the hard deadline (*Time until Next Deadline*) is perceived by the project team to be the scheduled launch date of the product under design. However, in the policy analysis discussed in the next chapter the deadline is alternatively set to the next build event to explore the effects on project performance.

Once the number of designers desired for design (*Desired Designers for Design*), redesign (*Desired Designers for RW*), and coordination (*Desired Designers for DIC*) are determined the total number of *Desired Designers* is easily calculated by summing the three.

Target Designers

The second section of the Labor Sector sets the target number of designers for the project team (*TGT Designers*) for each component (A and B). The target number of designers is simply a smoothed forecast of the desired designers using first-ordered exponential smoothing which eliminates much of the short-term, high frequency "noise" that is inherent in the actual number of *Desired Designers*. The model structure for determining the target number of designers (*TGT Designers*) is shown in Figure 129.

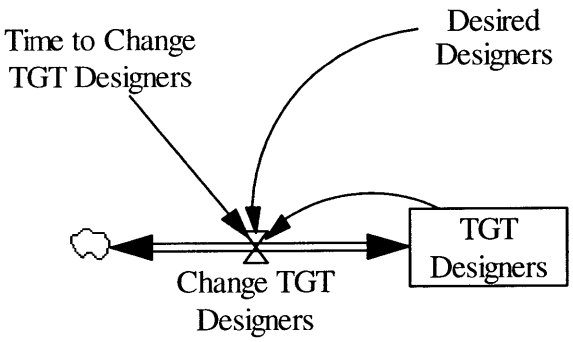


Figure 129 Target Designers

The target number of designers (*TGT Designers*) is represented by a stock which changes based on the difference between its current value and the current value of *Desired Designers*. The amount it changes, represented by the flow *Change TGT Designers*, is tempered by the *Time to Change TGT Designers*. This negative feedback structure, in effect, smoothes the forecast of the number of designers needed by eliminating the short-term variation in *Desired Designers* while still adjusting to the longer-term changes. Given that the actual number of desired designers can vary widely based on the amount of work to be done, the productivity of designers and the amount of time until the next deadline it is important "smooth" this number using adaptive expectations.

Total Designers

The last section of the Labor Sector determines the total number of designers by adding new designers to the project team while accounting for the experience level of all the designers using a simple aging chain structure. This model structure is depicted in Figure 130Figure 130. The result is a means for determining the number of designers (*Total Designers*) available to do project work and their experience level (*Experienced Fraction*) which drives the average productivity of the workforce.

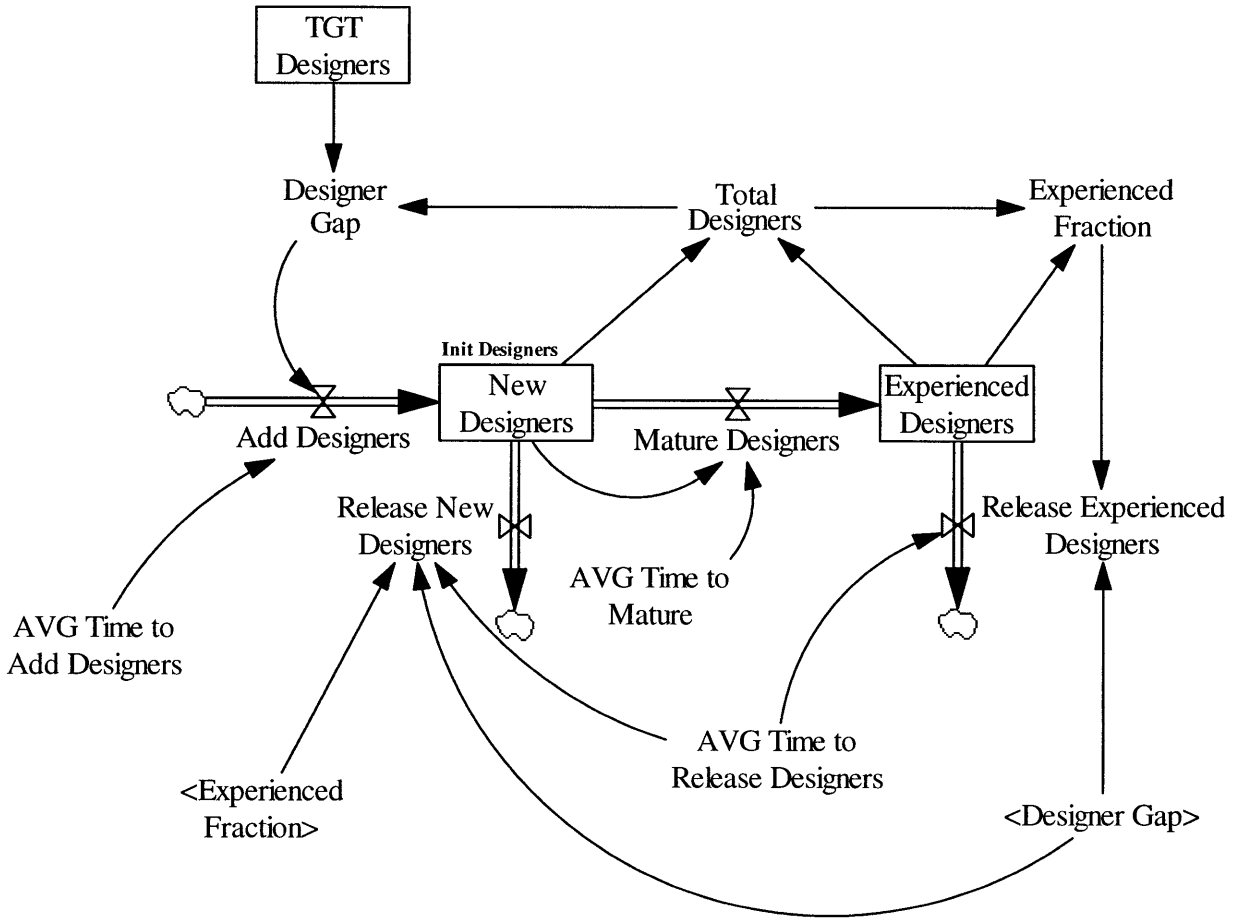


Figure 130 Total Designers

As you can see in Figure 37, this section of the Labor Sector consists of two main stocks - *New Designers* and *Experienced Designers* - which together represent the *Total Designers* available to the project team. Again, these stocks are subscripted by component and therefore they represent the designers who are assigned to work on the specific components (A or B) of the project. The third stock, *TGT Designers*, was previously discussed and represents the target number of designers the project team desires to have available for work.

The stock of *New Designers* is affected by adding new designers (*Add Designers*) to the team, releasing new designers (*Release New Designers*), and designers maturing to become experienced designers (*Mature Designers*). The model adds designers based on difference between the *Total Designers* and *TGT Designers* and adds them to the stock of *New Designers* through the inflow *Add Designers*. The rate at which it closes the gap is tempered by the *AVG Time to Add Designers* which represents the time it takes to find pull designers off of other projects or hire a new designer into the company. Similarly, the model releases designers when there are more designers available than are desired using the *AVG Time to Release Designers* as a means for tempering the outflows. It is assumed that the designers are released proportionally from both stocks (New and Experienced Designers).

New Designers mature into *Experienced Designers* using a simple first-order delay where the *Mature Designers* flow is set by simply dividing the number of designers in the stock of *New Designers* by the *AVG Time to Mature*, which is set at two months. Given the two stocks, it is easy to calculate the total number of designers available (*Total Designers*) and the fraction of the designers which are experienced (*Experienced Fraction*). These numbers are used to determine the average productivity (for design, redesign, and coordination) of the designers where it is assumed that a new designer works at 67% of the rate at which an experienced designer works.

DESIGNER ALLOCATION

This sector of the NPD model allocates designers among the different activities - design, redesign and coordination – conducted directly by designers in the development stage of design

work. While there are certainly other activities that compete for a designer's time, these three represent the major value added engineering activities most prominently discussed by designers in the client organization.

The basic framework for allocating designers is to always allocate the necessary designers to complete new design work first and then to allocate the remaining designers between the redesign and coordination work based on the relative demand for designers for each activity. Figure 131 depicts the relationship between the key variables included in the allocation of designers in the NPD model.

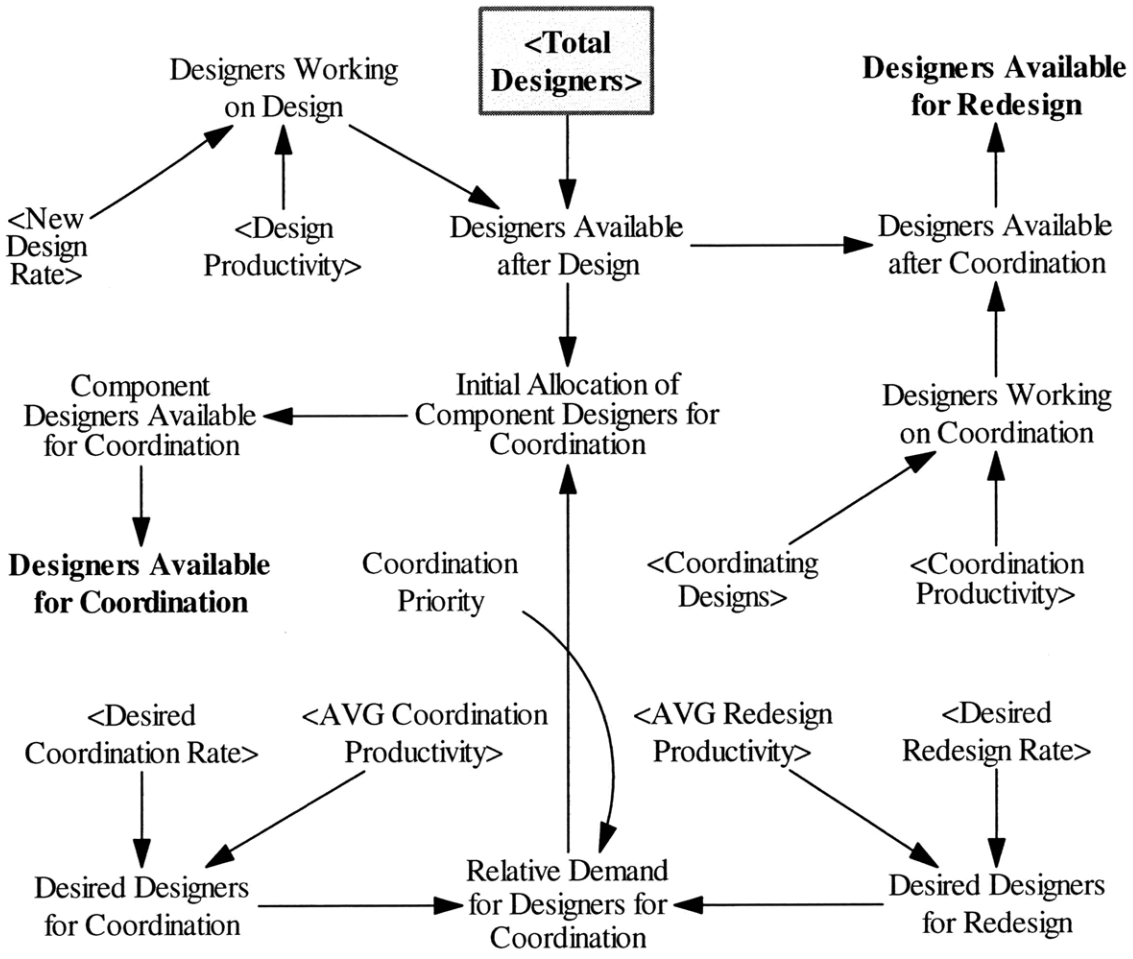


Figure 131 Designer Allocation

The allocation of designers for a given component in the NPD model begins with a given number of designers for each component based on the Labor sector described above. These designers are always allocated to complete new design work first. We can determine the number of designers who are actually working on new designs by dividing the *New Design Rate* by the *Design Productivity* as shown in equation (146). We can then subtract this number from the number of *Total Designers* to determine the number of designers available after new design work.

However, in order to prevent the number of designers available from going negative, we use the MAX function as outlined in equation (145)[21].

Designers Available after Design = MAX (Designers - Designers Working on Design, 0)

(145)

Designers Working on Design = ZIDZ (New Design Rate , Design Productivity)

(146)

New Design Rate is the rate at which new designs are completed as formulated in the design sector of the model. *Design Productivity* is the rate at which a given designer is able to complete new designs. Its formulation will be discussed in the Productivity section of this appendix.

Given the number of *Designers Available after Design*, we need to allocate them between redesign work and coordination. The basic framework for doing so first allocates a number of designers for coordination work based on the relative demand for designers to do coordination versus redesign work. Then, once it is determined how many of these designers are actually working on coordination, the model allocates the remaining designers to do redesign work.

The number of component designers initially allocated for coordination work is calculated by taking the number of designers available after the new design work is accounted for and multiplying it by the relative demand for designers to do coordination as in equation (147). From this value, the model sets the number of *Component Designers Available for Coordination* using

[21] The formulation for designer allocation theoretically ensures that the number of designers available after design for coordination and/or rework is never negative. However, due to limitations of the simulation software in handling values close to zero it is possible to experience extremely small negative values (on the order of $1x10^{-7}$). For this reason, the MAX function is used to protect against the clearly infeasible situation of having negative designers available to do anything.

the IF THEN ELSE formulation in equation (148). This is done to recognize when there is an insufficient number of designers available for coordination – in this case less than one designer working quarter time on coordination – that the model will not allocate any designers for coordination and make them available for other work.

Initial Allocation of Component Designers for Coordination = Designers Available after Design * Relative Demand for Designers for Coordination

(147)

Component Designers Available for Coordination = IF THEN ELSE (Initial Allocation of Component Designers for Coordination < 0.25, 0, Initial Allocation of Component Designers for Coordination)

(148)

The relative demand for designers to do coordination work is determined by comparing this demand to the total demand for designers to do coordination and redesign work as formulated in equation (149). . This is done using an US / (US+THEM) structure. Essentially, this formulation adds together the desired designers for coordination and redesign and then determines what fraction of the total desired designers are desired for coordination.

Relative Demand for Designers for Coordination = ZIDZ (Desired Designers for Coordination * Coordination Priority , Desired Designers for Coordination * Coordination Priority + (1 - Coordination Priority) * Desired Designers for Redesign)

(149)

Coordination Priority = 0.5

(150)

The formulation for *Relative Demand for Designers for Coordination* as listed in equation (149) also allows for a priority to be placed on coordination with respect to redesign. While the

Coordination Priority is set at its default value of 0.5 coordination work and redesign work have equal priority. As the *Coordination Priority* increases above 0.5 toward its maximum value of 1, the weight placed on the demand for designers to do coordination increases resulting in a higher *Relative Demand for Designers for Coordination.* This in turn would result in more designers being allocated to do coordination work. Conversely, as the *Coordination Priority* falls below 0.5 toward its minimum value of 0, the relative demand for designers to do coordination falls resulting in fewer designers being allocated as such.

In order to determine the *Desired Designers for Redesign*, one simply divides the *Desired Redesign Rate* by the *Normal Redesign Productivity* as in equation (151). However, to determine the *Desired Designers for Coordination* one must sum up the *Desired Coordination Rates* across components before dividing by the *Coordination Productivity.* Again, this is necessary because an integration error attached to a design from the "other" component is still an error involving designs from both components and must be coordinated by designers from both in order to be fixed.

Desired Designers for Redesign = Desired Redesign Rate / Normal Redesign Productivity

(151)

Desired Designers for Coordination = SUM (Desired Coordination Rate[Component!]) / Coordination Productivity

(152)

Once it is determined how many designers are available to do coordination work for a given component (*Component Designers Available for Coordination*), it is necessary to look across the components to determine the true number of designers available to do coordination. This is

necessary because, as mentioned, coordination requires a designer from each component and one component may have fewer designers available than the other to conduct coordination. In this case, the number of designers that are truly available to do coordination is the minimum of the number available across the components. This formulation is captured in equation (153) using the VMIN function from VENSIM which returns the minimum value across the vector of a subscripted variable.

Designers Available for Coordination = VMIN (Component Designers Available for Coordination[Component!])

(153)

The last piece of accounting in the allocation of designers is to determine the number of *Designers Available for Redesign.* We begin by determining the number of designers that are available after designers have been allocated for design work and coordination work. This number (*Designers Available after Coordination*) is calculated simply by taking the number of *Designers Available after Design* and subtracting the number of *Designers Working on Coordination.* However, in order to prevent this number from becoming negative the MAX function is used in VENSIM as it was used in equation (145). Calculating the number of designers involved in coordination requires summing the rates at which designs are being coordinated for both components since designers from each component are involved in the coordination of all of them. Dividing this summed rate of coordination by the *Coordination Productivity* determines the number of designers a given component has involved in coordination work at any point in time.

Designers Available for Redesign = IF THEN ELSE (Designers Available after Coordination < 0.25, 0, Designers Available after Coordination)

(154)

Designers Available after Coordination = MAX (Designers Available after Design - Designers Working on Coordination , 0)

(155)

Designers Working on Coordination[Component] = SUM (Coordinating Designs[Component!]) / Coordination Productivity

(156)

PRODUCTIVITY

Given the allocation of designers among design, redesign and coordination work it is necessary to determine the productivity of the designers in completing each kind of work. The productivity sector of the NPD model is what determines the rate at which designers will design, redesign and/or coordinate designs given the availability of designers and work to be done. At a basic level, it sets the various work rates in the model from a people perspective. The actual design, redesign and coordination rates are determined in the design sector as the minimum of the work rate possible from people and the work rate possible from the work available. The reader can look at the formulation of *New Design Rate* in equation (9) as an example. In the sections that follow, each of the work rates (*Design Rate from People*, *Redesign Rate from People*, and *Coordination Rate from People*) and their formulations will be discussed.

Design Rate from People

The design rate from people represents the rate at which new designs can be completed based on the number of designers available, their productivity, the amount of design work to be done and the time remaining until the next build. This design rate is set at the minimum of the *Desired*

Design Rate and the *Capable Design Rate* as outlined in equation (157). Figure 132 shows a schematic of the relationship between the key variables that determine the *Design Rate from People.*

Design Rate from People = min (Desired Design Rate , Capable Design Rate)

(157)

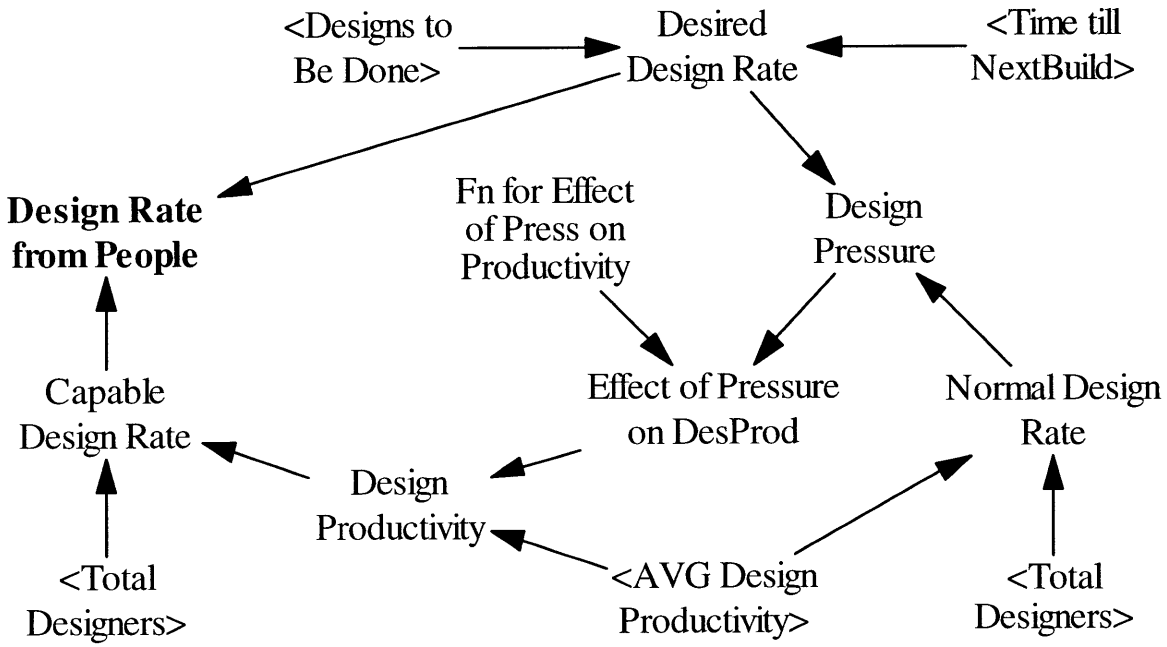


Figure 132 Design Rate from People

The *Desired Design Rate* is based on the number of designs to be completed and the time remaining until the next build event. It answers the question "How fast do we need to complete these designs in order to finish by the next build?" It can easily be calculated by dividing the number of designs still left in the stock of *Designs to Be Done* by the time left until the next build (*Time till NextBuild*). The *Capable Design Rate* is the rate at which new designs can be completed

based on the number of designers available and their design productivity as outlined in equation (159).

Desired Design Rate= ZIDZ (Designs to Be Done , Time till NextBuild)

(158)

Capable Design Rate = Designers * Design Productivity

(159)

By setting the *Design Rate from People* to the minimum of these two variables the model recognizes two features of design work. First, designers aren't able to work any faster than they are capable of even if they have a strong desire to speed up. As you will see later in this section, designers are able to work faster than their normal productivity to an extent but at some point they can't work any faster. The second feature of design work captured is the fact that designers will work at a slower pace than what they are capable of if they have more than enough time available to get the required work done. In this case, the designers will use the extra time to work on other activities either related to the project at hand or to other projects to which they are assigned.

Design Productivity represents the rate at which an individual designer is able to complete new designs. As outlined in equation (160), it is based on a normal productivity rate (*Normal Design Productivity*) and the effect that pressure has on that rate (*Effect of Pressure on DesProd*). Presumably as designers feel pressure to complete their designs prior to an upcoming build event they will work harder and faster, in some cases cutting corners, in order to get done on time. This effect is captured in the formulation for *Design Productivity.*

Design Productivity=AVG Design Productivity*Effect of Pressure on DesProd

(160)

Effect of Pressure on DesProd = Fn for Effect of Press on Productivity (Design Pressure)

(161)

Design Pressure = IF THEN ELSE (Desired Design Rate = 0, 0, xidz (Desired Design Rate , Normal Design Rate , 10))

(162)

Normal Design Rate=Total Designers*AVG Design Productivity

(163)

AVG Design Productivity=Normal Design Productivity*Experienced Fraction+Normal Design Productivity*New Designer Productivity Fraction*(1-Experienced Fraction)

(164)

The *AVG Design Productivity* is based on the *Normal Design Productivity* is set individually for the different components as the normal speed at which the components can be designed and may differ between components. For example, it might be easier (and therefore faster) to design the parts for a vehicle frame than it is to design the parts for the transmission. In the case of the NPD model, it is presumed that component A is able to design faster than component B. The formulation for the differences in both work and test rates between the two components will be discussed in the Speed Factor section of this chapter.

The effect that pressure has on *AVG Design Productivity* is to increase the rate at which new designs are completed as the design pressure increases. This effect is non-linear and reaches a maximum value of 1.667 times the normal productivity as depicted in Figure 133. Equation (161)

calculates the effect by using the non-linear table function (*Fn for Effect of Press on Productivity*) to return a value based on the value of *Design Pressure*.

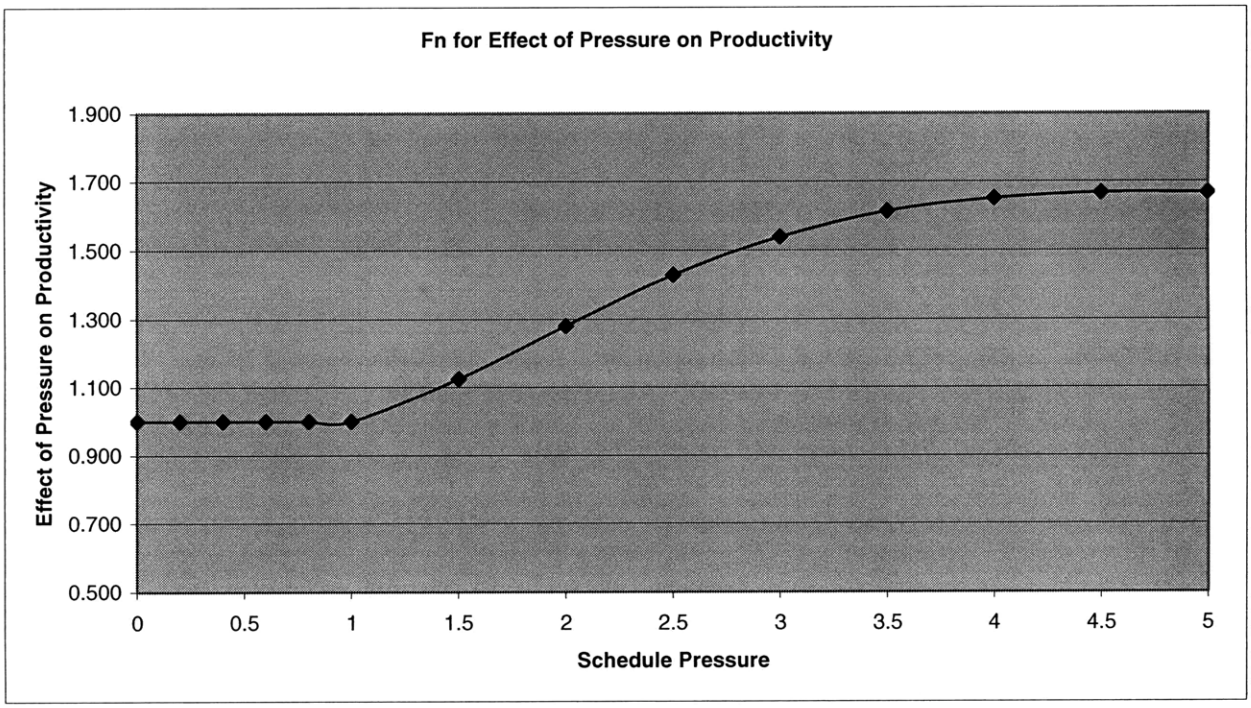


Figure 133 Effect of Pressure on Productivity

Design Pressure is a dimensionless variable used to represent the pressure felt by designers to complete design work as the *Desired Design Rate* increases relative to the *Normal Design Rate* which is the rate at which designs can be completed based on the number of designers available for new design work and the normal design productivity. The formulation for *Design Pressure* can be found in equation (162).

Redesign Rate from People

The redesign rate from people represents the rate at which designs in the stock of rework can be redesigned based on the number of designers available to do rework, their productivity, the amount of redesign work to be done and the time remaining until the next build. Figure 134 provides a schematic of the relationship between the key variables used in determining the *Redesign Rate from People*.

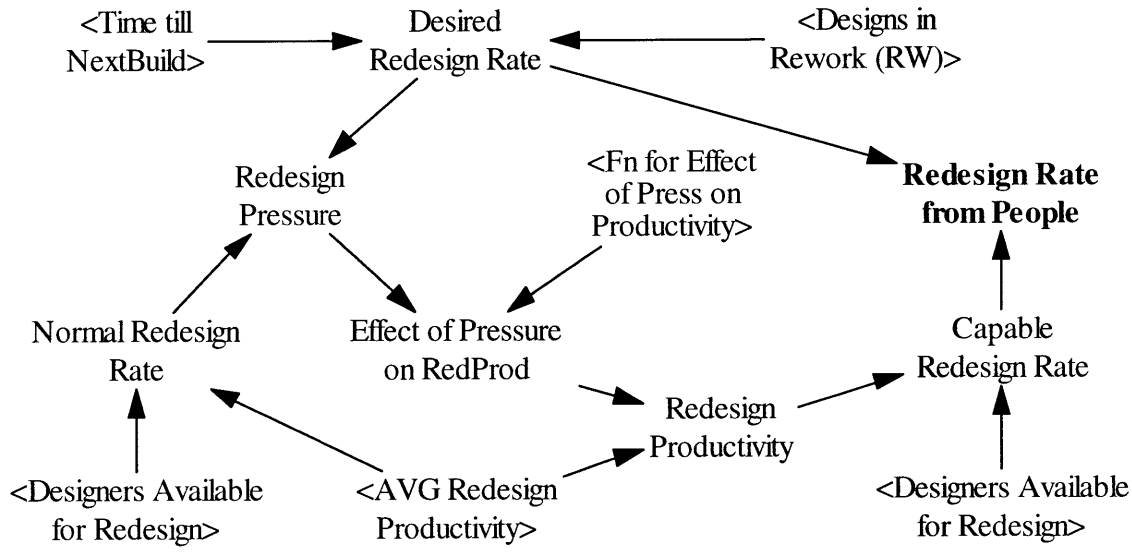


Figure 134 Redesign Rate from People

The formulation for *Redesign Rate from People* is completely analogous to the one described for *Design Rate from People* in the previous section. From the previous section we learned that the number of designers available for redesign work (*Designers Available for Redesign*) is determined based on a priority to always do new design work first and then to allocate the remaining designers between redesign work and coordination. You'll note that the same non-linear function used to determine the effect of pressure on *Design Productivity* (shown in equation (161)) is used to

determine the effect of pressure on *Redesign Productivity.* However, in this case *Redesign Pressure* is entered into the function to determine the effect.

The following is a partial equation listing of the relevant variables in determining the *Redesign Rate from People.* Equations for variables that have already been discussed are not included. See the technical documentation for more information regarding any of the variables.

Redesign Rate from People = MIN (Desired Redesign Rate , Capable Redesign Rate)

(165)

Desired Redesign Rate= ZIDZ (Designs in Rework (RW) , Time till NextBuild)

(166)

Capable Redesign Rate = Designers Available for Redesign * Redesign Productivity

(167)

Redesign Productivity=AVG Redesign Productivity*Effect of Pressure on RedProd

(168)

Effect of Pressure on RedProd = Fn for Effect of Press on Productivity (Redesign Pressure)

(169)

Redesign Pressure = IF THEN ELSE (Desired Redesign Rate = 0, 0, xidz (Desired Redesign Rate , Normal Redesign Rate , 10))

(170)

Normal Redesign Rate=Designers Available for Redesign*AVG Redesign Productivity

(171)

Coordination Rate from People

The coordination rate from people represents the rate at which designs with integration errors can be coordinated based on the number of designers available to do coordination, their productivity, the amount of coordination work to be done and the time remaining until the next build. Figure 135 provides a schematic of the relevant variables used in determining the *Coordination Rate from People*.

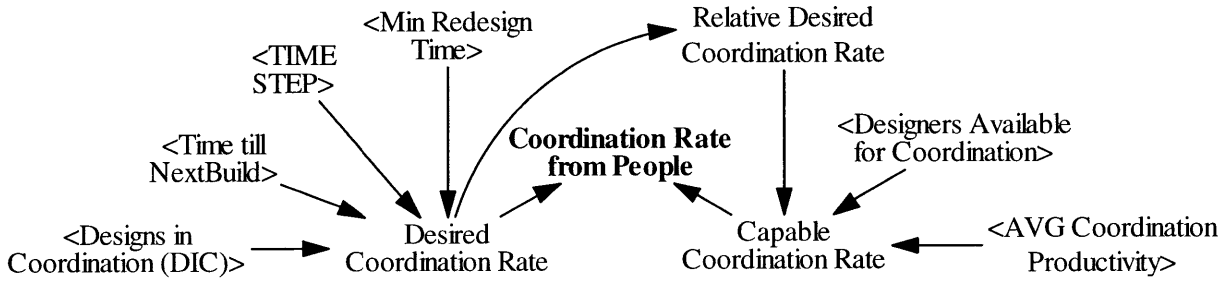


Figure 135 Coordination Rate from People

The basic formulation for the *Coordination Rate from People* is the same as the formulations used for the design and redesign rates above. The coordination rate from people is determined by taking the minimum of the *Desired Coordination Rate* and the *Capable Coordination Rate* as shown in equation (172). However, the formulation for these two rates differs from the methods used in the two previous sections.

Coordination Rate from People = MIN (Desired Coordination Rate , Capable Coordination Rate)

(172)

Desired Coordination Rate = zidz(Designs in Coordination (DIC),max(TIME STEP, Time till NextBuild-4*Min Redesign Time))

(173)

Capable Coordination Rate=Designers Available for Coordination*Relative Desired Coordination Rate*AVG Coordination Productivity

(174)

Relative Desired Coordination Rate[Component] = ZIDZ (Desired Coordination Rate[Component] , SUM (Desired Coordination Rate[Component!]))

(175)

AVG Coordination Productivity = Coordination Productivity*Experienced Fraction+Coordination Productivity*New Designer Productivity Fraction*(1-Experienced Fraction)

(176)

In determining the *Desired Coordination Rate*, the model factors in an estimate of the time it will take to complete a redesign after a given design is coordinated. This is handled in the formulation in equation (173) by subtracting four times the minimum time it takes for redesign (4*Min Redesign Time) from the time until the next build (Time till NextBuild). This represents the designers' estimate of redesign time based on the minimum time it takes to complete a redesign.22 Additionally, it is necessary to include the use of the MAX function in order to prevent the Desired Coordination Rate from becoming negative when the (Time till NextBuild – Min Redesign Time) term is negative. When this occurs, the model uses the time step of the simulation as the time remaining in order to set the Desired Coordination Rate.

The formulation for *Capable Coordination Rate* also differs from the analogous rates in design and redesign. Because coordination requires designers from both components but only one component design is being coordinated, it is necessary to allocate the coordination work between

the two components. This is done using the variable *Relative Desired Coordination Rate* which prioritizes the allocation of designers available for coordination using an US / (US+THEM) structure as formulated in equation (175). Essentially, this formulation adds together the desired coordination rates from both components and then determines what fraction of the total each component comprises and then allocates the available designers proportionally.

Lastly, the *Coordination Productivity* is set in the model to be the same for both components (A and B) and is not affected by pressure as was the case with design and redesign productivity. This formulation represents the reality that coordination involves two designers working closely together – not one faster than the other. Also, although one or both designers may feel pressure from a backlog of work and an impending build event it will certainly be different for both designers and thus when they come together to coordinate a design with an integration error they will maintain their normal coordination productivity and relieve their pressure elsewhere.

[22] Ideally, we would use an average redesign time based on historical data. For simplification, we do not maintain such an historical average in the model.

SPEED FACTOR

The Speed Factor sector of the NPD model sets the relative speed at which the different components (A and B) are able to complete their activities. This includes design and redesign work as well as bench testing and vehicle testing. Figure 136 provides a schematic of the relationship between the variables used to set the relative speeds of the two components' activities.

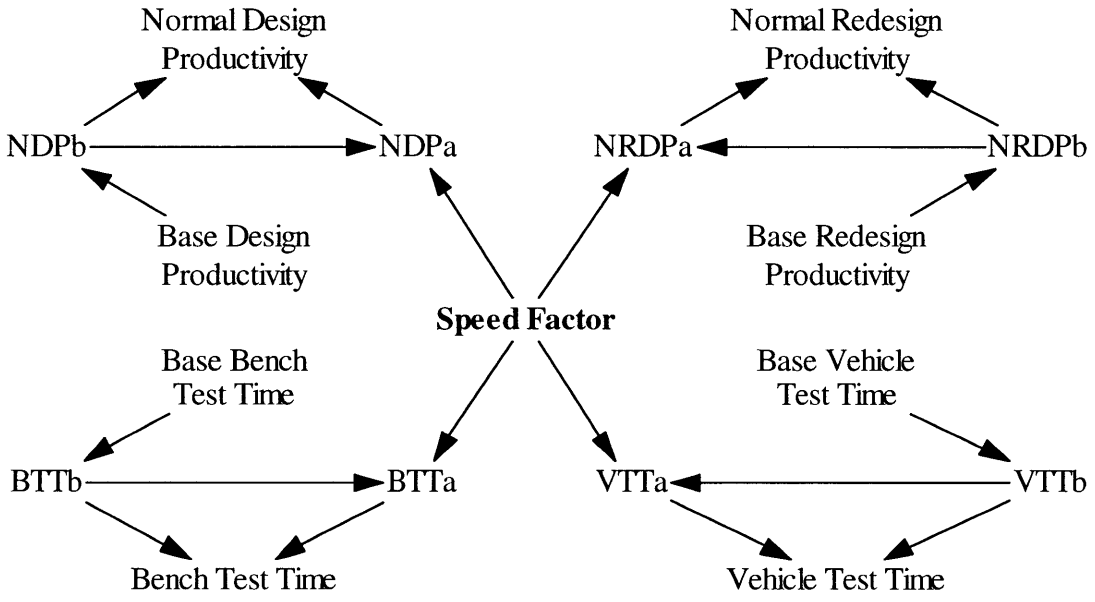


Figure 136 Speed Factor

Generally speaking, when there is a difference in speed between the two components it is assumed that component A is faster than component B. However, as formulated the NPD model can accommodate either component being faster than the other. The model variable *Speed Factor* sets the relative speed imbalance between the components. For example, the model's base speed factor of 2 indicates that component A is able to operate twice as fast as component B. This means

that component A's normal productivity rates are twice as fast as those for component B and its test times are only half as long.

In setting the *Normal Design Productivity* for the two components the model uses two intermediate variables – *NDPa* and *NDPb* – one for each component (A and B respectively) as outlined in equations (177) and (178). *NDPb*, the normal design productivity for component B, is set equal to the *Base Design Productivity* multiplied by a factor which is 2/(1+ Speed Factor) as outlined in equation
NDPb=Base Design Productivity*2*(1/(1+Speed Factor)

(179). *NDPa* is calculated by multiplying *NDPb* by the *Speed Factor* which is set at 2 in the model. The effect of these two equations is to ensure that Component A's design productivity is equal to a multiple (equal to the *Speed Factor*) of Component B's productivity while the average productivity remains equal to the *Base Design Productivity.* For example, when the *Speed Factor* equals 1 both *NDPa* and *NDPb* are equal to the *Base Design Productivity* of 10. When the *Speed Factor* equals 2, as is the case in the base NPD model, *NDPb* equals 2/3 of the *Base Design Productivity* (or 6.67) and *NDPa* equals twice that (or 13.33) for an average of 10.

Normal Design Productivity[a] = NDPa

(177)

Normal Design Productivity[b] = NDPb

(178)

NDPb=Base Design Productivity*2*(1/(1+Speed Factor)

(179)

NDPa = NDPb * Speed Factor

(180)

Base Design Productivity = 10

(181)

Speed Factor = 2

(182)

Setting the *Normal Redesign Productivity* for the two components is completely analogous to the design productivity just discussed. However, the Base Redesign Productivity is set at 5 designs per month per designer to reflect the fact that it takes more time to conduct fault analysis to establish the root cause of the error(s) in a design prior to redesigning it. The corresponding model equations are listed below.

Normal Redesign Productivity[a] = NRDPa

(183)

Normal Redesign Productivity[b] = NRDPb

(184)

NRDPb=Base Redesign Productivity*2*(1/(1+Speed Factor))

(185)

NRDPa = NRDPb * Speed Factor

(186)

Base Redesign Productivity = 5

(187)

The formulation required for setting the *Bench Test Time* for the two components is similar to the formulation used above for design and redesign. The model uses intermediate variables – *BTTa* and *BTTb* – to set the duration of the average *Bench Test Time* for each component. *BTTb*, the average bench test time for component B, is set equal to the *Base Bench Test Time* multiplied by a factor of 2 times the *Speed Factor* divided by 1 plus the *Speed Factor* as outlined in equation (190). *BTTa* is calculated by dividing *BTTb* by the Speed Factor. The combination of these two formulations ensures that the *Bench Test Time* of component B is a multiple of the *Bench Test Time* of Component A equal to the *Speed Factor* and that the average of the two is equal to the *Base Bench Test Time* which is set in the model to be 1 month. This means that on average it takes one month to prototype a design, outfit it with any special test gauges, run the bench test, receive and analyze the results.

Bench Test Time[a] = BTTa

(188)

Bench Test Time[b] = BTTb

(189)

BTTb = Base Bench Test Time*2*(Speed Factor/(1+Speed Factor))

(190)

BTTa = BTTb / Speed Factor

(191)

Base Bench Test Time = 1

(192)

The formulation for setting the *Vehicle Test Time* for the two components is exactly analogous to the one used above for setting the times for bench testing. However, the *Base Vehicle Test Time* is set at 3 months, considerably longer than the *Base Bench Test Time*. This longer test time reflects the time it takes to accumulate miles on the test vehicle and the time required to outfit the vehicle with various test gauges in order to collect different data. The equations needed to set the *Vehicle Test Time* that have not already been discussed are included for reference.

Vehicle Test Time[a] = VTTa

(193)

Vehicle Test Time[b] = VTTb

(194)

VTTb=Base Vehicle Test Time*2*(Speed Factor/(1+Speed Factor))

(195)

VTTa = VTTb / Speed Factor

(196)

Base Vehicle Test Time = 3

(197)

QUALITY

The Quality sector of the NPD model establishes the quality of design and redesign work completed by designers both in terms of component quality and integration quality. The quality of the work done is what determines the rate at which errors are introduced into the designs and corresponding parts as outlined in the Design and Build sectors. The basic framework for setting the level of quality recognizes four separate quality inputs: the component quality of design (*Design CQ*), the component quality of redesign (*Redesign CQ*), the integration quality of design (*Design IQ*) and the integration quality of redesign (*Redesign IQ*). A schematic of the relationship between these variables and those necessary to determine their values is included in Figure 137 .

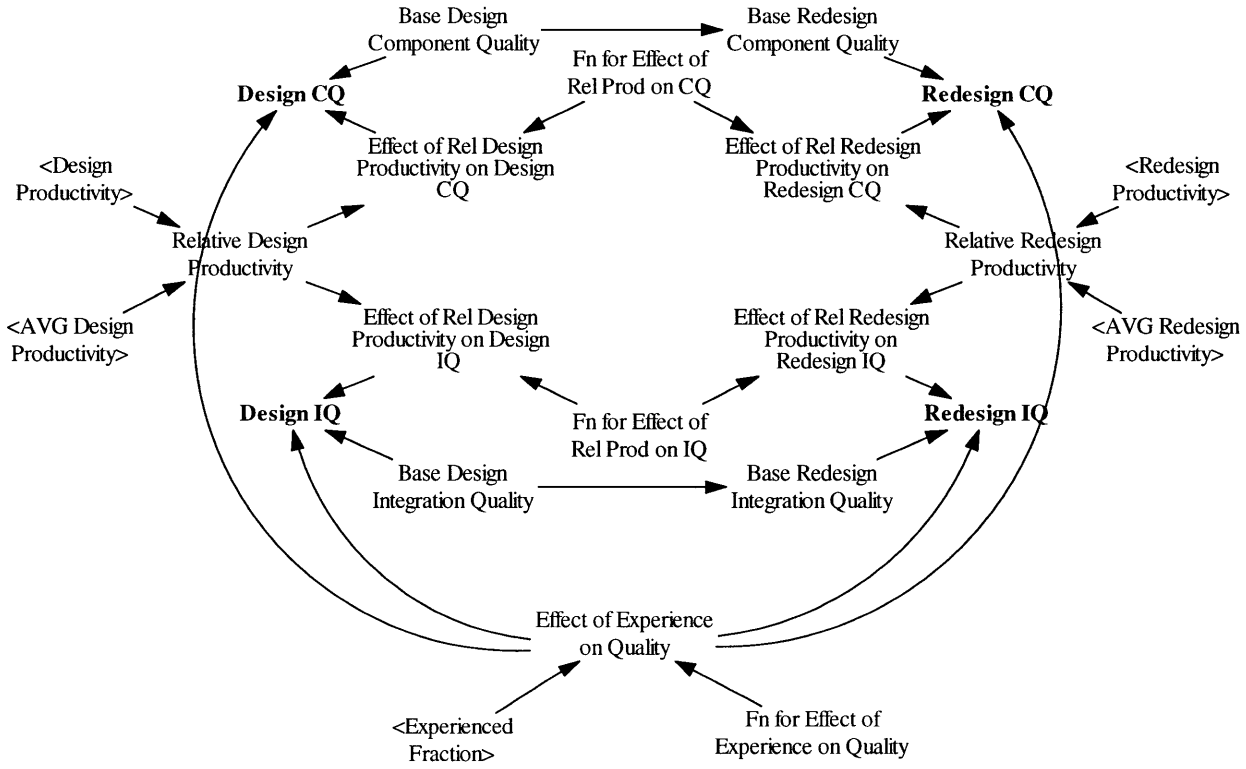


Figure 137 Quality

The formulation of the various values for quality is based on the premise that there is a base or normal quality of design (or redesign) that is affected by rate at which designers work relative to the normal work rate given the pressure of a work backlog and an impending build event. The result is that the actual quality of design (or redesign) can be considerably less than the normal quality when a designer speeds up their work rate, perhaps cutting corners, in an effort to clear their backlog before the next build. The formulation is quite similar to the one used to capture the effect of work pressure on productivity.

You'll recall that an increase in work pressure results in an increase in the productivity of individual designers with non-linear and ultimately diminishing returns. Designers work faster and

are able to get more designs or redesigns done as there work backlog increases and/or the next build event approaches. However, this working faster comes at the expense of quality as designers will often cut corners in order to get things done more quickly. While the impact on quality affects both component quality and integration quality, the drop-off in integration quality is more severe as designers tend to become more isolationist in their design work as the build approaches. Their chief concern is to make sure they have their part ready for the build and care less about making sure it fits with the other parts.

Component Quality of Design

The component quality of design work (*Design CQ*) determines what fraction of new designs are completed without introducing a component error as discussed in the Design sector of the NPD model. *Design CQ* is calculated by taking the normal component quality of design work (*Base Design Component Quality*) and multiplying it by the effect of relative design productivity on design component quality (*Effect of Rel Design Productivity on Design CQ*) and again by the effect of experience on component quality (*Effect of Experience on Quality*) as shown in equation (198). The base component quality for design is set at 0.85 for both components. This base quality means that when designers are working at their normal productivity pace 85% of their new designs will be completed without introducing a component error.

Design CQ = Base Design Component Quality*Effect of Rel Design Productivity on Design CQ*Effect of Experience on Quality

(198)

Base Design Component Quality = 0.85

(199)

Effect of Rel Design Productivity on Design CQ = Fn for Effect of Rel Prod on CQ(Relative Design Productivity)

(200)

Relative Design Productivity=Design Productivity/AVG Design Productivity

(201)

Effect of Experience on Quality=Fn for Effect of Experience on Quality(Experienced Fraction)

(202)

The effect of increasing the *Relative Design Productivity* (i.e. speeding up the rate at which designers complete designs) is a corresponding decrease in the design component quality and thus a decrease in the percent of new designs completed without a component error. The effect of the relative design productivity on component quality is determined using a table function which takes *Relative Design Productivity* as an input and returns a value between 0 and 1 as formulated in equation (200). This table function is depicted graphically in Figure 138. *Relative Design Productivity* is calculated simply by dividing the actual *Design Productivity* by the *AVG Redesign Productivity* as outlined in equation **Relative Design Productivity=Design Productivity/AVG Design Productivity** (201).

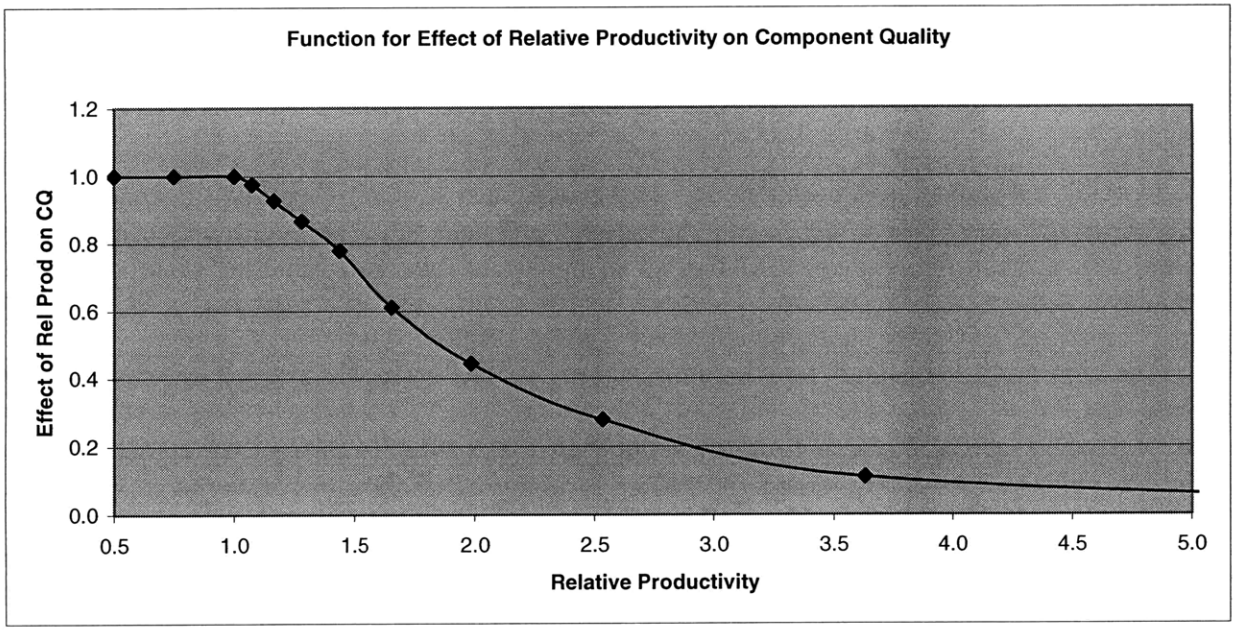


Figure 138 Effect of Relative Productivity on Component Quality

As you can see, the function is decreasing and S-shaped with the drop-off fairly severe as the relative design productivity increases from 1 to 2.5 with the effect saturating at a minimum value of 0 at a *Relative Design Productivity* of 12. As an example, the effect of speeding up the design productivity to three times the normal rate (*Relative Productivity* = 3) will cause the component quality to drop to less than 20% of the base component quality.

You'll note that even in the region of a low relative design productivity (<1) the effect on component quality never goes above 1; thus, *Design CQ* can never be higher than the *Base Design Component Quality*. In other words, reducing the rate at which designs are completed below the normal productivity level does not have a positive impact on component quality. This recognizes that designers with low *Design Pressure* will not slow down their individual design work (thus reducing their productivity below normal) but rather will work on other requirements within the

new product development project (perhaps redesign or coordination) or on requirements from other projects to which they are assigned.

The effect of a decrease in the experience of the designers is a corresponding decrease in the quality of design work. The effect of experience on component quality is determined using a table function which takes *Experienced Fraction* as an input and returns a value between 0 and 1 as formulated in equation

Effect of Experience on Quality=Fn for Effect of Experience on Quality(Experienced Fraction)

(202). In this case, the table function also is based on an S-shaped curve that decreases from a value of 1 when the *Experienced Fraction* is equal to 1 to a value of 0.67 when the *Experienced Fraction* is equal to 0. In effect, this means that when all of the designers on a project team are inexperienced (i.e. *New Designers*) then the effect of experience on quality will be to reduce the component quality of design to 67% of what it would be otherwise.

Component Quality of Redesign

The component quality of redesign (*Redesign CQ*) together with the *Redesign Rate* determines the rate at which known component errors are fixed in redesign and the rate at which new component errors are introduced during redesign work. It is formulated in the same manner as *Design CQ* except it uses a different base component quality for redesign *Base Redesign CQ* and uses *Relative Redesign Productivity* as the input to the table function that determines the effect of relative productivity on component quality. The exact same *Effect of Experience on Quality* is used for redesign component quality as was used for design quality. The same table function, depicted graphically in Figure 138, that was used for *Design CQ* is used to determine the effect of relative productivity on *Redesign CQ* as well.

Redesign CQ= Base Redesign Component Quality*Effect of Rel Redesign Productivity on Redesign CQ*Effect of Experience on Quality

(203)

Base Redesign Component Quality = 1 - (1 - Base Design Component Quality) / 2

(204)

Effect of Rel Redesign Productivity on Redesign CQ=Fn for Effect of Rel Prod on CQ(Relative Redesign Productivity)

(205)

Relative Redesign Productivity = Redesign Productivity/Normal Redesign Productivity

(206)

You'll note that the *Base Redesign Component Quality* is a function of the base design CQ. The formulation in equation (204) sets the value halfway between the *Base Design Component Quality* and 1. Given the model's default value of 0.85 for the *Base Design Component Quality*, the base value for redesign CQ is 0.925.

Integration Quality of Design

The integration quality of design (Design IQ) determines the fraction of new designs that are completed without introducing an integration error as outlined in the Design sector of the NPD model. The formulation for Design IQ is completely analogous to that used for Design CQ except that it uses a different Base Design Integration Quality and a different table function for determining the effect of pressure on integration quality. Equations (207) through (209) show the formulation.

Design IQ=Base Design Integration Quality*Effect of Rel Design Productivity on Design IQ*Effect of Experience on Quality

(207)

Base Design Integration Quality = 0. 5

(208)

Effect of Rel Design Productivity on Design IQ= Fn for Effect of Rel Prod on IQ(Relative Design Productivity)

(209)

Again, the *Base Design Integration Quality* is the same for both components (A and B) and is set in this case to 0.5. This means that when designers are working at their base rate of productivity they will introduce integration errors into 50% of their new designs. The table function used to determine the effect of relative design productivity is somewhat different than the one used for component quality. The table function used for integration quality is depicted graphically in Figure 139.

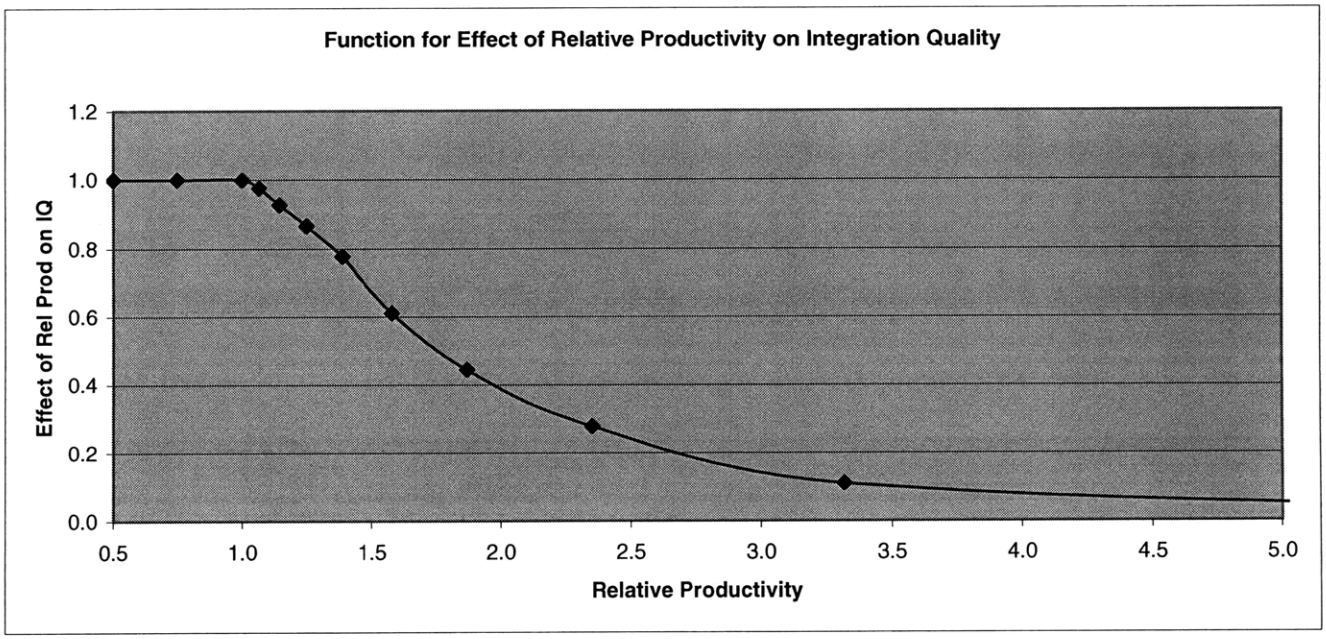


Figure 139 Effect of Relative Design Productivity on Integration Quality

You'll note that the general shape of the curve is the same as that for component quality. However, the drop-off is slightly more severe and sudden in the case of integration quality as one of the first places that designers cut corners when reacting to design pressure is to become more isolated in their work. The result is more of a concern to get their "own work" done and spend less time worrying about integrating their work (i.e. their designs) with other designers. As a result, the *Effect of Relative Design Productivity on Design IQ* falls below 0.4 when a designer is working at twice the normal productivity rate. As in the case of component quality, reducing the *Relative Design Productivity* below 1 does not result in an increase in the integration quality above its base value.

Integration Quality of Redesign

The integration quality of redesign (*Redesign IQ*) together with the *Redesign Rate* determines the rate at which coordinated integration errors are fixed in redesign and the rate at which new integration errors are generated during redesign work. It is formulated in the same manner as *Design IQ* except it uses a different base integration quality for redesign (*Base Redesign IQ*) and uses *Relative Redesign Productivity* as the input to the table function that determines the effect of redesign productivity on quality. The same table function, depicted graphically in Figure 139, that was used for *Design IQ* is used to determine the effect of productivity on *Redesign IQ*. Lastly, the same *Effect of Experience on Quality* is used here to adjust quality based on the experience of the designers.

Redesign IQ=Base Redesign Integration Quality*Effect of Rel Redesign Productivity on Redesign IQ*Effect of Experience on Quality

(210)

Base Redesign Integration Quality = 1 - (1 - Base Redesign Component Quality) / 2

(211)

Effect of Rel Redesign Productivity on Redesign IQ=Fn for Effect of Rel Prod on IQ(Relative Redesign Productivity)

(212)

Like the base component quality of redesign, the *Base Redesign Integration Quality* is set halfway between the *Base Design Integration Quality* and 1. At the default value for the NPD model, this sets the base value for redesign IQ at 0.75.

BENCH TEST FRACTION

The *Bench Test Fraction* determines the fraction of designs and redesigns that are sent for bench testing at a given point in time. It represents the designers' willingness to send a design or redesign for bench testing prior to the next build event. There are two prevailing thought processes that play into the decision as to whether or not to bench test a design. First, because vehicle testing is presumed to be a more comprehensive, realistic and accurate test of a design's suitability some designers choose to wait until the next build event and vehicle testing rather than bench test their design. Secondly, as the next build event draws near some designers fear that there will be insufficient time to get bench test results and/or make the necessary corrections to their design prior to the next build event. This fear results in fewer designs and redesigns being sent for bench testing as a build event approaches. As such, the bench test fraction (*BT Fraction*) is based on both the designers' intentions toward bench testing (*Intended Bench Test Fraction*) and the effect of redesign pressure (*Effect of Redesign Pressure on BT Fraction*) as depicted in Figure 140.

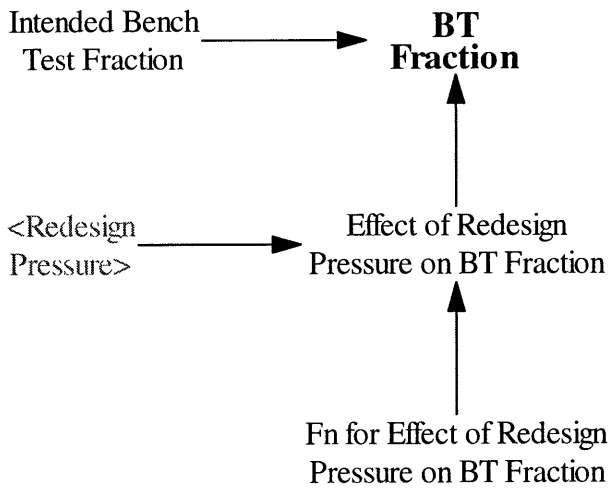


Figure 140 Bench Test Fraction

The equations for BT Fraction and the relevant input variables are included below with a brief description of each. As with all model variables, the technical documentation can be found in Appendix B.

BT Fraction = Intended Bench Test Fraction * Effect of Redesign Pressure on BT Fraction

(213)

Intended Bench Test Fraction = 0.5

(214)

Effect of Redesign Pressure on BT Fraction = Fn for Effect of Redesign Pressure on BT Fraction (Redesign Pressure)

(215)

The bench test fraction is calculated simply by taking the *Intended Bench Test Fraction* and multiplying it by the effect of redesign time remaining prior to the next build (*Effect of Rd Time*

aBT on BT Fraction). In the NPD model, the *Intended Bench Test Fraction* is set at 0.5 meaning that given ample time prior to the next build designers would send only half of their designs or redesigns for bench testing. The *Intended Bench Test Fraction* reflects the fact that designers are somewhat leery of the results of bench testing. Some believe that the bench test conditions are unrealistic and ignore errors found in bench testing until they are "verified" in vehicle testing.. Other designers fear that bench testing "over tests" their design and discover "false" errors and thus they avoid bench testing altogether. The *Effect of Redesign Pressure on BT Fraction* is a fraction between 0 and 1 which decreases non-linearly as *Redesign Pressure* grows. The effect represents the designers' unwillingness to send designs for bench testing when they are concerned that they might not have time to rework the design if it is found to have an error. Designers are very concerned about showing up to a build event with a part that has a known error in its design. The effect is captured using a table function depicted graphically in Figure 141.

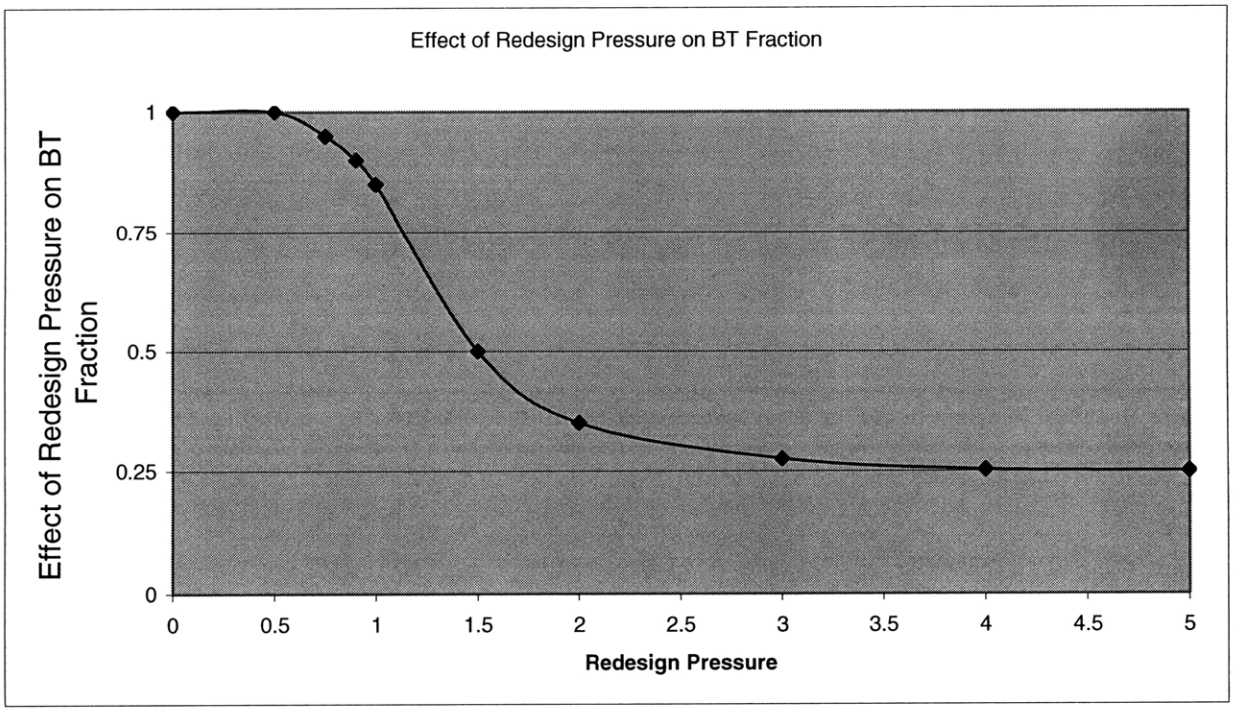


Figure 141 Effect of Redesign Pressure on Bench Test Fraction

When *Redesign Pressure* is equal to 1 (i.e. *Desired Redesign Rate* is equal to the *Normal Redesign Rate*) – the effect on the bench test fraction is equal to 0.85 which means that the bench test fraction will be reduced to 85% of the *Intended Bench Test Fraction.* This means that when designers are completing redesign work at a normal pace and meeting the demand (i.e. the Desired Redesign Rate) that there will still be some reluctance by designers to send their designs for bench testing out of fear that there will be insufficient time to rework a design found to have an error. It is only when the *Redesign Pressure* falls to 0.5 or below (when the desired redesign rate is only half of the normal redesign rate given available designers) that the *Effect of Redesign Pressure on BT Fraction* equals one meaning that designers will send the full *Intended Bench Test Fraction* of designs for bench testing. From the graph in Figure 141, it becomes clear that the effect of increasing redesign pressure becomes more severe in depressing the bench test fraction. However, the effect saturates at a value of 0.25 to reflect the fact that no matter how severe the redesign pressure some designers will send a portion of their designs to bench testing anyhow.

The table function for the *Effect of Redesign Pressure on BT Fraction* is included in Figure 142. This function is the same one that is graphed in Figure 141 above.

Redesign Pressure	Effect of Redesign Pressure on BT Fraction
0	1
0.5	1
0.75	0.95
0.9	0.9
1	0.85
1.5	0.5
2	0.35
3	0.275
4	0.2525
5	0.25
10	0.25

Figure 142 Table Function for Effect of Redesign Pressure on BT Fraction

COORDINATION FRACTION

The *Coordination Fraction* determines the fraction of designs found to have both a component and an integration error that are sent for coordination prior to being redesigned. The act of coordinating a design with an integration error requires a designer from each of the coupled components (A and B). Together the designers will coordinate the rework required in order to fix the integration error. The outcome of coordinating a design with an integration error is to send either just the design with the attached integration error or both designs from the coupled "design pair" to the stock of *Designs in Rework (RW).*

The *Coordination Fraction* represents the willingness of a designer to delay rework on a known component error until he or she is able to link up with a designer from the coupled

component to coordinate the integration error. Similar to the bench test fraction, there are two factors that determine the *Coordination Fraction.* The first factor is the fraction of designs with both errors that designers would send for coordination if they had ample time until the next build to both coordinate and redesign the given design. This is represented in the NPD model by the variable *Intended Coordination Fraction.* The second factor is the amount of coordination pressure that exists at the time the design is found to have both an integration error and a component error. This represents the designers' unwillingness to put off reworking a design to fix a component error for too long while waiting to coordinate an integration error. A schematic of the relationship between the variables relevant in determining the *Coordination Fraction* is included in Figure 143.

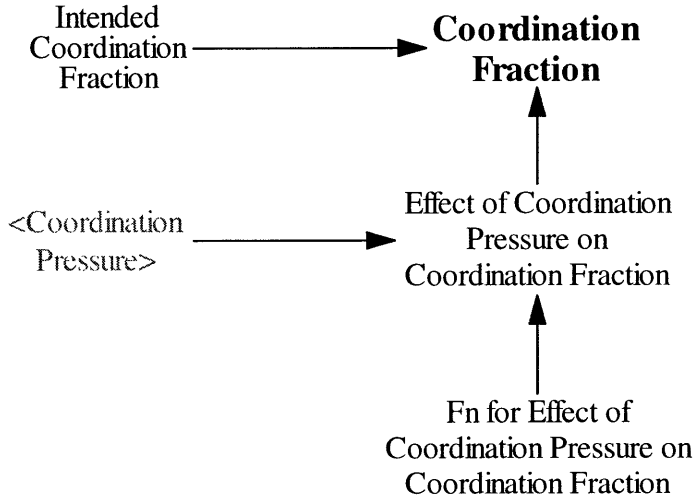


Figure 143 Coordination Fraction

The equation for *Coordination Fraction* and its relevant input variables along with a brief description of the formulation for each is included below. You'll note that the formulation is analogous to that for the bench test fraction (*BT Fraction*) discussed previously.

Coordination Fraction = Intended Coordination Fraction * Effect of Coordination Pressure on Coordination Fraction

(216)

Intended Coordination Fraction = 1

(217)

Effect of Coordination Pressure on Coordination Fraction = Fn for Effect of Coordination Pressure on Coordination Fraction (Coordination Pressure)

(218)

Coordination Pressure = IF THEN ELSE (Desired Coordination Rate = 0, 0, xidz (Desired Coordination Rate , Capable Coordination Rate , 10))

(219)

The basic formulation for *Coordination Fraction* as shown in equation (216) is identical to that of the bench test fraction in equation (213). However, in this case the *Intended Coordination Fraction* is set at 1. This is considerably higher than the *Intended BT Fraction* set at 0.5 because in this case there is a known integration error that must be fixed and in order to fix it the design must be coordinated. You'll recall that in the case of bench testing, designers do not have a known error prior to testing and are leery of the results of bench testing.

In addition to a higher *Intended Coordination Fraction*, the function used to determine the effect of redesign time available after coordination on the *Coordination Fraction* is different than the one used for the *BT Fraction*. It, too, decreases non-linearly as the Coordination Pressure increases. However, as you can see in Figure 144 the rate at which the effect decreases the *Coordination Fraction* as the pressure increases is more severe than that for *BT Fraction* in Figure 141. The effect decreases the *Coordination Fraction* both sooner and more dramatically as *Coordination Pressure* increases.

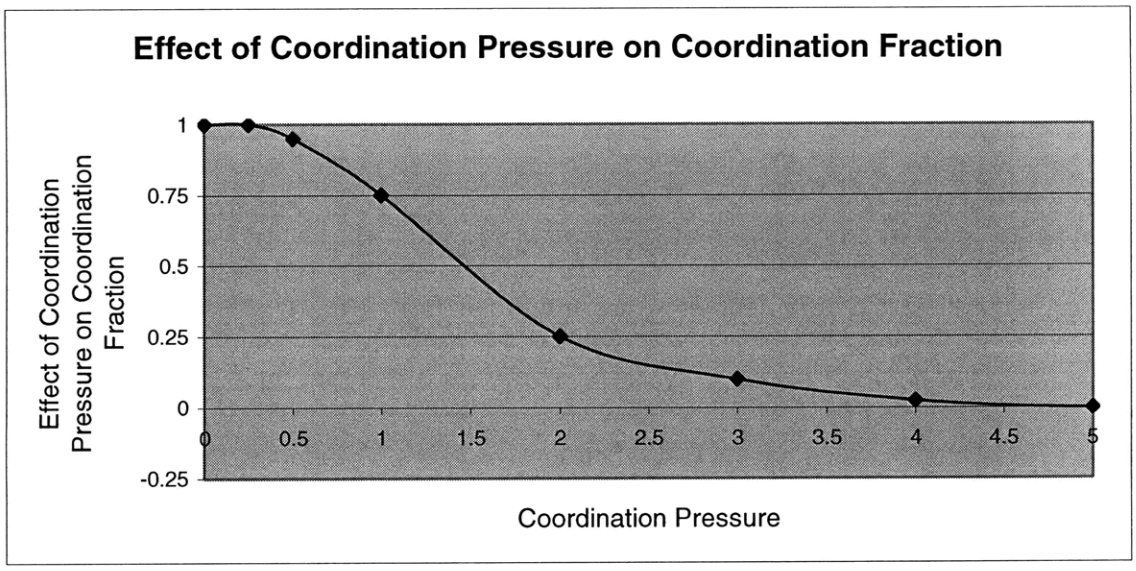


Figure 144 Effect of Coordination Pressure on Coordination Fraction

The effect of pressure is more severe in the case of the *Coordination Fraction* because in addition to the integration error associated with a given design the designer has a known component error that he must also try to fix prior to the next build. Thus, as the coordination pressure increases, the likelihood of getting a design with an integration error coordinated with sufficient time to fix it and its known component error prior to the next build goes down; and with

it, the fraction of designs with both types of errors sent for coordination goes down as well. This reflects a feeling among designers that a designer who brings a part to a build with a known component error has no one else to blame but himself while the blame for a build part with a known integration error can at least be divided between the designers from the coupled components.

When *Coordination Pressure* is equal to 1 (i.e. *Desired Coordination Rate* is equal to the *Capable Coordination Rate*) the effect on the *Coordination Fraction* is equal to 0.75 which means that the fraction of designs with both types of errors sent for coordination will be reduced to 75% of the *Intended Coordination Fraction.* When the *Coordination Pressure* falls to 0.5 (when the *Desired Coordination Rate* is half the *Capable Coordination Rate* given the available designers) the effect is equal to 0.95 meaning that the *Coordination Fraction* will be at 95% of its intended fraction. It isn't until the *Coordination Pressure* falls to 0.25 that the *Coordination Fraction* equals the *Intended Coordination Fraction.* You'll note that the effect falls all the way to zero at a *Coordination Pressure* of 5 which means that when the *Desired Coordination Rate* is five times the *Capable Coordination Rate* the *Coordination Fraction* will fall to zero meaning no designs with both types of errors will be sent for coordination prior to redesign.

The table function for the *Effect of Coordination Pressure on Coordination Fraction* is included in Figure 145.

Coordination Pressure	Effect of Coordination Pressure on Coordination Fraction
0	1
0.25	1
0.5	0.95
1	0.75
2	0.25
3	0.1
4	0.025
5	0
10	0

Figure 145 Table Function for Effect of Coordination Pressure on Coordination Fraction

The last piece of model formulation that is relevant to the *Coordination Fraction* is the computation for *Coordination Pressure*. In much the same manner as *Redesign Pressure*, it is calculated by dividing the *Desired Coordination Rate* by the *Capable Coordination Rate*.

Coordination Pressure = IF THEN ELSE (Desired Coordination Rate = 0, 0, xidz (Desired Coordination Rate , Capable Coordination Rate , 10))

(220)

EXTERNAL REWORK

External rework is the rework that is generated for a given component when it "shares" an integration error with a design in a coupled component. By convention in the NPD model, the integration error is attached to only one of the designs in coupled "design pair." During the process of coordinating the required rework necessary to fix the integration error, the two designers

(one from each component in the "design pair") determine if one or both of the designs must be redesigned. By convention, the design with the attached integration error will always require rework. With some probability the second design in the "design pair" will require rework as well. External rework refers to the redesign work required of this second design.

Accounting for external rework involves two processes. The first is to determine how much external rework to assign to a given component. The second is to determine which designs to move into the stock of Designs in Rework (RW). Figure 146 shows a schematic of the relationship between the key variables necessary to account for external rework. The equations necessary to account for external rework along with a brief description of the formulation for each variable are also included.

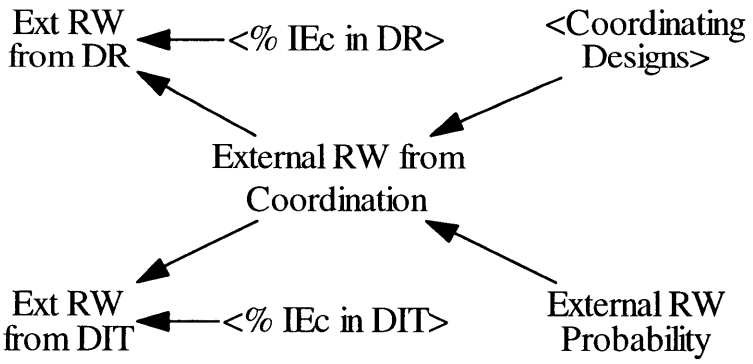


Figure 146 External Rework

Determining the amount of external rework to assign to a given component is fairly straightforward. In the NPD model it is assumed that as designs with integration errors are coordinated that external rework will be generated with a fixed probability. Because the external rework is assigned to the "other" component it is necessary to use two separate equations to determine the rate at which external rework is assigned as outlined in equations (221) and (222). By simply multiplying the rate at which designs from the coupled component are coordinated (*Coordinating Designs*) by the probability that both designs will require rework (*External RW Probability*), we can calculate the rate at which external rework is assigned to a given component (*External RW from Coordination*). The formulation for *Coordinating Designs* is included in equation (27)as is the *External RW Probability* in equation (77).

External RW from Coordination[a] = Coordinating Designs[b] * External RW Probability

(221)

External RW from Coordination[b] = Coordinating Designs[a] * External RW Probability

(222)

Once we know the rate at which external rework is assigned to a given component, we need to determine which designs will be sent for rework. The first step in doing so is to recognize that the designs we send for external rework, by definition, can not have integration errors. This is true because the integration error in question is attached to the other design in the coupled design pair. However, the designs we send for external rework may (or may not) have a component error,

known or unknown, at the time we send it for rework. As such, we need to determine the relative densities of designs without integration errors in the stocks of designs in progress. We refer to these designs as being "IE Clean." Since by definition designs in coordination (*DIC*) must have an integration error we do not need to consider this stock. Also, since the IE Clean designs that are already in the stock of rework do not need to be moved if we assign them external rework we do not need to explicitly determine the rate at which external rework is assigned to these designs. That leaves us with the following two equations for assigning external rework to designs in the stocks of designs released (*DR*) and designs in test (*DIT*):

Ext RW from DR = External RW from Coordination * % IEc in DR

(223)

Ext RW from DIT = External RW from Coordination* % IEc in DIT

(224)

The formulation takes the rate at which external rework is assigned to a given component and multiplies it by the percentage of IE Clean designs for that component that reside in either designs released (DR) or designs in test (DIT). These rates, *Ext RW from DR* and *Ext RW from DIT*, are used to move designs and their associated component errors to the stock of rework as outlined in the design sector. The formulation for *% IEc in DR* and *% IEc in DIT* are included in the technical documentation.

PHANTOM WORK

Phantom work is the term used to represent potentially unnecessary work that is done during the course of a development project. This phantom work is created from a mismatch in the speeds at which components can iterate their design work. The amount of phantom work can vary greatly depending on the degree to which the components design iteration speeds are mismatched, the degree to which the components are coupled, and the build event timing.

There are two types of phantom work that we characterize in the NPD model. The first is the creation of *phantom integration errors.* These represent integration errors that are created from previous integration errors that have been discovered, coordinated, found to require rework of both designs in the coupled "design pair", but only one of the two redesigns is completed by the next build event. The second type of phantom work is called *phantom rework.* It represents the rework done primarily on designs with phantom integration errors after a build event has occurred but before the new "phantom" error is discovered. Each of these types of phantom work will be discussed in more detail below and the model structure used to account for each will be described.

Phantom Integration Errors

Phantom integration errors are "new" integration errors created from old ones because one of the two coupled designs requiring rework did not get redesigned in time for the next round of vehicle testing (i.e. the next build). These are not the same as old integration errors that "carry over" to the next build because they were redesigned but not fixed. Nor are they the same as integration errors that reoccur on the next build because none of the required rework was completed in time.

Phantom integration errors occur when the coordination of an integration error results in a requirement for both designs from the coupled component "design pair" to be redesigned and one of the two redesigns is completed before the next build event and the other is not. When this happens, a new integration error will be discovered in vehicle testing. This integration error exists between the new version of the redesigned part and the old version of the coupled part that was not redesigned. Because this "design pair" is different and indeed more current than the old "design pair" that had the original integration error, the integration error discovered in the subsequent build will be considered a new integration error.

At a minimum the new (or phantom) integration error will need to be coordinated – requiring designers from both components – in order to determine the root cause of the new integration error and what rework will be required to fix it. In the worst case, the designers may determine that both designs will need to be reworked in order to fix the new integration error.

We refer to these new integration errors as *phantom* errors because if the slow iterating component had had enough time to complete its redesign prior to the second build and it did so correctly, the new integration error would not have occurred.

Tracking how many phantom integration errors occur during the course of a development project follows from the integration error accounting done in the Design Sector. Accordingly, discussion of the formulation of variables used from the Design Sector used will be limited and the reader can refer to the previous description in the Design Sector. The stock and flow structure used to account for phantom integration errors (*Phantom IE*) is depicted in Figure 147.

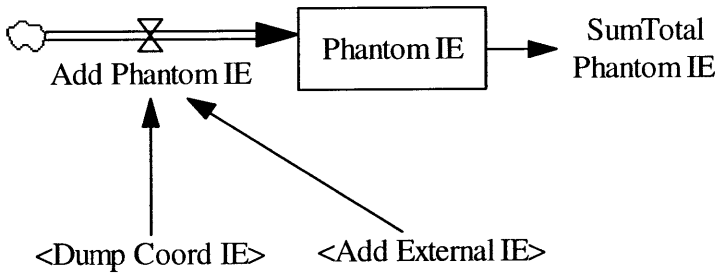


Figure 147 Phantom Integration Errors

The stock of *Phantom IE* accumulates the phantom integration errors as defined above for a given component over the course of the new product development project. It has a single inflow – *Add Phantom IE* – and no outflows. The equation for *Phantom IE* and its inflow are included below. You'll note that both *Dump Coord IE* and *Add External IE*, which serve as inputs to the inflow, are flows from the integration error co-flow in the Design Sector. As expected, the initial value of the stock of *Phantom IE* is zero.

Phantom IE = INTEG(Add Phantom IE , 0)

(225)

Add Phantom IE = Add External IE + Dump Coord IE

(226)

SumTotal Phantom IE = SUM (Phantom IE[Component!])

(227)

Lastly, the variable *SumTotal Phantom IE* represents the total number of accumulated phantom integration errors across all components. It is calculated using the SUM function in VENSIM to add together the *Phantom IE* across the Component subscript (i.e. for both component A and B) as listed in equation (227).

Phantom Rework

Phantom rework refers to the potentially unnecessary redesign work that is done during the course of a development project. There are two main sources of phantom rework. The first source of phantom rework is the redesign work done after the build event has occurred and vehicle testing has started on designs with coordinated integration errors that required both designs from the coupled design pair to be reworked and only one of the redesigns was done at the time of the build. This phantom rework can be done on designs that had the integration error attached to them or on designs that had been designated for external rework. Essentially, this phantom rework is the redesign work done on designs with phantom integration errors, as defined above, before the phantom integration errors are discovered. As such, the phantom integration errors cannot be fixed in redesign and will need to be coordinated and redesigned again upon discovery. Thus any rework done on these designs after the build but prior to discovering the phantom IE is potentially unnecessary since it will have to be done again. The second source of phantom rework is the external rework that is required of a component when a phantom integration error attached to a design from a coupled component is coordinated. In this case, because of the way that phantom integration errors are defined and generated in the model, we know that external rework assigned to a component from the coordination of a phantom integration error represents rework that it being done for the second time on the same design and could have been avoided had the phantom integration error not been generated.

It is necessary to create some new structure to account for the phantom rework (*Phantom RW*) as defined above. However, as in the case of accounting for *Phantom IE* we will be able to make use of existing model structure from the integration error co-flow in the Design Sector. Figure 148 shows the model structure and variables needed to track the *Phantom RW* as it is generated during the course of a project in the simulation.

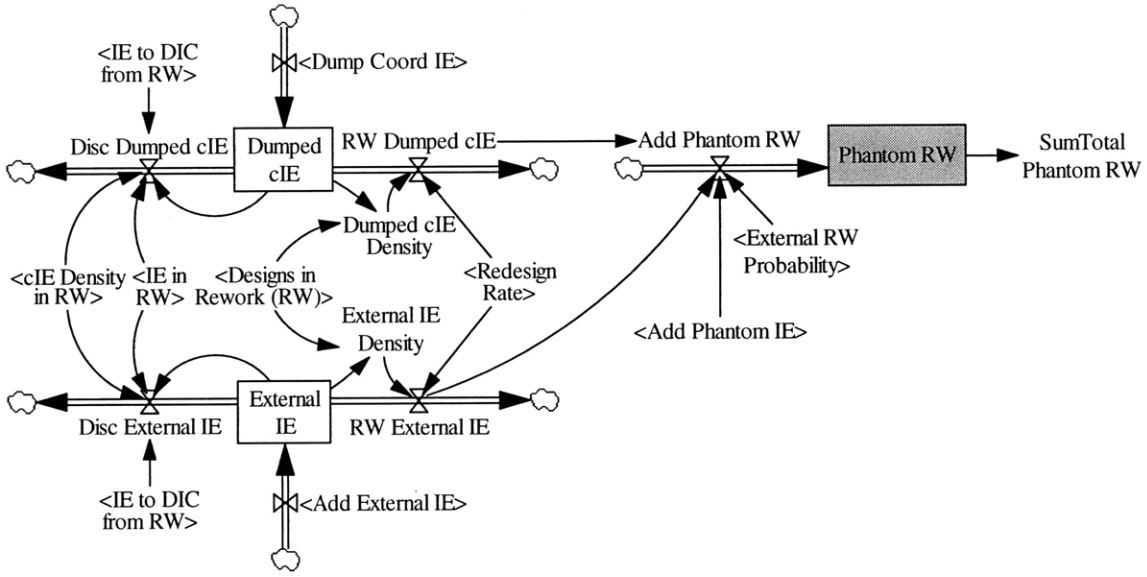


Figure 148 Phantom Rework

As you can see in Figure 148, the stock of *Phantom RW* has only one inflow which is *Add Phantom RW*. However, as mentioned above there are two primary sources for generating phantom rework. The first is reworking designs with outdated integration errors. This type of phantom rework is represented by the flows *RW Dumped cIE* and *RW External IE* which are both inputs to the inflow *Add Phantom RW*. The second source of phantom rework is the external

rework that is generated in coordinating phantom integration errors. While this phantom rework is not done until some time after the phantom integration errors are discovered and coordinated, it is added to the stock of *Phantom RW* at the time the phantom integration errors are added to the stock of *Phantom IE*. This process is represented by the use of *Add Phantom IE* and External *RW Probability* as inputs to the inflow *Add Phantom RW*.

The equations for *Phantom RW* and its inflow are included below. A description of the formulation follows.

Phantom RW = INTEG(Add Phantom RW , 0)

(228)

Add Phantom RW[a] = RW Dumped cIE[a] + RW External IE[a] + Add Phantom IE[b] * External RW Probability

(229)

Add Phantom RW[b] = RW Dumped cIE[b] + RW External IE[b] + Add Phantom IE[a] * External RW Probability

(230)

SumTotal Phantom RW = SUM (Phantom RW[Component!])

(231)

The stock of *Phantom RW* begins with an initial value of zero and accumulates the inflow *Add Phantom RW*. There are no outflows. Adding phantom rework to the stock requires a different equation for each component (A and B) as the inflows are dependent on the number of phantom integration errors being added to the coupled component. *RW Dumped cIE* represents the rate at

which designs with coordinated integration errors that have been dumped are redesigned. *RW External IE* represents the rate at which designs with external integration errors are redesigned. A description of how these rates are determined is included below. The last term, Add *Phantom IE * External RW Probability*, represents the additional external rework that will be required based on the coordination of designs with phantom integration errors. The sum of these three determines the rate at which designs are added to the stock of *Phantom RW.*

In order to determine the redesign rate of designs that have coordinated integration errors that were dumped we need to create a co-flow of these particular errors associated with the designs in the stock of rework. The stock of *Dumped cIE* and its associated inflow and outflows in Figure 148 represents this needed co-flow structure. The model equations and a description of each follow:

Dumped cIE = INTEG(Dump Coord IE - Disc Dumped cIE - RW Dumped cIE , 0)

(232)

Disc Dumped cIE = ZIDZ (Dumped cIE , IE in RW * (1 - cIE Density in RW)) * IE to DIC from RW

(233)

RW Dumped cIE = Redesign Rate * Dumped cIE Density

(234)

Dumped cIE Density = ZIDZ (Dumped cIE , Designs in Rework (RW))

(235)

From equation (232) we see that the stock of *Dumped cIE* begins with an initial value of zero and integrates a single inflow – *Dump Coord IE* and two outflows – *Disc Dumped cIE* and *RW Dumped cIE*. The inflow represents the rate at which coordinated integration errors are dumped because a build event has occurred and one of the two designs requiring rework did not complete its redesign. Each of these dumped coordinated integration errors are added to the stock of *Dumped cIE*. There are two ways that these dumped coordination errors can leave the stock. The first is when they are discovered in subsequent vehicle testing and is represented by the outflow *Disc Dumped cIE*. The second is when the design they are associated with is reworked represented by the outflow *RW Dumped cIE*.

The primary driver for the outflow *Disc Dumped cIE* is the variable *IE to DIC from RW* which represents the rate at which integration errors are moved to the stock of integration errors in designs in coordination (*IE in DIC*) from rework upon discovery in vehicle testing. The formulation for this variable was discussed in the Design Sector. However, not all of the integration errors moved to the stock of *IE in DIC* should be removed from the stock of *Dumped cIE*. In order to remove the appropriate amount, it is necessary to determine the fraction of integration errors being moved from the stock of *IE in RW* to *IE in DIC* that represent dumped coordinated integration errors. This is done in equation (233) by dividing the stock of *Dumped cIE* by the number of integration errors in rework that are not coordinated as these are the ones that are moved to *IE in DIC* upon discovery. By multiplying this fraction by the rate of IE to DIC from RW we determine the rate at which dumped coordinated integration errors are discovered and thus should be removed from the stock of *Dumped cIE*.

The second outflow – *RW Dumped cIE* – represents the rate at which integration errors that have been coordinated but then dumped are reworked along with their associated design. The outflow can be determined simply by multiplying the *Redesign Rate* by the density of dumped coordinated integration errors in the set of designs in rework (*Dumped cIE Density*). This outflow is one of the sources of phantom rework as described above.

Just as in the case of *Dumped cIE*, in order to determine the redesign rate of designs that have external integration errors we need to create a co-flow of these particular errors associated with the designs in the stock of rework. The stock of *External IE* and its associated inflow and outflows in Figure 148 represents this needed co-flow structure. The model structure for *External IE* and its formulation is completely analogous to that just described for *Dumped cIE*. The model equations follow:

External IE = INTEG(Add External IE - Disc External IE - RW External IE , 0)

(236)

Disc External IE = ZIDZ (External IE, IE in RW * (1 - cIE Density in RW)) * IE to DIC from RW

(237)

RW External IE = Redesign Rate * External IE Density

(238)

External IE Density = ZIDZ (External IE, Designs in Rework (RW))

(239)

Phantom Work Accounting

Given the amount of phantom work that is done in terms of the number of designs that have been reworked unnecessarily (phantom rework) and the number of phantom integration errors, we can calculate the number of months of designer work that are spent during the course of a development project on phantom work. This represents real work done by designers that could potentially have been avoided if the mismatch in design iteration speeds between components was eliminated and/or the timing of the build events was matched more closely to the slow iterating component.

To calculate the amount of phantom work done by designers in terms of time spent on unnecessary rework (*Phantom RW ManMonths*) we simply take the cumulative number of designs in the stock of *Phantom RW* and divide it by the *Normal Redesign Productivity* as outlined in equation (240).

Phantom RW ManMonths = Phantom RW / Normal Redesign Productivity

(240)

To calculate the amount of phantom work done in terms of the designer time spent on coordinating phantom integration errors (*Phantom IE ManMonths*) we take the cumulative number of errors in the stock of *Phantom IE* and divide it by the *Coordination Productivity*. Since coordinating an integration error requires two designers – one from each of the coupled components – we must multiply this quotient by two as outlined in equation (241) below.

Phantom IE ManMonths = (Phantom IE / Coordination Productivity) * 2

(241)

In order to calculate the total amount of designer time spent on phantom rework (*SumTotal Phantom ManMonths*) we must first sum the designer time spent on phantom rework and phantom integration errors across the components. This is done using the SUM function in VENSIM. We can then simply add these two vector sums together to determine the total amount of designer time spent on phantom rework.

SumTotal Phantom ManMonths = SUM (Phantom RW ManMonths[Component!]) +
SUM (Phantom IE ManMonths[Component!])

(242)

BUILD SWITCHES

This sector of the model handles two important structural issues in the NPD model. First, it establishes the dates of scheduled builds and turns them on and off as appropriate. Secondly, it ensures that no more than two builds are active at any point in time and provides the means for distinguishing between the two active builds - the "current build" versus the "previous build" as described in the Build Sector. The model structure and formulation for each will be discussed below.

Build Scheduling

Build scheduling determines how many builds will occur during the course of a given simulation of the new product development project and establishes the dates of those builds. During the course of the simulation, the build scheduling structure turns builds on and off as appropriate based on the dates of the scheduled builds and the current simulation time. The build scheduling structure also turns off builds and the corresponding vehicle testing when the number

parts remaining for testing falls below a given threshold. Lastly, the build scheduling structure provides accounting for some of the necessary inputs to decision processes and model structure outlined previously. For example, it tracks the amount of time until the next scheduled build which is used to calculate the *Desired Design Rate* in equation (158). A schematic of the relationship between the relevant variables needed for build scheduling is included below in Figure 149.

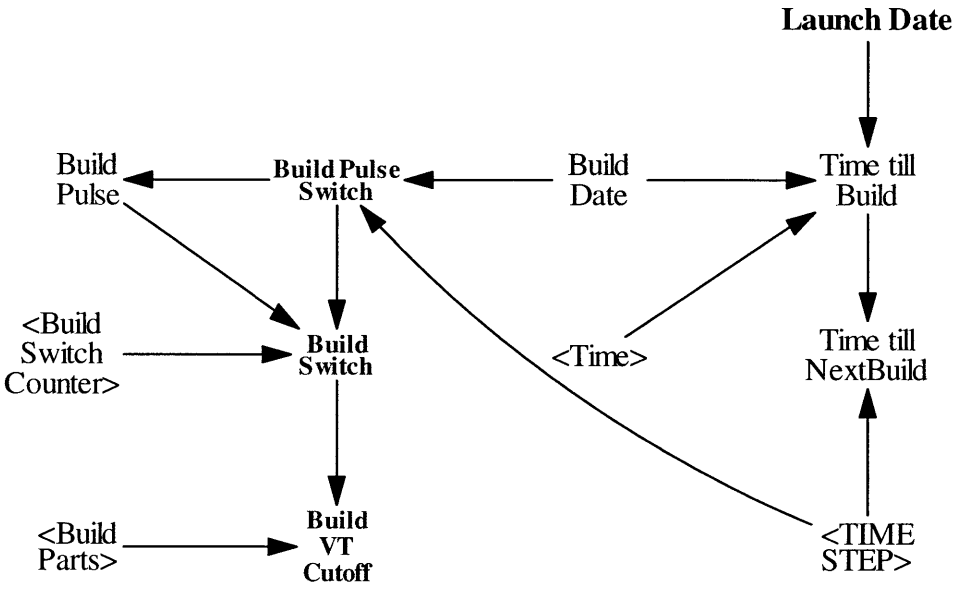


Figure 149 Build Scheduling

The most important variable in build scheduling is obviously the *Build Date*. It establishes when the builds will occur. In the NPD model, the variable *Build Date* is subscripted for the different builds allowing each build to have its own build date.

Build Date[Build] = 6, 1e+006, 1e+006, 1e+006, 18, 1e+006, 1e+006, 22, 1e+006, 1e+006

(243)

There are ten (10) build dates, separated by commas, listed in the equation for *Build Date* in equation (243). This is because the default subscripting for builds in the NPD model allows for up to ten builds listed in this order: Proto, Proto2, Proto3, Proto4, DIB, DIB2, DIB3, FPE, and FPE2. Given that the normal simulation run time is 50 months, by setting the start dates for certain builds at 1e+006 (month 1,000,000) in the base case we essentially remove these builds from the simulation and leave Proto, DIB and FPE as the three active builds. However, in model testing we are able to add builds to the simulation simply by resetting their build date to a time within the simulation time bounds.

Build Pulse Switch[Build] = PULSE (Build Date[Build] , TIME STEP)

(244)

Build Pulse = SUM (Build Pulse Switch[Build!])

(245)

The *Build Pulse Switch* pulses to a value of 1 at the time of given build's *Build Date* using the PULSE command in VENSIM as outlined in equation (244). The simulation variable TIME STEP is included in the PULSE command as the duration of the pulsed value of 1. The result is that for each subscripted build, the value of *Build Pulse Switch* is equal to zero except at the time of the *Build Date* when the value pulses to 1 for a single time step. The variable *Build Pulse* in equation (245) then sums these values across all builds resulting in a variable that pulses to a value of 1 for a single time step when any build event occurs. The variable Build Pulse is used in a number of places in the model to stop the rates of flow for stocks in both the design and build sector as previously described.

Different than the *Build Pulse Switch* which is turned on only at the instant of a build event, the *Build Switch* is turned on (i.e. equals 1) and stays on throughout the time a given build is active (i.e. while vehicle testing is ongoing); else it is equal to zero.

Build Switch = MAX (Current Build Switch , Previous Build Switch)

(246)

By definition, a build is active if it is either the current build or the previous build. Thus, a simple formulation for *Build Switch* is to take the maximum value of the *Current Build Switch* and the *Previous Build Switch* as outlined in equation (246). As the names imply, these switches are turned on (equal to 1) when a given build is active and the current or previous build respectively. These switches will be discussed in detail in the Active Builds section below.

A switch is used to cutoff vehicle testing when the number of build parts remaining to be tested falls below a certain threshold. This is done to reflect the fact that vehicle testing is shut down in the client company when further vehicle testing yields little results. This is accomplished in the model by using the IF THEN ELSE function in VENSIM to cutoff vehicle testing by setting the value of Build VT Cutoff to zero when the number of build parts remaining to be tested falls below 0.005 as outlined in equation (247).

Build VT Cutoff[Component,Build] = IF THEN ELSE (Build Parts[Component,Build] < 0.005, 0, 1)
*** Build Switch[Build]**

(247)

The use of Build Switch as a multiplier ensures that vehicle testing is shutoff for a given build whenever it is not an active build. You'll note that the *Build VT Cutoff* variable is subscripted for

both the different components and the different builds. This is because vehicle testing on a given build can be shut down for the different components separately.

The next piece of accounting handled by the Build Scheduling sector of the model is calculating the time remaining until a given build (*Time till Build*) and, from this, the time until the next build (*Time till NextBuild*).

Time till Build[Build] = MAX (0, IF THEN ELSE (Time < Build Date[Build] , Build Date[Build] - Time , Launch Date - Time))

(248)

Launch Date = 27

(249)

Time till NextBuild = MAX (TIME STEP , VMIN (Time till Build[Build!]))

(250)

The formulation for Time till Build handles a couple of scenarios. First, while the current time (i.e. simulation time) is less than the build date for a given build the time until the build is calculated simply by subtracting the current time (Time) from the Build Date. When the build date for a given build has passed (Time> Build Date), then the Time till Build is set to be the time remaining until product launch (Launch Date – Time). The launch date of the product is set at a default value of 27 months in the simulation to reflect the new product development timeline of the client company. However, when the launch date has also passed, resulting in a negative amount of time remaining, the formulation in equation (248) sets the value of *Time till Build* at 0 using the MAX function.

The variable *Time till NextBuild* represents the time remaining until the next scheduled build event. It is the variable used in setting the desired completion rates for design, redesign and coordination rates as discussed previously. It is also used in determining the fraction of designs and redesigns that are sent for bench testing as well as the fraction of designs with both component and integration errors that are sent for coordination. Calculating the time until the next build is simple in that it is merely the minimum value of the time remaining until each of the remaining builds. This is formulated in equation (250) using the VMIN function in VENSIM which takes the minimum value from the vector of *Time till Build* values across the subscript *Build.*

You'll notice that equation (250) also includes the use of the MAX function which ensures that when the returned value of the VMIN function is 0 then *Time till NextBuild* will be set at the simulation *TIME STEP.* This is necessary because the formulation of the desired work rates mentioned above (*Desired Design Rate*, *Desired Redesign Rate*, and *Desired Coordination Rate*) uses the value of *Time till NextBuild* in the denominator which would be impossible to evaluate if it were zero. This ultimately results in desired work rates that seek to complete all remaining work in a single time step once the launch date has passed.

Active Builds

In order to account for the active builds and to distinguish between the current and previous builds, the NPD model uses a separate stock which serves as a build counter. A depiction of this stock used as a counter and its associated inflow and outflow are included in Figure 150.

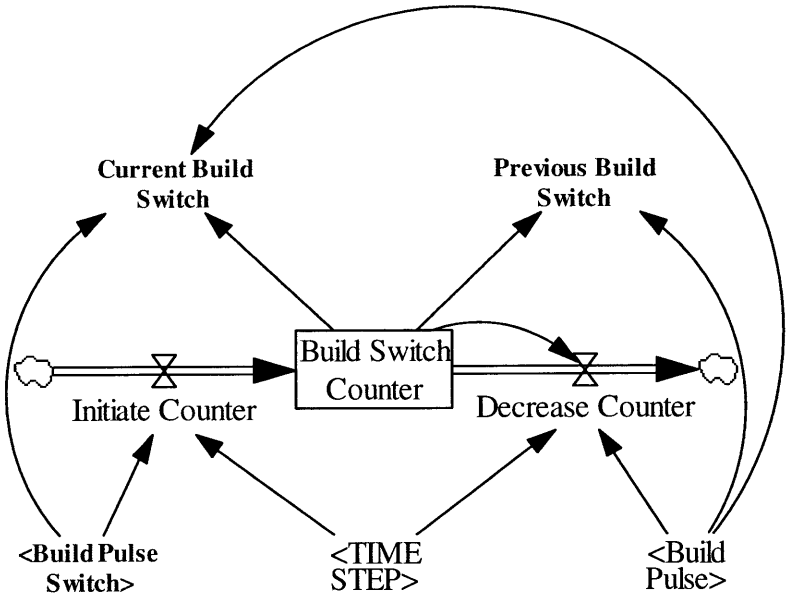


Figure 150 Build Switch Counter

There is a build counter for each build which is initiated at a value of 2 at the build date. This is done via the inflow *Initiate Counter* outlined in equation (252). The stock is depleted via the outflow *Decrease Counter* which decreases the counter by 1 at each subsequent build event (Build Pulse/TIME STEP) while the *Build Switch Counter* is greater than zero. This formulation is captured in (253). The result is a build switch counter which initiates at a value of 2 at the time of the given build, decreases to 1 at the time of the next build, and finally decreases to 0 at the second subsequent build.

Build Switch Counter[Build] = INTEG(Initiate Counter[Build] - Decrease Counter[Build] , 0)

(251)

Initiate Counter[Build] = 2 * Build Pulse Switch[Build] / TIME STEP

(252)

Decrease Counter[Build] = IF THEN ELSE (Build Switch Counter[Build] > 0, Build Pulse / TIME STEP , 0)

(253)

Given the formulation of the *Build Switch Counter*, it isn't difficult to distinguish between the current and the previous builds. Two switches are used in the NPD model that turn on when a given build is considered the current build (*Current Build Switch*) or when it is considered the previous build (*Previous Build Switch*).

Current Build Switch[Build] = MAX (Build Pulse Switch[Build] , IF THEN ELSE (Build Switch Counter[Build] - Build Pulse = 2, 1, 0))

(254)

Previous Build Switch[Build] = IF THEN ELSE (Build Switch Counter[Build] - Build Pulse = 1, 1, 0)

(255)

A given build is considered the current build from the time of its build date until the date of the next build event. A *Current Build Switch* can be formulated to reflect this definition as outlined in equation (254). By including the variable *Build Pulse Switch* in the MAX function, we ensure that the *Current Build Switch* will be turned on (set equal to 1) on the build date for a given build. This is necessary because at the instant of the build event the *Build Switch Counter* is being initiated via its inflow but the counter won't reflect its new value until the next simulation time step. Else, the *Current Build Switch* will be set at a value of 1 while the value of the term *Build Switch Counter – Build Pulse* equals two. Subtracting the value of *Build Pulse* from the *Build Switch Counter* ensures that the *Current Build Switch* turns off on the date of the next build (i.e. when *Build Pulse* next equals one). Again, this is necessary because while the Build Switch Counter is being

decremented by one at the time of the next build, the new value won't be reflected until the next time step.

A given build is considered the previous build beginning on the date of the build that follows it up until the date of the subsequent build. In other words, Build 1 is considered the previous build beginning on the date of Build 2 and up until the date of Build 3 at which time it is no longer considered an active build. A *Previous Build Switch* is formulated to reflect this definition as outlined in equation (255). In this case while the term *Build Switch Counter – Build Pulse* equals one the *Previous Build Switch* is turned on; else it is turned off. Again, the purpose for subtracting *Build Pulse* from the *Build Switch Counter* in this equation is to ensure that the switch turns off on the date of the next build event since the corresponding decrease in the counter won't be reflected until the next time step after the build event date.

APPENDIX B: MODEL EQUATION LISTING

This appendix provides a full listing of the equations that comprise the simulation model used in this research. It is intended to provide the reader with the necessary information to replicate the simulation model and the associated analysis. The simulation model was created and analyzed using the VENSIM simulation software package. A free version of this software can be downloaded at http://www.vensim.com although some of the functionality used in analyzing the model is not available on the free version of the software. The equation listing below was generated using the VENSIM documentation tool. The general format for the equation listing below is as follows:

(###) variable = equation
Units: *units*
Comment or description of variable.

****

Bench Test Fraction

****

(001) BT Fraction[Component]=(Intended Bench Test Fraction*Fixed BT Fraction Switch+Intended Bench Test Fraction*Effect of Redesign Pressure on BT Fraction[Component]*(1-Fixed BT Fraction Switch))*(1-Fix BT Perfect Switch)+Fix BT Perfect Switch

Units: fraction [0,1,0.05]

The fraction of designs that are sent for bench testing at a given point in time. This fraction is based on the Intended Bench Test Fraction and the effect of redesign pressure.

(002) Effect of Redesign Pressure on BT Fraction[Component]=Fn for Effect of Redesign Pressure on BT Fraction(Redesign Pressure[Component])

Units: Dmnl

The effect of redesign pressure on the fraction of designs that are sent for bench testing which assumes that increased redesign pressure results in a decrease in the fraction of designs sent for bench testing.

(003) Fix BT Perfect Switch=0

Units: Dmnl

Switch that when turned on sets the BT Fraction to 1 and the BT Effectiveness to 1. This means that all designs will be sent for bench testing and the testing will discover all component errors.

(004) Fn for Effect of Redesign Pressure on BT Fraction([(0,0)-(5,1)], (0,1),(0.5,1),(0.75,0.95), (0.9,0.9),(1,0.85),(1.5,0.5),(2,0.35),(3,0.275),(4,0.2525),(5,0.25),(10,0.25))

Units: Dmnl

A table function used to capture the effect of redesign pressure on the fraction of designs that are sent for bench testing which assumes that increased redesign pressure results in a decrease in the fraction of designs sent for bench testing.

(005) Intended Bench Test Fraction=0.5

Units: fraction

The fraction of designs that are intended to be bench tested prior to a build event.

****

Build Switches

****

(006) Build atAll Switch=1

Units: Dmnl

A dimensionless switch that determines whether or not builds are conducted at all in the simulation. Used for model testing purposes.

(007) Build Date[Build]=13, 1e+006, 1e+006, 1e+006, 25, 1e+006, 31, 1e+006

Units: Month [18,22,0.25]

The date (in months from the beginning of the simulation) when a given build event occurs.

(008) Build Pulse=SUM(Build Pulse Switch[Build!])

Units: Dmnl

A dimensionless switch which pulses on at the instant when any build event occurs.

(009) Build Pulse Switch[Build]=Pulse(Build Date[Build], TIME STEP)*Build atAll Switch

Units: Dmnl

A dimensionless switch which pulses on at the instant when the given build event occurs.

(010) Build Switch[Build]=max(Current Build Switch[Build],Previous Build Switch[Build])

Units: Dmnl

Dimensionless switch indicating when a given build is active or ongoing. Assumes that no more than two builds will be active at any one time.

(011) Build Switch Counter[Build]= INTEG (+Initiate Counter[Build]-Decrease Counter[Build],
0)

Units: Dmnl

Stock used as a counter to distinguish between the build events to identify which build is the newest (or current) build and which is the older (or previous) build. A value of 2 indicates that the given build is the current build. A value of 1 indicates that the given build is the previous build.

(012) Build VT Cutoff[Component,Build]=IF THEN ELSE(Build Parts[Component,Build] <0.005, 0, 1)*Build Switch[Build]

Units: Dmnl

A cutoff switch designed to shut down vehicle testing when there are near zero (less than 0.005) parts remaining to be tested.

(013) Current Build Switch[Build]=max(Build Pulse Switch[Build],IF THEN ELSE(Build Switch Counter[Build]-Build Pulse=2, 1, 0))

Units: Dmnl

Switch that inidcates when a given build is the current (or most recent) build.

(014) Decrease Counter[Build]= IF THEN ELSE(Build Switch Counter[Build]>0, Build Pulse/TIME STEP, 0)

Units: Dmnl/Month

Flow used to decrease the build switch counter for a given build while it is active.

(015) Initiate Counter[Build]=2*Build Pulse Switch[Build]/TIME STEP

Units: Dmnl/Month

Flow used to initiate the build counter with a value of 2 when a given build event first occurs.

(016) Launch Date=36

Units: Month

The scheduled date for launching the new product.

(017) Previous Build Switch[Build]= IF THEN ELSE(Build Switch Counter[Build]-Build Pulse=1, 1, 0)

Units: Dmnl

Switch that indicates when a given build is the older (or previous) build of the two active builds.

(018) Real Proto Switch=0

Units: Dmnl

Switch that when turned on activates designer reaction to Proto Builds for schedule pressure.

(019) Time till Build[Build]=max (0, IF THEN ELSE(Time<Build Date[Build], Build Date[Build]-Time, Launch Date-Time))

Units: Month

For each scheduled build event, this is the time remaining (in months) until that given build event. Once the given build has occurred, the time till build is calculated as the time until the scheduled launch date. Once the schedule launch time has occurred, the time till build takes on a value of zero.

(020) Time till NextBuild=IF THEN ELSE(Time<Build Date[DIB], Time till Build[DIB], max(TIME STEP,VMIN(Time till Build[Build!])))*(1-Real Proto Switch)+Real Proto Switch*max(TIME STEP,VMIN(Time till Build[Build!]))

Units: Month

The time remaining (in months) until the next build event or, if no additional build events are scheduled, the time remaining until the scheduled launch date.

****

Builds

****

(021) Build CE[Component,Build]= INTEG (Building CE[Component,Build]-Discovering CE[Component,Build]-Removing CE[Component,Build],0)

Units: errors

The stock of component errors associated with the stock of parts for a given build (Build Parts) that are in vehicle testing.

(022) Build CE Density[Component,Build]=zidz(Build CE[Component,Build],Build Parts[Component,Build])

Units: fraction

The density of component errors in the stock of build parts for a given component (A or B).

(023) Build CE Fraction[Component,Build]=zidz(Build CE[Component,Build],SUM(Current Build CE[Component,Build!]))*Previous Build Switch[Build]

Units: fraction

The fraction of component errors from the current (most recent) build that are identical to the component errors in the build parts still in testing from the previous build for or a given component.

(024) Build CE Ratio[Component,Build]=zidz(Build CE[Component,Build],CE in Designs[Component])

Units: fraction

The fraction of component errors in the designs for a given component that are in the stock of component errors in build parts (Build CE) for a given build.

(025) Build IE[Component,Build]= INTEG (Building IE[Component,Build]-Discovering IE[Component,Build]-Removing IE[Component,Build],0)

Units: Ierrors

The stock of integration errors associated with the stock of parts for a given build (Build Parts) that are in vehicle testing.

(026) Build IE Density[Component,Build]=zidz(Build IE[Component,Build],Build Parts[Component,Build])

Units: fraction

The density of integration errors in the stock of build parts for a given component (A or B).

(027) Build IE Fraction[Component,Build]=zidz(Build IE[Component,Build],SUM(Current Build IE[Component,Build!]))*Previous Build Switch[Build]

Units: fraction

The fraction of integration errors from the current (most recent) build that are identical to the integration errors in the build parts still in testing from the previous build for or a given component.

(028) Build IE Ratio[Component,Build]=zidz(Build IE[Component,Build],IE in Designs[Component])

Units: fraction

The fraction of integration errors in the designs for a given component that are in the stock of integration errors in build parts (Build IE) for a given build.

(029) Build Part Fraction[Component,Build]=zidz(Build Parts[Component,Build],SUM(Current Build Parts[Component,Build!]))*Previous Build Switch[Build]

Units: fraction

The fraction of build parts from the current (most recent) build that are identical to the build parts still in testing from the previous build for or a given component.

(030) Build Parts[Component,Build]= INTEG (Building Parts[Component,Build]-Parts Failing[Component,Build]-Removing Parts[Component,Build]-Validating Parts[Component,Build],0)

Units: parts

The stock of parts for a given component (A or B) for a given build (e.g. Proto, DIB or FPE) that are in vehicle testing.

(031) Building CE[Component,Build]=Build Pulse Switch[Build]*CE in Designs[Component]/TIME STEP

Units: errors/Month

The rate at which component errors are added to the stock of Build CE for a given build. In this model, the component errors are added instantaneously at the time of the build event.

(032) Building IE[Component,Build]=Build Pulse Switch[Build]*(IE in Designs[Component]/TIME STEP+Add External IE[Component])

Units: Ierrors/Month

The rate at which integration errors are added to the stock of Build IE for a given build. In this model, the integration errors are added instantaneously at the time of the build event.

(033) Building Parts[Component,Build]=Build Pulse Switch[Build]*Designs inProg[Component]/TIME STEP

Units: parts/Month

The rate at which build parts are added to the stock of build parts for a given build. In this model, the build parts are added instantaneously at the time of the build event. This rate assumes that only designs that are in progress (at a minimum the initial design has been done) can be built.

(034) Bwd Remove CE[Component,Build]=(SUM(cFail VT[Component,Build!])-cFail VT[Component,Build])*Build CE Fraction[Component,Build]*Previous Build Switch[Build]

Units: errors/Month

An auxiliary variable that represents the rate at which component errors are removed from the stock of Build CE for a given build (when it is considered to be the previous build) because identical component errors have been identified in a more recent (the current) build.

(035) Bwd Remove IE[Component,Build]=(SUM(iFail VT[Component,Build!])-iFail VT[Component,Build])*Build IE Fraction[Component,Build]*Previous Build Switch[Build]

Units: Ierrors/Month

An auxiliary variable that represents the rate at which integration errors are removed from the stock of Build IE for a given build (when it is considered to be the previous build) because identical integration errors have been identified in a more recent (the current) build.

(036) Bwd Remove Parts[Component,Build]=(SUM(netFail VT[Component,Build!])-netFail VT[Component,Build])*Build Part Fraction[Component,Build]*Previous Build Switch[Build]

Units: parts/Month

An auxiliary variable that represents the rate at which parts are removed from the stock of build parts for a given build (when it is considered to be the previous build) because identical parts have been tested in a more recent (the current)

build.

(037) Current Build CE[Component,Build]= Build CE[Component,Build]*Current Build Switch[Build]

Units: errors

An auxilliary variable representing the stock of component errors associated with the current (most recent) vehicle build.

(038) Current Build IE[Component,Build]=Build IE[Component,Build]*Current Build Switch[Build]

Units: Ierrors

An auxilliary variable representing the stock of integration errors associated with the current (most recent) vehicle build.

(039) Current Build Parts[Component,Build]=Build Parts[Component,Build]*Current Build Switch[Build]

Units: parts

An auxilliary variable representing the stock of build parts associated with the current (most recent) vehicle build.

(040) Discovering CE[Component,Build]=cFail VT[Component,Build]

Units: errors/Month

The rate at which component errors are removed from the stock ofBuild CE for a given build as they are identified in vehicle testing.

(041) Discovering IE[Component,Build]=iFail VT[Component,Build]

Units: Ierrors/Month

The rate at which integration errors are removed from the stock of Build IE for a given build as they are identified in vehicle testing.

(042) Fwd Remove CE[Component,Build]=(SUM(cFail VT[Component,Build!])-cFail VT[Component,Build])*Current Build Switch[Build]

Units: errors/Month

An auxiliary variable that represents the rate at which component errors are removed from the stock of Build CE for a given build (when it is considered to be the current build)because identical component errors have been identified in a previous build.

(043) Fwd Remove IE[Component,Build]=(SUM(iFail VT[Component,Build!])-iFail VT[Component,Build])*Current Build Switch[Build]

Units: Ierrors/Month

An auxiliary variable that represents the rate at which integration errors are removed from the stock of Build IE for a given build (when it is considered to be the current build) because identical integration errors have been identified in a previous build.

(044) Fwd Remove Parts[Component,Build]=(SUM(netFail VT[Component,Build!])-netFail VT[Component,Build])*Current Build Switch[Build]

Units: parts/Month

An auxiliary variable that represents the rate at which parts are removed from the stock of build parts for a given build (when it is considered to be the current build) because identical parts have been tested in a previous build.

(045) Parts Failing[Component,Build]=netFail VT[Component,Build]

Units: parts/Month

The rate at which parts are removed from the stock of Build Parts because they failed vehicle testing because either a component error, an integration error or both was discovered.

(046) Redesign CE[Component]=Redesign Rate[Component]*CE Density in RW[Component]

Units: errors/Month

An auxiliary variable representing the rate at which designs with component errors are redesigned.

(047) Redesign IE[Component]=Redesign Rate[Component]*IE Density in RW[Component]

Units: Ierrors/Month

An auxiliary variable representing the rate at which designs with integration errors are redesigned.

(048) Remove Redesigned CE[Component,Build]= Redesign CE[Component]*Build CE Ratio[Component,Build]*Build Switch[Build]

Units: errors/Month

The rate at which component errors are removed from the stock of Build CE because the part with which the error is associated has been redesigned since the build event and the start of vehicle testing.

(049) Remove Redesigned IE[Component,Build]=Redesign IE[Component]*Build IE Ratio[Component,Build]*Build Switch[Build]

Units: Ierrors/Month

The rate at which integration errors are removed from the stock of Build IE because the part with which the error is associated has been redesigned since the build event and the start of vehicle testing.

(050) Removing CE[Component,Build]=Remove Redesigned CE[Component,Build]+Bwd Remove CE[Component,Build]+Fwd Remove CE[Component,Build]

Units: errors/Month

The rate at which component errors are removed from the stock of Build CE in vehicle testing from a given build because they have been discovered on a previous build or a subsequent build.

(051) Removing IE[Component,Build]= Remove Redesigned IE[Component,Build]+Bwd Remove IE[Component,Build]+Fwd Remove IE[Component,Build]

Units: Ierrors/Month

The rate at which integration errors are removed from the stock of Build IE in vehicle testing from a given build because they have been discovered on a previous build or a subsequent build.

(052) Removing Parts[Component,Build]=Bwd Remove Parts[Component,Build]+Fwd Remove Parts[Component,Build]

Units: parts/Month

The rate at which build parts are removed from the stock of build parts in vehicle testing (Build Parts) from a given build because they have been tested on a previous build or a subsequent build.

(053) Validating Parts[Component,Build]=VT TPut[Component,Build]-Parts Failing[Component,Build]

Units: parts/Month

The rate at which parts are removed from the stock of Build Parts during vehicle testing having been validated - that is, found to contain neither a component or integration error.

(054) VT TPut[Component,Build]=(Build Parts[Component,Build]/Vehicle Test Time[Component])*Build VT Cutoff[Component,Build]*(1-Build Pulse)

Units: parts/Month

An auxiliary variable representing the throughput rate of build parts in vehicle testing.

****

CE Accounting

****

(055) "% CEo in DR (VT)"[Component]=zidz(CEonly Errors in DR[Component],"Total CEo (VT)"[Component])

Units: fraction

The percent of designs for a given component (for vehicletesting purposes) that have a component error only that reside in the stock of Designs Released.

(056) "% CEo in RW (VT)"[Component]=zidz(CEonly Errors in RW[Component],"Total CEo (VT)"[Component])

Units: fraction

The percent of designs for a given component (for vehicle testing purposes) that have a component error only that reside in the stock of Designs in Rework.

(057) CE Carried[Component]=Carry CE to DIT[Component]+Carry CE to DR[Component]

Units: errors/Month

The rate at which component errors are carried over either by not fixing a known component error or redesigning a design with an unknown component error.

(058) CE Density in DIC[Component]=zidz(CE in DIC[Component],"Designs in Coordination (DIC)"[Component])

Units: fraction

The density of component errors in the stock of Designs in Coordination for a given component.

(059) CE Density in DIT[Component]=zidz(CE in DIT[Component],"Designs in Test (DIT)"[Component])

Units: fraction

The density of component errors in the stock of Designs in Test (the queue for bench testing) for a given component.

(060) CE Density in DR[Component]=zidz(CE in DR[Component],"Designs Released (DR)"[Component])

Units: fraction

The density of component errors in the stock of Designs Released for a given component.

(061) CE Density in RW[Component]=zidz(CE in RW[Component], "Designs in Rework (RW)"[Component])

Units: fraction

The density of component errors in the stock of Designs in Rework for a given component.

(062) CE Generated[Component]=CErrors from Design[Component]+CErrors from Redesign[Component]

Units: errors/Month

The rate at which component errors are generated either through new design or redesign for a given component.

(063) CE in Designs[Component]=CE in DIT[Component]+CE in RW[Component]+CE in DR[Component]+CE in DIC[Component]

Units: errors

An auxiliary variable representing the total number of component errors in designs in progress (i.e. in testing, in rework, in coordination, and released) for a given component.

(064) CEonly Errors in DR[Component]="Designs Released (DR)"[Component]*CE Density in DR[Component]-"Designs Released (DR)"[Component]*CE Density in DR[Component]*IE Density in DR[Component]

Units: errors

The number of designs in the stock of Designs in Released that have a component error ONLY for a given component (A or B).

(065) CEonly Errors in RW[Component]="Designs in Rework (RW)"[Component]*CE Density in RW[Component]-"Designs in Rework (RW)"[Component]*CE Density in RW[Component]*IE Density in RW[Component]

Units: errors

The number of designs in the stock of Designs in Rework that have a component error ONLY for a given component (A or B).

(066) Cumulative CE[Component]=Cumulative CE Generated[Component]+Cumulative CE Carried[Component]

Units: errors

The cumulative number of component errors that are generated and/or carried over throughout the NPD project for a given component.

(067) Cumulative CE Carried[Component]= INTEG (CE Carried[Component],0)

Units: errors

The cumulative number of component errors that are carried over throughout the NPD project for a given component.

(068) Cumulative CE Generated[Component]= INTEG (CE Generated[Component],0)

Units: errors

The cumulative number of new component errors that are generated throughout the NPD project for a given component.

(069) SumTotal CE in Designs=SUM(CE in Designs[Component!])

Units: errors

An auxiliary variable representing the total number of component errors in designs in progress (i.e. in testing, in rework, in coordination, and released) across all components.

(070) SumTotal Cumulative CE=SUM(Cumulative CE[Component!])

Units: errors

The cumulative number of component errors that are generated and/or carried over throughout the NPD project across all components.

(071) "Total CEo (VT)"[Component]=CEonly Errors in DR[Component]+CEonly Errors in RW[Component]

Units: errors

The total number of designs with a component error ONLY in designs in progress for Vehicle Testing purposes (which means that the designs with component errors only in DIT do not count since they are not part of vehicle testing) for a given component (A or B). By definition, designs that are in coordination have an integration error so therefore cannot have a component error only.

****

Clean Designs

****

(072) Clean Designs[Component]=Clean DIT[Component]+Clean DR[Component]

Units: designs

The total number of "clean" designs for a given component - meaning that they do not have a component error or an integration error, nor are they associated with a coupled design with an integration error that has not yet been discovered or coordinated. Designs that are

in the stock of Designs in Rework or Designs in Coordination by definition are not "clean" because they require rework.

(073) Clean DIT[Component]="Designs in Test (DIT)"[Component]-("Designs in Test (DIT)"[Component]*CE Density in DIT[Component]+"Designs in Test (DIT)"[Component]*IE Density in DIT[Component]-"Designs in Test (DIT)"[Component]*CE Density in DIT[Component]*IE Density in DIT[Component])-uErw in DIT[Component]

Units: designs

An auxiliary variable representing the number of designs in the stock of Designs in Test that are considered "clean" - meaning that they do not have a component error or an integration error, nor are they associated with a coupled design with an integration error that has not yet been discovered or coordinated.

(074) Clean DR[Component]="Designs Released (DR)"[Component]-("Designs Released (DR)"[Component]*CE Density in DR[Component]+"Designs Released (DR)"[Component]*IE Density in DR[Component]-"Designs Released (DR)"[Component]*CE Density in DR[Component]*IE Density in DR[Component])-uErw in DR[Component]

Units: designs

An auxiliary variable representing the number of designs in the stock of Designs Released that are considered "clean" - meaning that they do not have a component error or an integration error, nor are they associated with a coupled design with an integration error that has not yet been discovered or coordinated.

(075) SumTotal Clean Designs=SUM(Clean Designs[Component!])

Units: designs

The sum total number of designs across components that are error free at any given point in time.

(076) uErw in DIT[a]=unCoordinated IE[b]*External RW Probability*"% IEc in DIT"[a]*(1-CE Density in DIT[a])

uErw in DIT[b]=unCoordinated IE[a]*External RW Probability*"% IEc in DIT"[b]*(1-CE Density in DIT[b])

Units: designs

Auxiliary variable that represents the expected amount of external rework for otherwise clean designs in the stock of Designs in Test for a given component from integration errors in coupled designs that have not yet been discovered and/or coordinated. As such these designs are not considered "clean" or error free since they are expected to require redesign.

(077) uErw in DR[a]=unCoordinated IE[b]*External RW Probability*"% IEc in DR"[a]*(1-CE Density in DR[a])

uErw in DR[b]=unCoordinated IE[a]*External RW Probability*"% IEc in DR"[b]*(1-CE Density in DR[b])

Units: designs

Auxiliary variable that represents the expected amount of external rework for otherwise clean designs in the stock of Designs Released for a given component from integration errors in coupled designs that have not yet been discovered and/or coordinated. As such these designs are not considered "clean" or error free since they are expected to require redesign.

(078) unCoordinated IE[Component]=IE in Designs[Component]-Coordinated IE in RW[Component]

Units: Ierrors

An auxiliary variable that represents the number of integration errors for a given component that have not yet been discovered and/or coordinated.

****

Control

****

Simulation Control Parameters

(079) FINAL TIME = 40

Units: Month

The final time for the simulation.

(080) INITIAL TIME = 0

Units: Month

The initial time for the simulation.

(081) SAVEPER = TIME STEP

Units: Month [0,?]

The frequency with which output is stored.

(082) TIME STEP = 0.0625

Units: Month [0,?]

The time step for the simulation.

****

Coordination Fraction

****

(083) Coordination Fraction[Component]=Intended Coordination Fraction*Fixed Coord Fraction Switch+Intended Coordination Fraction*Effect of Coordination Pressure on Coordination Fraction[Component]*(1-Fixed Coord Fraction Switch)

Units: fraction

The fraction of designs/parts discovered to have both an integration and a component error that are sent for coordination prior to redesign.

(084) Effect of Coordination Pressure on Coordination Fraction[Component]=Fn for Effect of Coordination Pressure on Coordination Fraction(Coordination Pressure[Component])

Units: Dmnl

The effect of coordination pressure on the fraction of designs with both types of errors that are sent for coordination prior to redesign. The assumption is that fewer designs will be sent for coordination prior to rework as the amount of coordination pressure builds.

(085) Fixed Coord Fraction Switch=0

Units: Dmnl [0,1,1]

A switch that when turned on fixes the fraction of designs with both types of errors (component and integration) that are sent for coordination prior to redesign at the Intended Coordination Fraction essentially turning off the effect of redesign time available on the Coordination Fraction.

(086) Fn for Effect of Coordination Pressure on Coordination Fraction([(0,0)-(5,1)],(0,1),(0.25,1),(0.5,0.95),(1,0.75),(2,0.25),(3,0.1),(4,0.025),(5,0),(10,0.25))

Units: Dmnl

A table function used to capture the effect of coordination pressure on the fraction of designs with both types of errors (component and integration) that are sent for coordination prior to redesign. The assumption is that fewer designs will be sent for coordination prior to rework as the amount of coordination pressure builds.

(087) Intended Coordination Fraction=1

Units: fraction

The fraction of designs discovered to have both types of errors (component and integration) that are intended to be sent for coordination prior to being redesigned.

****

Design Accounting

****

(088) "% Both in DR (VT)"[Component]=zidz(Both Errors in DR[Component],"Total BothErrors (VT)"[Component])

Units: fraction

The percent of designs for a given component (for vehicle testing purposes) that have both types of errors (component and integration) that are in the stock of Designs Released.

(089) "% Both in RW (VT)"[Component]=zidz(Both Errors in RW[Component],"Total BothErrors (VT)"[Component])

Units: fraction

The percent of designs for a given component (for vehicle testing purposes) that have both types of errors (component and integration) that are in the stock of Designs in Rework.

(090) Both Errors in DIC[Component]="Designs in Coordination (DIC)"[Component]*CE Density in DIC[Component]*IE Density in DIC[Component]

Units: designs

The number of designs for a given component that have both types of errors (component and integration) in the stock of Designs in Coordination.

(091) Both Errors in DR[Component]="Designs Released (DR)"[Component]*CE Density in DR[Component]*IE Density in DR[Component]

Units: designs

The number of designs for a given component that have both types of errors (component and integration) in the stock of Designs in Released.

(092) Both Errors in RW[Component]="Designs in Rework (RW)"[Component]*CE Density in RW[Component]*IE Density in RW[Component]

Units: designs

The number of designs for a given component that have both types of errors (component and integration) in the stock of Designs in Rework.

(093) Cumulative Designs[Component]= INTEG (Designing[Component],0)

Units: designs

The cumulative number of designs (including new designs and redesigns) completed by designers for a given component during the course of a new product development project.

(094) Designing[Component]=New Design Rate[Component]+Redesign Rate[Component]

Units: designs/Month

The rate at which designs (both new designs and redesigns) are being completed and added to the stock of cumulative designs for a given component (A or B).

(095) Designs inProg[Component]="Designs in Rework (RW)"[Component]+"Designs in Test (DIT)"[Component]+"Designs Released (DR)"[Component]+"Designs in Coordination (DIC)"[Component]

Units: designs

An auxiliary variable representing the total number of designs in progress (in test, in rework, in coordination or released) for a given component (A or B) at any given point in time.

(096) Fraction of Designs inProg[Component]=zidz(Designs inProg[Component],Total Designs[Component])

Units: fraction

The fraction of designs in progress for a given component. This includes designs in any state of design other than "designs to be done."

(097) SumTotal Cumulative Designs=SUM(Cumulative Designs[Component!])

Units: designs

The sum total of designs (including new designs and redesigns) completed by designers across all components.

(098) SumTotal Designs in Progress=SUM(Designs inProg[Component!])

Units: designs

The total number of designs in progress (in test, in rework, in coordination or released) across all components at any given point in time.

(099) "Total BothErrors (VT)"[Component]=Both Errors in DR[Component]+Both Errors in RW[Component]+Both Errors in DIC[Component]

Units: designs

The number of designs for vehicle testing purposes (in rework, in coordination, or released) for a given component that have both types of errors (component and integration).

(100) Total Designs[Component]=Designs inProg[Component]+Designs to Be Done[Component]

Units: designs

The total number of designs for a given component including those in progress and those still to be done. This is a fixed number for each component.

****

Designer Allocation

****

(101) Component Designers Available for Coordination[Component]=IF THEN ELSE(Initial Allocation of Component Designers for Coordination[Component]<0.25 , 0, Initial Allocation of Component Designers for Coordination[Component])

Units: designers

The number of designers available for coordination for a given component based on the number of designers available after new design work is accounted for and designers have been allocated between redesign work and coordination work.

(102) Coordination Priority=0.5

Units: fraction

A fraction (0 to 1) that represents the relative priority placed on coordination with respect to redesign activities. A value of 0.5 indicates that coordination work and redesign work are equally weighted. A value of 1 indicates that coordination work takes full priority over redesign work. This priority level is used to allocate designers between coordination and redesign work.

(103) Designers Available after Coordination[Component]=max(Designers Available after Design[Component]-Designers Working on Coordination[Component],0)

Units: designers

The number of designers available for redesign after designers have been allocated for design and coordination work.

(104) Designers Available after Design[Component]= max(Total Designers[Component]-Designers Working on Design[Component],0)

Units: designers

The number of designers available for other work (coordination and redesign) after designers have been allocated for design work.

(105) Designers Available for Coordination= VMIN(Component Designers Available for Coordination[Component!])

Units: designers

The number of designer PAIRS available for coordination based on the number of designers available from each component after new design work is accounted for and designers have been allocated between redesign work and coordination work.

(106) Designers Available for Redesign[Component]=IF THEN ELSE(Designers Available after Coordination[Component]<0.25, 0, Designers Available after Coordination[Component])

Units: designers

The number of designers available for redesign after designers have been allocated for design work and coordination.

(107) Designers Working on Coordination[Component]=SUM(Coordinating Designs[Component!])/AVG Coordination Productivity[Component]

Units: designers

The number of designers for a given component working on coordination of integration errors (for either the given component or the coupled component) at any given point in time.

(108) Designers Working on Design[Component]=zidz(New Design Rate[Component],Design Productivity[Component])

Units: designers

The number of designers working on design work at any given point in time.

(109) Designers Working on Redesign[Component]=zidz(Redesign Rate[Component],Redesign Productivity[Component])

Units: designers

The number of designers working on redesign work at any given point in time.

(110) Desired Designers for Coordination=SUM(Desired Coordination Rate[Component!])/(SUM(AVG Coordination Productivity[Component!])/2)

Units: designers

The number of designers desired in order to complete all of the coordination required prior to the next build event based on the normal coordination productivity.

(111) Desired Designers for Redesign[Component]=Desired Redesign Rate[Component]/AVG Redesign Productivity[Component]

Units: designers

The number of designers desired in order to complete all of the designs in rework prior to the next build event if each designer works at the normal redesign productivity rate.

(112) Initial Allocation of Component Designers for Coordination[Component]=Designers Available after Design[Component]*Relative Demand for Designers for Coordination[Component]

Units: designers

The initial number of designers allocated for coordination given the relative demand for designers to do coordination as opposed to redesign work.

(113) Relative Demand for Designers for Coordination[Component]=zidz(Desired Designers for Coordination*Coordination Priority,Desired Designers for Coordination*Coordination Priority+(1-Coordination Priority)*Desired Designers for Redesign[Component])

Units: Dmnl

The relative demand for designers to do coordination work versus redesign work. This relative demand is used to allocate designers between the two tasks.

****

Designs

****

(114) Add Coord IE[Component]=Coordinating IE[Component]

Units: Ierrors/Month

The rate at which coordinated integration errors are added to the stock of Coordinated IE. This rate is wholly determined by the rate of Coordinating IE.

(115) Add eRW[a]=Coordinating Designs[b]*External RW Probability

Add eRW[b]=Coordinating Designs[a]*External RW Probability

Units: designs/Month

The rate at which designs requiring rework are added to the stock of External RW. The rate is based on the rate of coordinating designs and the External RW probability (i.e. the probability of both designs from coupled components with an integration error requiring rework).

(116) Add External IE[a]=max(Excess eRW[a]-Unfixed cIE[b]*External RW Probability,0)/TIME STEP*Build Pulse

Add External IE[b]=max(Excess eRW[b]-Unfixed cIE[a]*External RW Probability,0)/TIME STEP*Build Pulse

Units: Ierrors/Month

An auxiliary variable representing the rate at which integration errors are added to the stock of IE in RW during a build event because the given component did not complete required external rework associated with a previously coordinated integration error when the coupled component did complete its required rework.

(117) Add kCE[Component]= BT CError Discovery Rate[Component]+CE to RW from DIC[Component]+VT ce2rw from DR[Component]+VT ce2rw from RW[Component]*(1-Fraction of CE in RW Known[Component])

Units: errors/Month

The rate at which known component errors are added to the stock of Known CE in RW based on the discovery of component errors through bench testing and vehicle testing and the coordination of designs with known component errors.

(118) Add UcIE[Component]=Redesigning Coord IE[Component]-Fix IE[Component]

Units: Ierrors/Month

The rate at which designs with coordinated Integration Errors are redesigned without fixing the integration error and are added to the stock of unfixed coordinated integration errors (UcIE).

(119) Bench Test TPut[Component]=BT Switch*(1-Build Pulse)*"Designs in Test (DIT)"[Component]/Bench Test Time[Component]

Units: designs/Month

The throughput of designs undergoing bench testing.

(120) BT CError Discovery Rate[Component]=(CE Density in DIT[Component]*Bench Test TPut[Component]*BT Effectiveness*BT Effectiveness Switch+(1-BT Effectiveness Switch)*CE Density in DIT[Component]*Bench Test TPut[Component])*(1-Fix BT Perfect Switch)+CE Density in DIT[Component]*Bench Test TPut[Component]*Fix BT Perfect Switch

Units: designs/Month

The rate at which componenet errors are discovered in designs during bench testing.

(121) BT Effectiveness=0.5

Units: fraction [0,1,0.05]

The effectiveness of bench testing at finding component errors. The variable can take on a value between 0 and 1 where 1 represents perfect testing.

(122) BT Effectiveness Switch=1

Units: Dmnl

Switch used to turn on the bench test effectiveness variable. When it it switched off, bench test effectiveness is assumed to be 1 (i.e. perfect testing).

(123) BT Switch=1

Units: Dmnl [0,1,1]

Switch used to turn bench testing on and off.

(124) Carry CE to DIT[Component]=(Redesign Rate[Component]*CE Density in RW[Component]-Fix CE[Component])*BT Fraction[Component]

Units: errors/Month

The rate at which component errors are "carried over" to CE in DIT. This means the rate at which designs with component errors are redesigned where the component errors are not fixed.

(125) Carry CE to DR[Component]=(Redesign Rate[Component]*CE Density in RW[Component]-Fix CE[Component])*(1-BT Fraction[Component])

Units: errors/Month

The rate at which component errors are "carried over" to CE in DR. This means that designs with component errors are redesigned but the component errors are not fixed and released without bench testing.

(126) Carry IE to DIT[Component]=(Redesign Rate[Component]*IE Density in RW[Component]-Fix IE[Component])*BT Fraction[Component]

Units: Ierrors/Month

The rate at which integration errors are "carried over" to IE in DIT. This means the rate at which designs with integration errors are redesigned where the integration errors are not fixed.

(127) Carry IE to DR[Component]=(Redesign Rate[Component]*IE Density in RW[Component]-Fix IE[Component])*(1-BT Fraction[Component])

Units: Ierrors/Month

The rate at which component errors are "carried over" to CE in DR. This means that designs with component errors are redesigned but the component errors are not fixed and released without bench testing.

(128) CE Created in DR[Component]= (CErrors from Design[Component]+CErrors from Redesign[Component])*(1-BT Fraction[Component])

Units: errors/Month

The rate at which component errors are generated through design and redesign and added to the stock of component errors in designs released - without going through bench testing.

(129) CE in DIC[Component]= INTEG (CE to DIC from DR[Component]+CE to DIC from RW[Component]-CE to RW from DIC[Component],0)

Units: errors

The stock of component errors associated with the stock of Designs in Coordination for a given component.

(130) CE in DIT[Component]= INTEG (Design CE[Component]+Carry CE to DIT[Component]-CE to RW from DIT[Component]-Release CE[Component],0)

Units: errors

The stock of component errors associated with the stock of Designs in Test (in the queue for bench testing) for a given component.

(131) CE in DR[Component]= INTEG (+Release CE[Component]-CE to RW from DR[Component]+Carry CE to DR[Component]+CE Created in DR[Component]-CE to DIC from DR[Component],0)

Units: errors

The stock of component errors associated with the stock of Designs Released for a given component.

(132) CE in RW[Component]= INTEG (+CE to RW from DIT[Component]+CE to RW from DR[Component]-Carry CE to DIT[Component]-Fix CE[Component]-Carry CE to DR[Component]-CE to DIC from RW[Component]+CE to RW from DIC[Component], 0)

Units: errors

The stock of component errors associated with the stock of Designs in Rework for a given component.

(133) CE to DIC from DR[Component]=VT ce2dic from DR[Component]

Units: errors/Month

The rate at which component errors are moved to the stock of component errors in designs in coordination (CE in DIC) from the stock of component errors in designs released (CE in DR) after vehicle testing.

(134) CE to DIC from RW[Component]=VT ce2dic from RW[Component]

Units: errors/Month

The rate at which component errors are moved to the stock of component errors in designs in coordination (CE in DIC) from rework after vehicle testing.

(135) CE to RW from DIC[Component]=Coordinating Designs[Component]*CE Density in DIC[Component]

Units: errors/Month

The rate at which component errors in Designs in Coordination are moved to the stock of CE in RW as designs are coordinated.

(136) CE to RW from DIT[Component]=BT CError Discovery Rate[Component]+Ext RW from DIT[Component]*CE Density in DIT[Component]

Units: errors/Month

The rate at which component errors are moved to the stock of component errors in rework either through the discovery of component errors in bench testing or in identifying designs identified for rework through coordination (Ext RW from DIT).

(137) CE to RW from DR[Component]= VT ce2rw from DR[Component]+Ext RW from DR[Component]*CE Density in DR[Component]

Units: errors/Month

The rate at which component errors are moved from designs released to the stock of component errors in rework (CE in RW) either through the discovery of component errors in vehicle testing or in identifying designs identified for rework through coordination (Ext RW from DIT).

(138) CErrors from Design[Component]=New Design Rate[Component]*(1-Design CQ[Component])

Units: errors/Month

The rate at which component errors are generated through new design.

(139) CErrors from Redesign[Component]=Redesign Rate[Component]*(1-Redesign CQ[Component])*(1-CE Density in RW[Component])

Units: errors/Month

The rate at which component errors are generated through redesign.

(140) cIE Density in RW[Component]=zidz(Coordinated IE in RW[Component],"Designs in Rework (RW)"[Component])

Units: fraction

The density (or fraction) of designs in rework that have integration errors that have been coordinated. Only coordinated integration errors can be fixed in redesign.

(141) Coordinated IE in RW[Component]= INTEG (Add Coord IE[Component]-Redesigning Coord IE[Component]-Dump Coord IE[Component,0)

Units: Ierrors

The stock of integration errors for a given component that have been coordinated.

(142) Coordinating Designs[Component]=min(Coordination Rate from People[Component],"Designs in Coordination (DIC)"[Component]/Min Coordination Time)*(1-Build Pulse)*Coordination Switch

Units: designs/Month

The rate at which designs with integration errors are coordinated by the required designers. This rate adds to the stock of rework required.

(143) Coordinating IE[Component]=Coordinating Designs[Component]*IE Density in DIC[Component]

Units: Ierrors/Month

The rate at which integration errors are coordinated (not yet fixed) and added to the stock of integration errors in rework (IE in RW).

(144) Coordination Switch=1

Units: Dmnl [0,1,1]

Switch used to turn coordination on (1) and off (0).

(145) Design CE[Component]=(CErrors from Design[Component]+CErrors from Redesign[Component])*BT Fraction[Component]

Units: errors/Month

The rate at which component errors are generated through design and redesign and added to the stock of component errors in designs in test (CE in DIT).

(146) Design IE[Component]=(IErrors from Design[Component]+IErrors from Redesign[Component])*BT Fraction[Component]

Units: Ierrors/Month

The rate at which integration errors are generated through design and redesign and added to the stock of integration errors in designs in test (IE in DIT).

(147) "Designs in Coordination (DIC)"[Component]= INTEG (DR to DIC[Component]-Coordinating Designs[Component]+RW to DIC[Component],0)

Units: designs

Designs with an identified integration error that are in the queue for coordination between designers to identify required rework to fix the integration error.

(148) "Designs in Rework (RW)"[Component]= INTEG (BT CError Discovery Rate[Component]-RW to DIT[Component]+DIT to RW[Component]+Coordinating Designs[Component]+DR to RW[Component]-Redesigns not Bench Tested[Component]-RW to DIC[Component],0)

Units: designs

Stock of designs that are in the queue for rework.

(149) "Designs in Test (DIT)"[Component]= INTEG (+New Designs to DIT[Component]+RW to DIT[Component]-BT CError Discovery Rate[Component]-Release Rate[Component]-DIT to RW[Component],0)

Units: designs

Stock of designs that are in the queue for bench testing.

(150) "Designs Released (DR)"[Component]= INTEG (New Designs not Bench Tested[Component]+Release Rate[Component]-DR to DIC[Component]-DR to RW[Component]+Redesigns not Bench Tested[Component],0)

Units: designs

Stock of designs that have been released by the designers awaiting the next vehicle build event.

(151) Designs to Be Done[Component]= INTEG (-New Designs to DIT[Component]-New Designs not Bench Tested[Component],Initial Designs[Component])

Units: designs

The stock of paper designs that still need to be completed for a given component.

(152) DIT to RW[Component]=Ext RW from DIT[Component]

Units: designs/Month

The rate at which designs awaiting bench testing are sent directly to rework based on the external discovery of rework (i.e. coordinating an integration error).

(153) DR to DIC[Component]=VT dr2dic[Component]

Units: designs/Month

The rate at which designs released are sent to the stock of designs requiring coordination based on the discovery of an integration error in vehicle testing.

(154) DR to RW[Component]= VT dr2rw[Component]+Ext RW from DR[Component]

Units: designs/Month

The rate at which designs released are sent to the stock of designs needing rework based on the discovery of errors through vehicle testing and/ or identifying required rework from coordinating integration errors.

(155) Dump Coord IE[Component]=(max((Coordinated IE in RW[Component]-Indicated cIE[Component]),0)/TIME STEP)*Build Pulse

Units: Ierrors/Month

This variable reflects the number of integration errors for a given component that have been coordinated but not yet redesigned at the time of the build where the coupled design from the other component that also required rework has been redesigned.

(156) Dump DIT[Component]="Designs in Test (DIT)"[Component]*Build Pulse/TIME STEP

Units: designs/Month

This variable releases the remaining stock of designs in the bench testing queue at the time of a build.

(157) Dump eRW[Component]=(Excess eRW[Component]/TIME STEP)*Build Pulse

Units: designs/Month

This variable releases the remaining stock of external rework (External RW) at the time of a build.

(158) Dump UcIE[Component]=(Unfixed cIE[Component]/TIME STEP)*Build Pulse

Units: Ierrors/Month

This variable releases the remaining stock of coordinated integration errors that have been redesigned but still have an integration error at the time of a build.

(159) eRW Density in RW[Component]=zidz(External RW[Component],"Designs in Rework (RW)"[Component])

Units: fraction

The density of designs in the stock of rework that are from external rework (a coordinated integration error).

(160) Excess eRW[Component]=max (External RW[Component]-Indicated eRW[Component],0)

Units: designs

(161) External RW[Component]= INTEG (Add eRW[Component]-Dump eRW[Component]-Redesigning eRW[Component],0)

Units: designs

The stock of rework that is due to an integration error (residing in a coupled component design) discovered in vehicle testing and identified for rework through coordination.

(162) Fix CE[Component]=Redesign Rate[Component]*kCE Density in RW[Component]*Redesign CQ[Component]

Units: errors/Month

The rate at which component errors are fixed as Designs in Rework are redesigned. This rate assumes that only known component errors can be fixed.

(163) Fix IE[Component]=Redesign Rate[Component]*cIE Density in RW[Component]*Redesign IQ[Component]

Units: Ierrors/Month

The rate at which integration errors are fixed as Designs in Rework are redesigned. This rate assumes that only known and coordinated integration errors can be fixed.

(164) Fraction of CE in RW Known[Component]=zidz(Known CE in RW[Component],CE in RW[Component])

Units: fraction

The fraction of component errors associated with designs in rework (CE in RW) that are known.

(165) Fraction of IE in RW Coordinated[Component]=zidz(Coordinated IE in RW[Component],IE in RW[Component])

Units: fraction

The fraction of integration errors associated with designs in rework (IE in RW) that have been coordinated.

(166) IE Created in DR[Component]=(IErrors from Design[Component]+IErrors from Redesign[Component])*(1-BT Fraction[Component])

Units: Ierrors/Month

The rate at which integration errors are generated through design and redesign and added to the stock of integration errors in designs released.

(167) IE in DIC[Component]= INTEG (IE to DIC from DR[Component]+IE to DIC from RW[Component]-Coordinating IE[Component],0)

Units: Ierrors

The stock of integration errors associated with the stock of Designs in Coordination for a given component.

(168) IE in DIT[Component]= INTEG (Design IE[Component]+Carry IE to DIT[Component]-IE to RW from DIT[Component]-Release IE[Component],0)

Units: Ierrors

The stock of integration errors associated with the stock of Designs in Test (in the queue for bench testing) for a given component.

(169) IE in DR[Component]= INTEG (+Release IE[Component]-IE to DIC from DR[Component]+IE Created in DR[Component]+Carry IE to DR[Component]-IE to RW from DR[Component],0)

Units: Ierrors

The stock of integration errors associated with the stock of Designs Released for a given component.

(170) IE in RW[Component]= INTEG (+IE to RW from DIT[Component]-Carry IE to DIT[Component]-Carry IE to DR[Component]+IE to RW from DR[Component]+Coordinating IE[Component]-Fix IE[Component]-IE to DIC from RW[Component]+Add External IE[Component],0)

Units: Ierrors

The stock of integration errors associated with the stock of Designs in Rework for a given component.

(171) IE to DIC from DR[Component]=VT ie2dic from DR[Component]

Units: Ierrors/Month

The rate at which integration errors are moved to the stock of integration errors in designs in coordination (IE in DIC) from the stock of integration errors in designs released (IE in DR) after vehicle testing.

(172) IE to DIC from RW[Component]=VT ie2dic from RW[Component]

Units: Ierrors/Month

The rate at which integration errors are moved to the stock of integration errors in designs in coordination (IE in DIC) from rework after vehicle testing.

(173) IE to RW from DIT[Component]=BT CError Discovery Rate[Component]*IE Density in DIT[Component]

Units: Ierrors/Month

The rate at which integration errors in designs in testing (IE in DIT) are moved to the stock of integration errors in rework (IE in RW). This occurs when designs with integration errors that also have component errors which discovered in bench testing are moved to rework.

(174) IE to RW from DR[Component]=VT ie2rw from DR[Component]

Units: Ierrors/Month

The rate at which integration errors are moved from designs released to the stock of integration errors in rework (IE in RW) after discovery through vehicle testing. These integration errors are from designs that also have component errors discovered in vehicle testing where the designer chooses not to coordinate the integration error but to send it directly for rework to try to fix the component error only instead.

(175) IErrors from Design[a]=New Design Rate[a]*(1-Design IQ[a])*Fraction of Coupled Designs[a]*Fraction of Designs inProg[b]

IErrors from Design[b]=New Design Rate[b]*(1-Design IQ[b])*Fraction of Coupled Designs[b]*Fraction of Designs inProg[a]

Units: Ierrors/Month

The rate at which integration errors are generated through new design.

(176) IErrors from Redesign[a]=Redesign Rate[a]*(1-Design IQ[a])*(1-IE Density in RW[a])*(1-eRW Density in RW[a])*Fraction of Coupled Designs[a]*Fraction of Designs inProg[b]*(1-Overall IE Density)

IErrors from Redesign[b]=Redesign Rate[b]*(1-Design IQ[b])*(1-IE Density in RW[b])*(1-eRW Density in RW[b])*Fraction of Coupled Designs[b]*Fraction of Designs inProg[a]*(1-Overall IE Density)

Units: Ierrors/Month

The rate at which integration errors are generated through redesign.

(177) Indicated cIE[a]=zidz(External RW[b],External RW Probability)

Indicated cIE[b]=zidz(External RW[a],External RW Probability)

Units: Ierrors

The indicated number of designs with coordinated integration errors that a component should have based on the number of designs with external rework that its coupled component has awaiting rework.

(178) Indicated eRW[a]=Coordinated IE in RW[b]*External RW Probability

Indicated eRW[b]=Coordinated IE in RW[a]*External RW Probability

Units: Ierrors

The indicated number of designs with external rework that a component should have based on the number of designs with coordinated integration errors that its coupled component has awaiting rework.

(179) Initial Designs[Component]=1000

Units: designs

Inital set of paper designs required of the project.

(180) kCE Density in RW[Component]=zidz(Known CE in RW[Component],"Designs in Rework (RW)"[Component])

Units: fraction

The density of known component errors in the stock of designs in rework.

(181) Known CE in RW[Component]= INTEG (Add kCE[Component]-Remove kCE[Component]-Redesigning kCE[Component],0)

Units: errors

The stock of known (identified) component errors in designs in rework.

(182) Min Coordination Time=0.25

Units: Month

The minimum time for designers to coordinate how an integration error should be fixed.

(183) Min Design Time[Component]=0.25

Units: Month

The minimum time required to complete a design - set in the simulation to one week.

(184) Min Redesign Time=0.25

Units: Month

The minimum time required to complete a redesign - set in the simulation to one week.

(185) New Design Rate[Component]=(1-Build Pulse)*min(Design Rate from People[Component],Designs to Be Done[Component]/Min Design Time[Component])

Units: designs/Month

The rate at which new designs are completed. The rate is limited by the availability of designers and/or work.

(186) New Designs not Bench Tested[Component]=New Design Rate[Component]*(1-BT Fraction[Component])

Units: designs/Month

Rate at which new designs are completed and released without being sent for bench testing.

(187) New Designs to DIT[Component]=New Design Rate[Component]*BT Fraction[Component]

Units: designs/Month

Rate at which new designs are completed and sent for bench testing.

(188) Redesign Rate[Component]=Redesign Switch*(1-Build Pulse)*min(Redesign Rate from People[Component], "Designs in Rework (RW)"[Component]/Min Redesign Time)

Units: designs/Month

The rate at which redesigns are completed. The rate is limited by the availability of designers and/or work.

(189) Redesign Switch=1

Units: Dmnl [0,1,1]

Switch used to turn Redesigning on an off.

(190) Redesigning Coord IE[Component]=Redesign Rate[Component]*cIE Density in RW[Component]

Units: Ierrors/Month

The rate at which coordinated integration errors are redesigned and thus removed from the stock of Coordinated IE.

(191) Redesigning eRW[Component]=Redesign Rate[Component]*eRW Density in RW[Component]

Units: designs/Month

The rate at which designs are removed from the stock of External Rework based on redesigning designs that required rework due to an integration error with a coupled component design.

(192) Redesigning kCE[Component]=Redesign Rate[Component]*kCE Density in RW[Component]

Units: errors/Month

The rate at which known component errors are redesigned.

(193) Redesigns not Bench Tested[Component]=Redesign Rate[Component]*(1-BT Fraction[Component])

Units: designs/Month

Rate at which redesigns are completed and released without being sent for bench testing.

(194) Release CE[Component]=Bench Test TPut[Component]*CE Density in DIT[Component]-BT CError Discovery Rate[Component]+Dump DIT[Component]*CE Density in DIT[Component]

Units: errors/Month

The rate at which component errors are released to the stock of component errors in designs released (CE in DR). This occurs as component errors are missed in bench testing or the designs are released without bench testing upon a build event (Dump DIT).

(195) Release IE[Component]=Release Rate[Component]*IE Density in DIT[Component]

Units: Ierrors/Month

The rate at which integration errors are released to the stock of integration errors in designs released (IE in DR). This occurs as designs with integration errors are released after bench testing or the designs are released without bench testing upon a build event (Dump DIT).

(196) Release Rate[Component]=Bench Test TPut[Component]-BT CError Discovery Rate[Component]+Dump DIT[Component]

Units: designs/Month

The rate at which designs are released after bench testing. This rate represents those designs that are bench tested and no errors are discovered.

(197) Remove kCE[Component]=VT ce2dic from RW[Component]*Fraction of CE in RW Known[Component]

Units: errors/Month

The rate at which known component errors are removed from the stock of Known CE in RW based on vehicle testing identifying designs in rework that require coordination.

(198) RW to DIC[Component]=VT rw2dic[Component]

Units: designs/Month

The rate at which designs in the rework stock are sent for coordination upon discovery of an integration error in vehicle testing.

(199) RW to DIT[Component]=Redesign Rate[Component]*BT Fraction[Component]

Units: designs/Month

Rate at which redesigns are completed and sent for bench testing.

(200) Unfixed cIE[Component]= INTEG (Add UcIE[Component]-Dump UcIE[Component], 0)

Units: Ierrors

The stock of coordinated integration errors that have been redesigned but still have an integration error.

****

External RW

****

(201) Ext RW from DIT[Component]=External RW from Coordination[Component]*"% IEc in DIT"[Component]

Units: designs/Month

The rate at which designs awaiting bench testing are sent directly to rework based on the generation of external rework when coordinating an integration error associated with a design in a coupled component.

(202) Ext RW from DR[Component]=External RW from Coordination[Component]*"% IEc in DR"[Component]

Units: designs/Month

The rate at which designs released are sent directly to rework based on the generation of external rework when coordinating an integration error associated with a design in a coupled component.

(203) Ext RW switch=1

Units: Dmnl [0,1,1]

Binary switch used to turn external rework on (1) and off (0).

(204) External RW from Coordination[a]=Coordinating Designs[b]*External RW Probability*Ext RW switch

External RW from Coordination[b]=Coordinating Designs[a]*External RW Probability*Ext RW switch

Units: designs/Month

An auxiliary variable representing the rate at which rework is generated for a given component when it coordinates an integration error associated with a design in a coupled component.

(205) External RW Probability=0.5

Units: fraction

The probability that both components (A and B) will need to rework a design corresponding to a given integration error. This probability assumes that one component will always be required to rework a design.

****

Subscript Variables

****

(206) Build: Proto, Proto2, Proto3, Proto4, DIB, DIB2, FPE, FPE2

(207) Component: a,b

****

IE Accounting

****

(208) "% IEc in DIT"[Component]=zidz(IEclean DIT[Component],IEClean Designs InProg[Component])

Units: fraction

The percentage of designs in progress for a given component (A or B) that do not have an integration error that are in the stock of designs in testing.

(209) "% IEc in DR"[Component]=zidz(IEclean DR[Component],IEClean Designs InProg[Component])

Units: fraction

The percentage of designs in progress for a given component (A or B) that do not have an integration error that are in the stock of designs released.

(210) "% IEo in DR"[Component]=zidz(IEonly Errors in DR[Component],Total IEo in VT[Component])

Units: fraction

The percent of designs for a given component (for vehicle testing purposes) that have a integration error only that reside in the stock of Designs Released.

(211) "% IEo in RW"[Component]= zidz(IEonly Errors in RW[Component],Total IEo in VT[Component])

Units: fraction

The percent of designs for a given component (for vehicle testing purposes) that have a integration error only that reside in the stock of Designs in Rework.

(212) Component IE Density[Component]=zidz(IE in Designs[Component],Designs inProg[Component])

Units: fraction

The density of integration errors in the designs in progress for a given component.

(213) Cumulative IE[Component]=Cumulative IE Generated[Component]+Cumulative IE Carried[Component]+Cumulative IE Transferred[Component]

Units: Ierrors

The cumulative number of integration errors that are generated and/or carried over throughout the NPD project for a given component.

(214) Cumulative IE Carried[Component]= INTEG (IE Carried[Component],0)

Units: Ierrors

The cumulative number of integration errors that are carried over throughout the NPD project for a given component.

(215) Cumulative IE Generated[Component]= INTEG (IE Generated[Component],0)

Units: Ierrors

The cumulative number of new integration errors that are generated throughout the NPD project for a given component.

(216) Cumulative IE Transferred[Component]= INTEG (IE Transferred[Component],0)

Units: Ierrors

The cumulative number of integration errors that are transferred to a given component because it did not complete its required rework to fix an integration error when its coupled component design did.

(217) Fixed BT Fraction Switch=0

Units: Dmnl [0,1,1]

A switch that when turned on fixes the fraction of designs that are bench tested (BT Fraction) at the intended fraction essentially turning off the effect of redesign time available on the BT Fraction.

(218) Fraction of Coupled Designs[Component]=0.9

Units: fraction [0,1,0.05]

The fraction of designs for a given component that have a corresponding design in the other component with which it is coupled and therefore must be integrated.

(219) IE Carried[Component]=Carry IE to DIT[Component]+Carry IE to DR[Component]

Units: Ierrors/Month

The rate at which integration errors are carried over either by not fixing a known integration error or redesigning a design with an unknown or uncoordinated integration error.

(220) IE Density in DIC[Component]=zidz(IE in DIC[Component],"Designs in Coordination (DIC)"[Component])

Units: fraction

The density of integration errors in the stock of Designs in Coordination for a given component.

(221) IE Density in DIT[Component]=zidz(IE in DIT[Component],"Designs in Test (DIT)"[Component])

Units: fraction

The density of integration errors in the stock of Designs in Test (the queue for bench testing) for a given component.

(222) IE Density in DR[Component]=zidz(IE in DR[Component],"Designs Released (DR)"[Component])

Units: fraction

The density of integration errors in the stock of Designs Released for a given component.

(223) IE Density in RW[Component]=zidz(IE in RW[Component],"Designs in Rework (RW)"[Component])

Units: fraction

The density of integration errors in the stock of Designs in Rework for a given component..

(224) IE Generated[Component]=IErrors from Design[Component]+IErrors from Redesign[Component]

Units: Ierrors/Month

The rate at which integration errors are generated either through new design or redesign for a given component.

(225) IE in Designs[Component]=IE in DIT[Component]+IE in DR[Component]+IE in RW[Component]+IE in DIC[Component]

Units: Ierrors

An auxiliary variable representing the total number of integration errors in designs in progress (i.e. in testing, in rework, in coordination, and released) for a given component.

(226) IE Transferred[Component]=Add External IE[Component]

Units: Ierrors/Month

The rate at which integration errors are transferred to a given component because it did not complete its required rework to fix an integration error when its coupled component design did.

(227) IEClean Designs InProg[Component]=IEclean DIT[Component]+IEclean DR[Component]+IEclean RW[Component]

Units: Ierrors

The total number of designs in progress for a given component that do not have an integration error. By definition, designs in coordination have an integration error.

(228) IEclean DIT[Component]="Designs in Test (DIT)"[Component]*(1-IE Density in DIT[Component])

Units: designs

The number of designs in the stock of Designs in Test for a given component that do not have an integration error.

(229) IEclean DR[Component]= "Designs Released (DR)"[Component]*(1-IE Density in DR[Component])

Units: designs

The number of designs in the stock of Designs Released for a given component that do not have an integration error.

(230) IEclean RW[Component]="Designs in Rework (RW)"[Component]*(1-IE Density in RW[Component])

Units: designs

The number of designs in the stock of Designs in Rework for a given component that do not have an integration error.

(231) IEonly Errors in DIC[Component]="Designs in Coordination (DIC)"[Component]*IE Density in DIC[Component]-"Designs in Coordination (DIC)"[Component]*IE Density in DIC[Component]*CE Density in DIC[Component]

Units: designs

The number of designs in the stock of Designs in Coordination that have a integration error ONLY for a given component (A or B).

(232) IEonly Errors in DR[Component]="Designs Released (DR)"[Component]*IE Density in DR[Component]-"Designs Released (DR)"[Component]*IE Density in DR[Component] *CE Density in DR[Component]

Units: Ierrors

The number of designs in the stock of Designs Released that have a integration error ONLY for a given component (A or B).

(233) IEonly Errors in RW[Component]="Designs in Rework (RW)"[Component]*IE Density in RW[Component]-"Designs in Rework (RW)"[Component]*IE Density in RW[Component]*CE Density in RW[Component]

Units: designs

The number of designs in the stock of Designs in Rework that have a integration error ONLY for a given component (A or B).

(234) Max IErrors=VMIN(Max IErrors in Component[Component!])

Units: designs

The maximum number of integration errors possible across components based on the number of designs in progress and the fraction of coupled designs for each.

(235) Max IErrors in Component[Component]= Designs inProg[Component]*Fraction of Coupled Designs[Component]

Units: designs

The maximum number of integration errors possible for a given component given the number of designs in progress and the fraction of designs that are coupled with designs from another component.

(236) Overall IE Density=zidz(SumTotal IE in Designs,Max IErrors)

Units: fraction

Ratio of the total number of integration errors across components to the maximum number of integration errors possible across components.

(237) SumTotal Cumulative IE=SUM(Cumulative IE[Component!])

Units: Ierrors

The cumulative number of integration errors that are generated and/or carried over throughout the NPD project across all components.

(238) SumTotal Cumulative IE Generated=SUM(Cumulative IE Generated[Component!])

Units: Ierrors

The sum total of all Integration Errors generated during the NPD project.

(239) SumTotal IE in Designs=SUM(IE in Designs[Component!])

Units: Ierrors

An auxiliary variable representing the total number of integration errors in designs in progress (i.e. in testing, in rework, in coordination, and released) across all components.

(240) Total IEo in VT[Component]= IEonly Errors in DR[Component]+IEonly Errors in RW[Component]+IEonly Errors in DIC[Component]

Units: Ierrors

The total number of designs in rework, in coordination and released that have a integration error ONLY for a given component (A or B). Note: This total does not include IEonly in DIT because the total is used for vehicle testing accounting purposes and errors discovered through VT in parts that have been redesigned are ignored. Since all designs in test are released at the time of a build, any errors in DIT are not relevant to the current vehicle testing.

****

Labor

****

(241) Add Designers[Component]= max(0,Designer Gap[Component]/AVG Time to Add Designers)

Units: designers/Month

The rate at which new designers added to the project team.

(242) AVG Coordination Productivity[Component]=Coordination Productivity*Experienced Fraction[Component]+Coordination Productivity*New Designer Productivity Fraction*(1-Experienced Fraction[Component])

Units: designs/(Month*designer)

The average Coordination Productivity given the experience of the designers.

(243) AVG Design Productivity[Component]=Normal Design Productivity[Component]* Experienced Fraction[Component]+Normal Design Productivity[Component]*New Designer Productivity Fraction*(1-Experienced Fraction[Component])

Units: designs/(Month*designer)

The average design productivity given the experience level of the designers.

(244) AVG Redesign Productivity[Component]=Normal Redesign Productivity[Component]* Experienced Fraction[Component]+Normal Redesign Productivity[Component]*New Designer Productivity Fraction*(1-Experienced Fraction[Component])

Units: designs/(Month*designer)

The average Redesign Productivity given the experience level of the designers.

(245) AVG Time to Add Designers=0.5

Units: Month

The average time it takes to add a new designer to the project team.

(246) AVG Time to Mature=2

Units: Month [0,6,0.25]

The average time it takes a new designer to become experienced on the project team.

(247) AVG Time to Release Designers=0.25

Units: Month

The average time it takes to release a designer from the project team. Includes time to transfer responsibility to another team member.

(248) Change TGT Designers[Component]=(Desired Designers[Component]-TGT Designers[Component])/Time to Change TGT Designers

Units: designers/Month

Rate of change in the management target number of designers.

(249) Designer Gap[Component]=TGT Designers[Component]-Total Designers[Component]

Units: designers

The gap (+ or -) between the target number of designers and the actual number of designers assigned.

(250) Desired Designers[Component]=(max(Init Designers, Desired Designers for Design[Component]+Desired Designers for RW[Component]+Desired Designers for DIC[Component]))*(1-Fixed Labor Switch)+Init Designers*Fixed Labor Switch

Units: designers

Total number of desired designers to complete designs to be done, designs in rework and designs in coordination prior to launch.

(251) Desired Designers for Design[Component]=(Designs to Be Done[Component]/Time until Next Deadline)/AVG Design Productivity[Component]

Units: designers

Desired designers needed to complete design work prior to launch.

(252) Desired Designers for DIC[Component]=("Designs in Coordination (DIC)"[Component]/Time until Next Deadline)/AVG Coordination Productivity[Component]

Units: designers

Desired designers needed to complete coordination prior to launch.

(253) Desired Designers for RW[Component]=(("Designs in Rework (RW)"[Component]+"Designs in Coordination (DIC)"[Component])/Time until Next Deadline)/AVG Redesign Productivity[Component]

Units: designers

Desired designers needed to complete redesign work prior to launch.

(254) Experienced Designers[Component]= INTEG (Mature Designers[Component]-Release Experienced Designers[Component],0)

Units: designers

The number of experienced designers on the project team.

(255) Experienced Fraction[Component]= zidz(Experienced Designers[Component],Total Designers[Component])*(1-Fixed Base Productivity Switch)+Fixed Base Productivity Switch

Units: fraction

The fraction of designers assigned to the project team that are experienced.

(256) Fix Deadline to Launch Switch=1

Units: Dmnl

Switch that when turned on fixes the Time until Next Deadline as the Time until Launch.

(257) Fixed Base Productivity Switch=0

Units: Dmnl

Switch that when turned on fixes the productivity of designers at their base rates.

(258) Fixed Labor Switch=0

Units: Dmnl

Switch that when turned on fixes the number of designers at their initial level.

(259) Init Designers=10

Units: designers

The initial number of designers assigned to the project team for each component. Assumed to be the same for each component.

(260) Mature Designers[Component]=New Designers[Component]/AVG Time to Mature

Units: designers/Month

The rate at which new designers mature and become experienced designers.

(261) New Designer Productivity Fraction=0.5

Units: fraction

The fraction of the normal productivity rate at which a new designer works.

(262) New Designers[Component]= INTEG (Add Designers[Component]-Mature Designers[Component]-Release New Designers[Component], Init Designers)

Units: designers

The number of new designers currently assigned to the project team.

(263) Release Experienced Designers[Component]=-min(0,Designer Gap[Component]* Experienced Fraction[Component]/AVG Time to Release Designers)

Units: designers/Month

The rate at which experienced designers are released from the project team when there are too many assigned.

(264) Release New Designers[Component]=-min(0,Designer Gap[Component]*(1-Experienced Fraction[Component])/AVG Time to Release Designers)

Units: designers/Month

The rate at which new designers are released from the project team when there are too many assigned.

(265) TGT Designers[Component]= INTEG (Change TGT Designers[Component],Init Designers)

Units: designers

The target number of designers management recognizes are needed by the team (by component).

(266) Time to Change TGT Designers=2

Units: Month

Time it takes for management to perceive a change in the desired number of designers and agree to add or subtract resources.

(267) Time until Launch=IF THEN ELSE(Time<Launch Date, max(TIME STEP,Launch Date-Time), max(TIME STEP,FINAL TIME-Time))

Units: Month

The time remaining until the launch date.

(268) Time until Next Deadline=IF THEN ELSE(Time<Launch Date, min(Time till NextBuild,Time until Launch), Time until Launch)*(1-Fix Deadline to Launch Switch)+Time until Launch*Fix Deadline to Launch Switch

Units: Month

Time until the next project deadline (either a build or launch).

(269) Total Designers[Component]=New Designers[Component]+Experienced Designers[Component]

Units: designers

The total number of designers assigned to the project team.

****

MOEs

****

(270) Add Labor[Component]= Total Designers[Component]*TIME STEP/TIME STEP

Units: designers*Month/Month

The flow of labor (in designer man-months) added to the stock of Total Labor per month.

(271) Clean Design Churn[Component]=zidz(Cumulative Designs[Component],Clean Designs[Component])

Units: Dmnl

A measure of effectiveness that represents a ratio of the number of cumulative designs drawn per "clean" design for a given component. A perfect design process (one that does not introduce any errors) would have a value of 1.

(272) Discovering Errors[Component]=SUM(netFail VT[Component,Build!])

Units: errors/Month

BT CError Discovery Rate[Component]+SUM(cFail VT[Component,Build!])+SUM(iFail VT[Component,Build!])

(273) Errors Discovered[Component]= INTEG (Discovering Errors[Component],0)

Units: errors

(274) Fraction of Component Designers Idle[Component]=zidz(Idle Designers[Component],Total Designers[Component])

Units: fraction

The fraction of designers assigned to a given component who are idle at any given point in time.

(275) Fraction of Total Designers Idle=zidz(SumTotal Idle Designers,SumTotal Designers)

Units: fraction

The fraction of all designers (from both component A and B) who are idle at any given point in time.

(276) Idle Designers[Component]=max(0,Total Designers[Component]-Designers Working on Design[Component]-Designers Working on Redesign[Component]-Designers Working on Coordination[Component])

Units: designers

The number of designers from a given component (A or B) who are not allocated for design work, coordination, redesign work or working ahead at any given point in time.

(277) MOE1[Component]=(Clean Designs[Component]/Initial Designs[Component])* zidz(1,Clean Design Churn[Component])

Units: Dmnl

A measure of effectiveness of each component which reflects the fraction of initial designs that have been designed and are error free ("clean") as well as the cumulative number of designs required to complete that many error free designs.

(278) Overall Clean Design Churn= zidz(SumTotal Cumulative Designs,SumTotal Clean Designs)

Units: Dmnl

A measure of how efficient the overall design process is across components that reflects how many cumulative designs were required to attain the present number of error free ("clean") designs.

(279) Overall MOE1=(SumTotal Clean Designs/SUM(Initial Designs[Component!]))* zidz(1,Overall Clean Design Churn)

Units: Dmnl

A measure of effectiveness which reflects the fraction of initial designs that have been designed and are error free ("clean") as well as the cumulative number of designs required to complete that many error free designs.

(280) "Phantom Work %"=zidz(SumTotal Phantom Work,SumTotal Cumulative Designs)

Units: fraction

(281) SumTotal Designers=SUM(Total Designers[Component!])

Units: designers

(282) SumTotal Errors=SumTotal CE in Designs+SumTotal IE in Designs

Units: errors

(283) SumTotal Errors Discovered=SUM(Errors Discovered[Component!])

Units: errors

(284) SumTotal Errors Generated= SumTotal Cumulative CE+SumTotal Cumulative IE Generated

Units: errors

The sum total of all errors generated.

(285) SumTotal Idle Designers=SUM(Idle Designers[Component!])

Units: designers

The number of all designers (from both components A and B) who are idle at any given point in time.

(286) SumTotal Labor=SUM(Total Labor[Component!])

Units: designers*Month

The sum total of labor in designer man-months assigned to both component design teams for the duration of the simulation.

(287) Total Labor[Component]= INTEG (Add Labor[Component],0)

Units: designers*Month

The total labor in designer man-months assigned to a component design team for the duration of the simulation.

****

Phantom Work

****

(288) Add Phantom IE[Component]=Add External IE[Component]+Dump Coord IE[Component]

Units: Ierrors/Month

The rate at which integration errors are added to the stock of Phantom IE for a given component.

(289) Add Phantom RW[a]=RW Dumped cIE[a]+RW External IE[a]+Add Phantom IE[b]*External RW Probability

Add Phantom RW[b]= RW Dumped cIE[b]+RW External IE[b]+Add Phantom IE[a]*External RW Probability

Units: Ierrors/Month

The rate at which phantom rework is added to the stock of Phantom RW. It represents the rate at which phantom rework is done by a given component through reworking designs with integration errors that have been "dumped", redesigning external rework when it is too late or redesigning an external integration error (External RW) for the second time.

(290) Cumulative Phantom Work[Component]=Phantom IE[Component]+Phantom RW[Component]

Units: Ierrors

An auxiliary variable reflecting the cumulative amount of phantom work (both Phantom RW and Phantom IE) for a given component at any point in time.

(291) Disc Dumped cIE[Component]=zidz(Dumped cIE[Component],IE in RW[Component]*(1-cIE Density in RW[Component]))*IE to DIC from RW[Component]

Units: Ierrors/Month

This variable reflects the rate at which coordinated integrations errors which have been dumped are discovered in subsequent builds through vehicle testing.

(292) Disc External IE[Component]=zidz(External IE[Component],IE in RW[Component]*(1-cIE Density in RW[Component]))*IE to DIC from RW[Component]

Units: Ierrors/Month

The rate at which external coordinated integration errors for a given component are discovered in subsequent builds through vehicle testing.

(293) Dumped cIE[Component]= INTEG (Dump Coord IE[Component]-Disc Dumped cIE[Component]-RW Dumped cIE[Component,0)

Units: Ierrors

The stock of previously coordinated integration errors in rework that were dumped at the time of a build because the given component did not complete its required rework while its coupled design counterpart did.

(294) Dumped cIE Density[Component]=zidz(Dumped cIE[Component],"Designs in Rework (RW)"[Component])

Units: fraction

The density (or fraction) of integration errors in rework that have integration errors that have been dumped.

(295) External IE[Component]= INTEG (Add External IE[Component]-Disc External IE[Component]-RW External IE[Component],0)

Units: Ierrors

The stock of previously coordinated integration errors in rework that came from the other component because the given component did not complete its required external rework while its coupled design counterpart did.

(296) External IE Density[Component]=zidz(External IE[Component],"Designs in Rework (RW)"[Component])

Units: fraction

The density (or fraction) of integration errors in rework that are External cIE.

(297) Phantom IE[Component]= INTEG (Add Phantom IE[Component],0)

Units: Ierrors

The cumulative number of integration errors that are created for a given component (A or B) during the design process because only one of the coupled designs requiring rework due to an integration error was completed prior to a given build event.

(298) Phantom IE ManMonths[Component]=(Phantom IE[Component]/Coordination Productivity)*2

Units: Month*designer

The number of months a designer must work coordinating phantom integration errors. The quotient is multiplied by two in the equation because coordination requires a designer from both of the coupled components.

(299) Phantom RW[Component]= INTEG (Add Phantom RW[Component],0)

Units: Ierrors

The cumulative number of designs for a given component that are redesigned after a build event has occurred where there exists an integration error and only one of the coupled designs requiring rework was completed prior to the build event.

(300) Phantom RW ManMonths[Component]=Phantom RW[Component]/Normal Redesign Productivity[Component]

Units: Month*designer

The number of months of designer work that are spent on phantom rework assuming the base (or normal) redesign productivity.

(301) RW Dumped cIE[Component]=Redesign Rate[Component]*Dumped cIE Density[Component]

Units: Ierrors/Month

This variable reflects the rate at which coordinated integrations errors which have been dumped are redesigned.

(302) RW External IE[Component]=Redesign Rate[Component]*External IE Density[Component]

Units: Ierrors/Month

The rate at which external coordinated integration errors for a given component are redesigned.

(303) SumTotal Phantom IE=SUM(Phantom IE[Component!])

Units: Ierrors

The total amount of accumulated Phantom IE across all components.

(304) SumTotal Phantom ManMonths=SUM(Phantom RW ManMonths[Component!])+ SUM(Phantom IE ManMonths[Component!])

Units: Month*designer

The total number of months of designer work that are spent on phantom rework and/or coordinating phantom integration errors.

(305) SumTotal Phantom RW=SUM(Phantom RW[Component!])

Units: Ierrors

The total amount of accumulated Phantom Rework (designs reworked unnecessarily) across all components.

(306) SumTotal Phantom Work=SUM(Cumulative Phantom Work[Component!])

Units: Ierrors

The total amount of accumulated Phantom Work across components (A and B).

****

Productivity

****

(307) Capable Coordination Rate[Component]=Designers Available for Coordination*Relative Desired Coordination Rate[Component]*AVG Coordination Productivity[Component]

Units: designs/Month

The rate at which designs can be coordinated based on the number of designers available for coordination and the normal coordination productivity. The rate is affected by the relative desired coordination rate which allocates the coordination work completed between the coupled componenets (A and B).

(308) Capable Design Rate[Component]=Total Designers[Component]*Design Productivity[Component]

Units: designs/Month

The rate at which new designs can be completed based on the number of designers available and their design productivity.

(309) Capable Redesign Rate[Component]=Designers Available for Redesign[Component]*Redesign Productivity[Component]

Units: designs/Month

(310) Coordination Pressure[Component]=IF THEN ELSE(Desired Coordination Rate[Component]=0, 0, xidz(Desired Coordination Rate[Component], Capable Coordination Rate[Component], 10))

Units: Dmnl

(311) Coordination Productivity=20

Units: designs/(Month*designer) [0,40,0.5]

The normal rate at which designers are able to coordinate integration errors in designs. This coordination rate requires a designer from both components (A and B) and is assumed to be constant and the same for both components.

(312) Coordination Rate from People[Component]=min(Desired Coordination Rate[Component],Capable Coordination Rate[Component])

Units: designs/Month

The rate at which designs with integration errors can be coordinated based on the amount of designers available and their productivity (normal coordination rate) and/ or the amount of coordination to be done and the time remaining until the next build (desired coordination rate).

(313) Design Pressure[Component]=IF THEN ELSE(Desired Design Rate[Component]=0, 0, xidz(Desired Design Rate[Component], Normal Design Rate[Component], 10))

Units: Dmnl

A dimensionless variable used to represent the pressure felt by designers to complete design work as the desired design rate increases relative to the normal design rate.

(314) Design Productivity[Component]=AVG Design Productivity[Component]*Effect of Pressure on DesProd[Component]

Units: designs/(Month*designer)

The rate at which an individual designer is able to complete new designs..

(315) Design Rate from People[Component]=min(Desired Design Rate[Component],Capable Design Rate[Component])

Units: designs/Month

The rate at which new designs can be completed based on the amount of designers available and their productivity (normal design rate) and/ or the amount of designs to be done and the time remaining until the next build (desired design rate). This design rate assumes that if the desired design rate is less than the normal design rate that the designers will complete the designs at the slower pace using their extra time for other activities.

(316) Desired Coordination Rate[Component]=zidz("Designs in Coordination (DIC)"[Component],max(TIME STEP, Time till NextBuild-4*Min Redesign Time))

Units: designs/Month

The coordination rate desired based on the number of designs to be coordinated (i.e. those with an integration error identified through vehicle testing) and the time remaining until the next build event adjusted for the minimum time it takes to redesign.

(317) Desired Design Rate[Component]= zidz(Designs to Be Done[Component],Time till NextBuild)

Units: designs/Month

The design rate desired based on the number of designs to be completed and the time remaining until the next build event.

(318) Desired Redesign Rate[Component]=zidz("Designs in Rework (RW)"[Component],Time till NextBuild)

Units: designs/Month

The redesign rate desired based on the number of designs to be reworked and the time remaining until the next build event.

(319) Effect of Pressure on DesProd[Component]= Fn for Effect of Press on Productivity(Design Pressure[Component])*(1-Fixed Productivity Switch)+Fixed Productivity Switch

Units: Dmnl

The effect that design pressure has on the rate at which individual designers are able to complete new designs.

(320) Effect of Pressure on RedProd[Component]=Fn for Effect of Press on Productivity(Redesign Pressure[Component])*(1-Fixed Productivity Switch)+Fixed Productivity Switch

Units: Dmnl

The effect that redesign pressure has on the rate at which individual designers are able to complete redesigns.

(321) Fixed Productivity Switch=0

Units: Dmnl [0,1,1]

A switch that when turned on fixes the design and redesign productivity at their base (normal) level by turning off the effect of pressure on productivity.

(322) Fn for Effect of Press on Productivity([(0,0)-(5,4)], (0,1),(0.5,1),(1,1),(1.5,1.124),(2,1.28),(2.5,1.429),(3,1.539),(3.5,1.613),(4,1.65),(4.5,1.667), (5,1.667))

Units: Dmnl

A table function which captures the effect that design pressure has on the rate at which individual designers are able to complete new designs.

(323) Normal Design Rate[Component]=Total Designers[Component]*AVG Design Productivity[Component]

Units: designs/Month

The rate at which designs can be completed based on the number of designers available for design work and the normal design productivity.

(324) Normal Redesign Rate[Component]=Designers Available for Redesign[Component]*AVG Redesign Productivity[Component]

Units: designs/Month

The rate at which redesigns can be completed based on the number of designers available for redesign work and the normal redesign productivity.

(325) Redesign Pressure[Component]=IF THEN ELSE(Desired Redesign Rate[Component]=0, 0, xidz(Desired Redesign Rate[Component], Normal Redesign Rate[Component], 10))

Units: Dmnl

A dimensionless variable used to represent the pressure felt by designers to complete redesign work as the desired redesign rate increases relative to the normal redesign rate.

(326) Redesign Productivity[Component]=AVG Redesign Productivity[Component]*Effect of Pressure on RedProd[Component]

Units: designs/(Month*designer)

The rate at which an individual designer is able to complete redesigns.

(327) Redesign Rate from People[Component]=min(Desired Redesign Rate[Component],Capable Redesign Rate[Component])

Units: designs/Month

The rate at which designs in rework can be completed based on the amount of designers available and their productivity (normal redesign rate) and/ or the amount of rework to be done and the time remaining until the next build (desired redesign rate). This redesign rate assumes that if the desired redesign rate is less than the normal redesign rate that the

designers will complete the designs in rework at the slower pace using their extra time for other activities.

(328) Relative Desired Coordination Rate[Component]=zidz(Desired Coordination Rate[Component],SUM(Desired Coordination Rate[Component!]))

Units: fraction

The desired coordination rate for a given component relative to the desired coordination rate of its coupled component.

****

Quality

****

(329) Base Design Component Quality=0.85

Units: fraction

A fraction between 0 and 1 representing the maximum designcomponent quality achievable given no resource or time constraints.

(330) Base Design Integration Quality=0.5

Units: fraction

A fraction between 0 and 1 representing the maximum design integration quality achievable given no resource or time constraints.

(331) Base Redesign Component Quality=1-(1-Base Design Component Quality)/2

Units: fraction

A fraction between 0 and 1 representing the maximum redesign component quality achievable given no resource or time constraints. The max value is set to be halfway between the Max Design Component Quality and 1.

(332) Base Redesign Integration Quality=1-(1-Base Design Integration Quality)/2

Units: fraction

A fraction between 0 and 1 representing the maximum redesign integration quality achievable given no resource or time constraints. The max value is set to be halfway between the Max Design Integration Quality and 1.

(333) Design CQ[Component]=Base Design Component Quality*Effect of Rel Design Productivity on Design CQ[Component]*Effect of Experience on Quality[Component]*(1-Fixed Quality Switch)+Base Design Component Quality*Fixed Quality Switch

Units: fraction

The component quality of design, valued between 0 and 1, representing the percent of designs completed without introducing a component error.

(334) Design IQ[Component]=Base Design Integration Quality*Effect of Rel Design Productivity on Design IQ[Component]*Effect of Experience on Quality[Component]*(1-Fixed Quality Switch)+Base Design Integration Quality*Fixed Quality Switch

Units: fraction

The integration quality of design, valued between 0 and 1, representing the percent of designs completed without introducing a integration error.

(335) Effect of Experience on Quality[Component]=Fn for Effect of Experience on Quality(Experienced Fraction[Component])*Exp on Quality Switch+(1-Exp on Quality Switch)

Units: fraction

The effect that experience has on the quality of design and redesign work - namely that inexperienced workers will have lower quality in their work resulting in more errors.

(336) Effect of Rel Design Productivity on Design CQ[Component]=Fn for Effect of Rel Prod on CQ(Relative Design Productivity[Component])

Units: Dmnl

The effect of increased productivity on the component quality of design work which assumes that increased design productivity results in a decrease in the component quality of design as designers cut corners and work longer hours in order to increase productivity.

(337) Effect of Rel Design Productivity on Design IQ[Component]=Fn for Effect of Rel Prod on IQ(Relative Design Productivity[Component])

Units: Dmnl

The effect of productivity on the integration quality of design work which assumes that increased design productivity results in a decrease in the integration quality of design as designers cut corners and work longer hours to increase productivity.

(338) Effect of Rel Redesign Productivity on Redesign CQ[Component]=Fn for Effect of Rel Prod on CQ(Relative Redesign Productivity[Component])

Units: Dmnl

The effect of increased productivity on the component quality of redesign work which assumes that increased redesign productivity results in a decrease in the component quality of redesign as designers cut corners and work longer hours in order to increase productivity.

(339) Effect of Rel Redesign Productivity on Redesign IQ[Component]=Fn for Effect of Rel Prod on IQ(Relative Redesign Productivity[Component])

Units: Dmnl

The effect of productivity on the integration quality of redesign work which assumes that increased redesign productivity results in a decrease in the integration quality of redesign as designers cut corners and work longer hours to increase productivity.

(340) Exp on Quality Switch=1

Units: Dmnl

Switch that when turned off removes the effect of experience on the quality (both component and integration) of designs and redesigns.

(341) Fixed Quality Switch=0

Units: Dmnl [0,1,1]

A switch that when turned on will fix the quality (component and integration) of design and redesign at their base quality levels.

(342) Fn for Effect of Experience on Quality([(0,0)-(1,1)], (0,0.67),(0.0948012,0.675439),(0.2,0.7),(0.7,0.95),(0.85,0.975),(1,1))

Units: Dmnl

Table function describing the effect that various levels of experience (the fraction of the team members that are experienced) has on quality.

(343) Fn for Effect of Rel Prod on CQ([(0.5,0)-(5,1.2)], (0,1),(0.25,1),(0.5,1),(0.75,1),(1,1),(1.07,0.976),(1.16,0.928),(1.28,0.867),(1.44,0.778),(1.66,0.611),(1.99,0.444),(2.54,0.278),(3.63,0.111),(6.92,0.033),(13.5,0))

Units: Dmnl

A table function used to capture the effect of relative design productivity (the relative rate at which engineers complete designs) on the component quality of design work which assumes that an increase in the relative design productivity results in a decrease in the component quality of design.

(344) Fn for Effect of Rel Prod on IQ([(0.5,0)-(5,1.2)], (0,1),(0.25,1),(0.5,1),(0.75,1),(1,1),(1.06,0.976),(1.14,0.928),(1.25,0.867),(1.39,0.778),(1.58,0.611),(1.87,0.444),(2.35,0.278),(3.32,0.111),(6.21,0.0333),(12,0))

Units: Dmnl

A table function used to capture the effect of design pressure on the integration quality of design work which assumes that increased design pressure results in a decrease in the integration quality of design.

(345) Perfect CE Fix Switch=0

Units: Dmnl

Dimensionless switch that when turned on is used to set the Redesign CQ to a value of 1 ensuring all known component errors are fixed in redesign and no new ones are introduced.

(346) Perfect IE Fix Switch=0

Units: Dmnl

Dimensionless switch that when turned on is used to set the Redesign IQ to a value of 1 ensuring all known and coordinated integration errors are fixed in redesign and no new ones are introduced.

(347) Redesign CQ[Component]= max(Base Redesign Component Quality*Effect of Rel Redesign Productivity on Redesign CQ[Component]*Effect of Experience on

Quality[Component]*(1-Fixed Quality Switch)+Base Redesign Component Quality*Fixed Quality Switch, Perfect CE Fix Switch)

Units: fraction

The component quality of redesign, valued between 0 and 1, representing the fraction of redesigns with known component errors that are fixed and the fraction of redesigns without a component error that are completed without introducing a component error.

(348) Redesign IQ[Component]=max(Base Redesign Integration Quality*Effect of Rel Redesign Productivity on Redesign IQ[Component]*Effect of Experience on Quality[Component]*(1-Fixed Quality Switch)+Base Redesign Integration Quality*Fixed Quality Switch, Perfect IE Fix Switch)

Units: fraction

The integration quality of redesign, valued between 0 and 1, representing the fraction of redesigns with known and coordinated integration errors that are fixed and the fraction of redesigns without an integration error that are completed without introducing an integration error.

(349) Relative Design Productivity[Component]=Design Productivity[Component]/AVG Design Productivity[Component]

Units: Dmnl

The relative rate at which designers are completing new designs as compared to the average design rate for completing new designs. A relative rate above 1 indicates that designers are working faster than they normally would on average in completing designs.

(350) Relative Redesign Productivity[Component]=Redesign Productivity[Component]/AVG Redesign Productivity[Component]

Units: Dmnl

The relative rate at which designers are completing redesigns as compared to the average design rate for completing redesigns. A relative rate above 1 indicates that designers are working faster than they normally would on average in completing designs.

****

Speed Factor

****

(351) Base Bench Test Time=1

Units: Month

The base time it takes to conduct bench testing.

(352) Base Design Productivity=10

Units: designs/(Month*designer)

The base number of designs completed per month by each designer.

(353) Base Redesign Productivity=5

Units: designs/(Month*designer)

The base number of redesigns completed per month by each designer.

(354) Base Vehicle Test Time=3

Units: Month

The base time it takes to conduct vehicle testing.

(355) Bench Test Time[a]=BTTa

Bench Test Time[b]=BTTb

Units: Month

The average time it takes to bench test a design for a given component.

(356) BTTa= BTTb/Speed Factor*(1-Fix SF Equal Switch)+Fix SF Equal Switch*BTTb

Units: Month

The average time it takes to conduct bench testing on Component A.

(357) BTTb=Base Bench Test Time*2*(Speed Factor/(1+Speed Factor))*(1-Fix SF Equal Switch)+Fix SF Equal Switch*Base Bench Test Time

Units: Month

The average time it takes to conduct bench testing on Component B.

(358) Fix SF Equal Switch=0

Units: Dmnl

Switch that when turned on fixes the speed of the two components design, redesign, bench test and vehicle test equal to each other.

(359) NDPa= NDPb*Speed Factor*(1-Fix SF Equal Switch)+Fix SF Equal Switch*NDPb

Units: designs/(Month*designer)

The normal design productivity (the average number of designs completed per month per designer) for component A.

(360) NDPb=Base Design Productivity*2*(1/(1+Speed Factor))*(1-Fix SF Equal Switch)+Fix SF Equal Switch*Base Design Productivity

Units: designs/(Month*designer)

The normal design productivity (the average number of designs completed per month per designer) for component B.

(361) Normal Design Productivity[a]=NDPa

Normal Design Productivity[b]=NDPb

Units: designs/(Month*designer)

The normal number of designs completed per month per designer. This number can be different based on the component (A or B) being worked on by the designer.

(362) Normal Redesign Productivity[a]=NRDPa

Normal Redesign Productivity[b]=NRDPb

Units: designs/(Month*designer)

The normal number of redesigns completed per month per designer. This number can be different based on the component (A or B) being worked on by the designer.

(363) NRDPa=NRDPb*Speed Factor*(1-Fix SF Equal Switch)+Fix SF Equal Switch*NRDPb

Units: designs/(Month*designer)

The normal redesign productivity (the average number of redesigns completed per month per designer) for component A.

(364) NRDPb=Base Redesign Productivity*2*(1/(1+Speed Factor))*(1-Fix SF Equal Switch)

+Fix SF Equal Switch*Base Redesign Productivity

Units: designs/(Month*designer) [0,40,0.25]

The normal redesign productivity (the average number of redesigns completed per month per designer) for component B.

(365) Speed Factor=2

Units: Dmnl [1,5,0.25]

The degree to which Component A is faster than Component B in terms of designing and redesigning components and conducting vehicle and bench testing. A speed factor of 2 means that Component A tasks are accomplished twice as fast as Component B tasks.

(366) Vehicle Test Time[a]=VTTa

Vehicle Test Time[b]=VTTb

Units: Month

The average time it takes to conduct vehicle testing for a given component.

(367) VTTa= VTTb/Speed Factor*(1-Fix SF Equal Switch)+Fix SF Equal Switch*VTTb

Units: Month

The average time it takes to conduct vehicle testing on Component A.

(368) VTTb=Base Vehicle Test Time*2*(Speed Factor/(1+Speed Factor))*(1-Fix SF Equal Switch)+Fix SF Equal Switch*Base Vehicle Test Time

Units: Month

The average time it takes to conduct vehicle testing on Component B.

****

Vehicle Testing

****

(369) bothFail VT[Component,Build]=VT TPut[Component,Build]*Build CE Density[Component,Build]*Build IE Density[Component,Build]

Units: parts/Month

An auxiliary variable representing the rate at which build parts fail in vehicle testing because a component error and an integration error were discovered.

(370) cFail VT[Component,Build]=VT TPut[Component,Build]*Build CE Density[Component,Build]

Units: parts/Month

An auxiliary variable representing the rate at which build parts fail in vehicle testing because a component error was discovered.

(371) cOnly Fail VT[Component,Build]=cFail VT[Component,Build]-bothFail VT[Component,Build]

Units: parts/Month

An auxiliary variable representing the rate at which build parts fail in vehicle testing because a part with ONLY a component error was discovered.

(372) iFail VT[Component,Build]=VT TPut[Component,Build]*Build IE Density[Component,Build]

Units: parts/Month

An auxiliary variable representing the rate at which build parts fail in vehicle testing because an integration error was discovered.

(373) iOnly Fail VT[Component,Build]=iFail VT[Component,Build]-bothFail VT[Component,Build]

Units: parts/Month

An auxiliary variable representing the rate at which build parts fail in vehicle testing because a part with ONLY an integration error was discovered.

(374) netFail VT[Component,Build]=cFail VT[Component,Build]+iFail VT[Component,Build]-bothFail VT[Component,Build]

Units: parts/Month

An auxiliary variable representing the net rate at which build parts fail in vehicle testing because either a component error, an integration error or both were discovered.

(375) Sum bothFail VT[Component]=SUM(bothFail VT[Component,Build!])

Units: parts/Month

An auxiliary variable representing the sum total of the rates at which build parts fail in vehicle testing across all active builds because a part with both a component error and an integration error was discovered.

(376) Sum cOnly Fail VT[Component]=SUM(cOnly Fail VT[Component,Build!])

Units: parts/Month

An auxiliary variable representing the sum total of the rate at which build parts fail in vehicle testing across all active builds because a part with ONLY a component error was discovered.

(377) Sum iOnly Fail VT[Component]=SUM(iOnly Fail VT[Component,Build!])

Units: parts/Month

An auxiliary variable representing the sum total of the rates at which build parts fail in vehicle testing across all active builds because a part with ONLY an integration error was discovered.

(378) VT ce2dic from DR[Component]= Sum bothFail VT[Component]*"% Both in DR (VT)"[Component]*Coordination Fraction[Component]

Units: designs/Month

The rate at which component errors in Designs Released (CE in DR) are identified through vehicle testing and designated to move with their associated designs to the stock of CE in DIC.

(379) VT ce2dic from RW[Component]= Sum bothFail VT[Component]*"% Both in RW (VT)"[Component]*(1-Fraction of IE in RW Coordinated[Component])*Coordination Fraction[Component]

Units: designs/Month

The rate at which component errors in the Designs in Rework (CE in RW) are identified through vehicle testing and designated to move with their associated designs to the stock of CE in DIC.

(380) VT ce2rw from DR[Component]= Sum cOnly Fail VT[Component]*"% CEo in DR (VT)"[Component]+Sum bothFail VT[Component]*"% Both in DR (VT)"[Component]*(1-Coordination Fraction[Component])

Units: designs/Month

The rate at which component errors in Designs Released (CE inDR) are identified through vehicle testing and designated to move with their associated designs to the stock of CE in RW.

(381) VT ce2rw from RW[Component]=Sum cOnly Fail VT[Component]*"% CEo in RW (VT)"[Component]+Sum bothFail VT[Component]*"% Both in RW (VT)"[Component]-Sum bothFail VT[Component]*"% Both in RW (VT)"[Component]*(1-Fraction of IE in RW Coordinated[Component])*Coordination Fraction[Component]

Units: designs/Month

The rate at which component errors in Designs in Rework (CE in RW) are identified through vehicle testing and designated to remain in the stock of CE in RW.

(382) VT dr2dic[Component]=VT ie2dic from DR[Component]

Units: designs/Month

The rate at which designs released are sent for coordination based on the discovery of integration errors through vehicle testing.

(383) VT dr2rw[Component]=VT ce2rw from DR[Component]

Units: designs/Month

The rate at which designs released are sent directly to rework based on the discovery of errors (component and/or integration errors) through vehicle testing.

(384) VT ie2dic from DR[Component]=Sum iOnly Fail VT[Component]*"% IEo in DR"[Component]+Sum bothFail VT[Component]*"% Both in DR (VT)"[Component]*Coordination Fraction[Component]

Units: designs/Month

The rate at which integration errors in Designs Released (IE in DR) are identified through vehicle testing and designated to move with their associated designs to the stock of IE in DIC.

(385) VT ie2dic from RW[Component]=Sum iOnly Fail VT[Component]*"% IEo in RW"[Component]*(1-Fraction of IE in RW Coordinated[Component])+Sum bothFail VT[Component]*"% Both in RW (VT)"[Component]*(1-Fraction of IE in RW Coordinated[Component])*Coordination Fraction[Component]

Units: designs/Month

The rate at which integration errors in the Designs in Rework (IE in RW) are identified through vehicle testing and designated to move with their associated designs to the stock of IE in DIC.

(386) VT ie2rw from DR[Component]= Sum bothFail VT[Component]*"% Both in DR (VT)"[Component]*(1-Coordination Fraction[Component])

Units: designs/Month

The rate at which integration errors in Designs Released (IE in DR) are identified through vehicle testing and designated to move with their associated designs to the stock of IE in RW.

(387) VT ie2rw from RW[Component]=Sum iOnly Fail VT[Component] * "% IEo in RW"[Component] * Fraction of IE in RW Coordinated[Component] +Sum bothFail VT[Component] * "% Both in RW (VT)"[Component] - Sum bothFail VT[Component] * "% Both in RW (VT)"[Component] * (1-Fraction of IE in RW Coordinated[Component]) * Coordination Fraction[Component]

Units: designs/Month

The rate at which integration errors in Designs in Rework (IE in RW) are identified through vehicle testing and designated to remain in the stock of IE in RW.

(388) VT rw2dic[Component]=VT ie2dic from RW[Component]

Units: designs/Month

The rate at which designs in rework are sent for coordination based on the discovery of integration errors through vehicle testing.

APPENDIX C: COMMAND SCRIPT LISTING

The following command scripts dictate the parameter settings for the Policy Analysis section of Chapter 5. Each set of commands sets the parameters for a different policy. For example, the first set of commands listed below sets the parameters for the base simulation of the Project ATOMac case study. Similarly, command scripts were used to conduct the sensitivity analysis described in Chapter 5. The parameter values used in this sensitivity analysis are included in tabular form in chapter 5 and thus are not repeated here.

Base Case (Project ATOMac)

```
SIMULATE>SETVAL|FINAL TIME=40
SIMULATE>SETVAL|Build Date[Proto]=4
SIMULATE>SETVAL|Build Date[Proto2]=12
SIMULATE>SETVAL|Build Date[Proto3]=17.5
SIMULATE>SETVAL|Build Date[Proto4]=22.5
SIMULATE>SETVAL|Build Date[DIB]=28.75
SIMULATE>SETVAL|Build Date[FPE]=34.5
SIMULATE>SETVAL|Build Date[FPE2]=37.75
SIMULATE>SETVAL|Launch Date=39.25
SIMULATE>RUNNAME|ATOMac Base
MENU>RUN|o
```

Single Prototype Build Policy

```
SIMULATE>SETVAL|FINAL TIME=40
SIMULATE>SETVAL|Build Date[Proto]=1e6
SIMULATE>SETVAL|Build Date[Proto2]=12
SIMULATE>SETVAL|Build Date[Proto3]=1e6
SIMULATE>SETVAL|Build Date[Proto4]=1e6
SIMULATE>SETVAL|Build Date[DIB]=28.75
SIMULATE>SETVAL|Build Date[FPE]=34.5
SIMULATE>SETVAL|Build Date[FPE2]=1e6
SIMULATE>SETVAL|Launch Date=39.25
SIMULATE>RUNNAME|ATOMac Fewer Builds
MENU>RUN|o
```

Aggressive Labor Policy

```
SIMULATE>SETVAL|FINAL TIME=40
SIMULATE>SETVAL|FIX DEADLINE TO LAUNCH SWITCH=0
SIMULATE>SETVAL|Build Date[Proto]=4
SIMULATE>SETVAL|Build Date[Proto2]=12
SIMULATE>SETVAL|Build Date[Proto3]=17.5
SIMULATE>SETVAL|Build Date[Proto4]=22.5
SIMULATE>SETVAL|Build Date[DIB]=28.75
SIMULATE>SETVAL|Build Date[FPE]=34.5
SIMULATE>SETVAL|Build Date[FPE2]=37.75
SIMULATE>SETVAL|Launch Date=39.25
SIMULATE>RUNNAME|ATOMac Labor
MENU>RUN|o
```

Combined Labor and Build Policy

```
SIMULATE>SETVAL|FINAL TIME=40
SIMULATE>SETVAL|FIX DEADLINE TO LAUNCH SWITCH=0
SIMULATE>SETVAL|FIX Real Proto Switch=1
SIMULATE>SETVAL|Build Date[Proto]=1e6
SIMULATE>SETVAL|Build Date[Proto2]=21
SIMULATE>SETVAL|Build Date[Proto3]=1e6
SIMULATE>SETVAL|Build Date[Proto4]=1e6
SIMULATE>SETVAL|Build Date[DIB]=28.75
SIMULATE>SETVAL|Build Date[FPE]=34.5
SIMULATE>SETVAL|Build Date[FPE2]=1e6
SIMULATE>SETVAL|Launch Date=39.25
SIMULATE>RUNNAME|ATOMac Combined Labor and Builds
MENU>RUN|o
```