UNDERSTANDING, MODELING AND IMPROVING THE DEVELOPMENT OF COMPLEX PRODUCTS: METHOD AND STUDY

by

Bradley W. Rogers

B.S. 2003, Mechanical Engineering
Auburn University

Submitted to the Engineering Systems Division and the
MIT Sloan School of Management
in partial fulfillment of the requirements for the degrees of

Masters of Science in Engineering Systems
and
Masters of Business Administration

at the
Massachusetts Institute of Technology
June 2009.

Signature of Author _____
May 8, 2009
Engineering Systems Division &
MIT Sloan School of Management

Certified by _____
Deborah Nightingale, Thesis Supervisor
Professor of the Practice of
Aeronautics & Astronautics and of Engineering Systems Division

Certified by _____
Steven D. Eppinger, Thesis Supervisor
General Motors LFM Professor of Management
MIT Sloan School of Management

Accepted by _____
Nancy Leveson
Professor of Aeronautics & Astronautics and of Engineering Systems Division
Chair, Engineering Systems Division Education Committee

Accepted by _____
Debbie Berechman
Executive Director of the MBA Program
MIT Sloan School of Management

This page has been intentionally left blank.

UNDERSTANDING, MODELING AND IMPROVING THE DEVELOPMENT OF COMPLEX PRODUCTS: METHOD AND STUDY

by

Bradley W. Rogers

Submitted to the Engineering Systems Division and the
MIT Sloan School of Management
on May 8, 2009
in partial fulfillment of the requirements for the degrees of

Masters of Science in Engineering Systems
and
Masters of Business Administration

ABSTRACT

Development of new aerostructure designs frequently occurs through a complex process that is difficult to understand and control. Tight requirements for weight, cost, strength, and aerodynamic behavior create many interdependencies in the product design, which translate through to the design process. An increasing fragmentation of the commercial aerospace industry has also added a dimension of complexity to the process – outsourced component designs are often interdependent with in-house component designs, resulting in frequently changing requirements for supplier components during the design process.

This thesis offers an analysis of the product development processes of a first-tier aerostructures supplier, Spirit AeroSystems. Although this host company provides the context for analysis, the method is meant to be generally applicable to the development of any complex product.

The Design Structure Matrix (DSM) methodology is used to capture the required interaction between tasks of the development of a propulsion structure for commercial aircraft. The task times, time variations, work loads, interdependencies, likelihoods of rework, and learning curves are then quantified and applied to a discrete-event Monte Carlo simulation model which outputs probabilistic completion time and workload of the project. The model is then used to show how changing the customer requirements at different points in the development cycle affect the cost and schedule of development. The failure modes and effects analysis (FMEA) is applied to quantify risks and ensure proper control of their likelihoods and consequences. A holistic industry-level analysis provides insight into the complexities of developing an interdependent product across multiple organizations. Potential recommendations to improve the development process are outlined. Finally, the "Three Lens" methodology is applied to identify implementation obstacles.

This paper builds upon product development process simulation theory by introducing process independent externalities into the model to show how changing customer requirements may impact the cost and schedule of development. It also proposes a new framework for optimal staffing based upon the maturity of the customer requirements. Finally this paper shows that a disintegrated, sections-based design process architecture, like that used for the Boeing 787, is sub-optimal for product development, and it proposes a new architecture for developing aircraft.

Thesis Supervisors:
Steve Pryor – Director of Product Definition, Propulsion Structures & Systems, Spirit AeroSystems
Steven Eppinger – General Motors LFM Professor of Management, MIT Sloan School of Management
Deborah Nightingale – Professor of the Practice, Aeronautics & Astronautics and ESD, MIT

This page has been intentionally left blank.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

First, I would like to thank the Leaders for Manufacturing partner companies for sponsoring my fellowship at MIT and the LFM staff for their effort that enables the program to run seamlessly.

In particular, I would like to thank Spirit AeroSystems for sponsoring my internship, which provides the context for this thesis, and for the opportunity to learn from their organization. Specifically seven people were instrumental to me throughout my time at Spirit. My project supervisor, Steve Pryor, was always helpful when I needed resources and always had time to help me develop my ideas. My project champion, Alan Hermanson, was supportive and encouraging throughout the internship. Tom Greenwood was the perfect sounding board, offering great insight and helping me not to forget the big picture. Tom Scott offered perspective into the company's technical capacities. Mark Hoffman provided an experienced perspective of product development at Spirit. Bill Cook managed the internship arrangements. Finally LuAnn Schaaf made certain that I was integrated and connected within the company and in Wichita. Her friendship alone would have made the internship worthwhile. I greatly appreciate the support of all those involved, which ensured that my experience would be fruitful and enjoyable.

My two thesis advisors, Steven Eppinger and Deborah Nightingale, also deserve much credit for their contribution to this thesis, but also for laying the groundwork that this thesis relies upon. Professor Eppinger is responsible for much of the advancement of the Design Structure Matrix methodology. Professor Nightingale has provided much insight into process improvement in the aerospace industry as co-director of the Lean Advancement Initiative.

I would like to thank several important friends who have profoundly impacted my life at different stages. Although I've been told that one should pick his friends and not let them pick him, I didn't pick any of these friends, and it has worked better than I could have imagined. My friends who have most influenced my life are my brother Brian, Chris Clem, Jeremy Ellis, Derek Brown, Sine Magassouba, Sean Sutton, Jeff McAulay and my fellow Peace Corps G6ers.

I would never have had such great opportunities in life were it not for the sacrifices and hard work of my parents and grandparents. I would like to thank my Mamaw & Papaw Wells and my Ma & Pa Rogers for their hard work that allowed me to grow up a family situation that was never dictated by finances. I would also like to thank my parents. Dad and Dianne, you taught me the value of honesty, work, and doing things right. Mom and Steve, you always encouraged me to dream big and honestly believed that I could accomplish anything. Thank you all for giving me such great opportunities throughout my life.

I would most importantly like to thank my wife, Gina, for her patience, understanding, and unconditional support throughout this time back at school. I am nothing without you, and will love you forever.

I would like to dedicate this thesis to my two beautiful children, Zade Wells Rogers & Mikaya Scaletta Rogers. I pray for your health and happiness every day. May you learn from my weaknesses and mistakes to become better than I. You can change the world if you try.

"Technical skill is mastery of complexity, while creativity is mastery of simplicity."
– Sir Erik Christopher Zeeman

1. INTRODUCTION

1.1 Objective

With the recent fragmentation of the commercial aircraft industry, many aircraft OEM's and suppliers are rethinking their strategic positions to create value. capture value, and minimize risk. Major aircraft OEM's Boeing and Airbus are increasingly outsourcing not only the supply of manufactured components and assemblies, but also their design. Areas that were viewed as simple cost centers when the industry was more vertically integrated have now become areas for differentiation and competitive advantage. The churn caused by the industry's fragmentation has created new opportunities to battle for strategic power. However, a side effect of this high degree of outsourcing is increased complexity. The companies that best understand and control these new complexities will be the most successful in a disintegrated aircraft industry.

Product development of aircraft has seen a great increase in complexity caused by the fragmentation of the industry. Component and assembly designs that are intrinsically interdependent were once collocated, but are now separate. Interdependent product development efforts must now communicate through the opaque corporate veils of the companies that own them – each company striving to optimize its own business and protect itself. It is certainly true that the companies which can master product development in the wake of industry fragmentation will stand to gain strategic power and better position themselves to capture the value created throughout the design chain.

Given the substantial strategic significance of having a mastery of new product development, a number of companies are taking a close look at their product development processes. Typically improvement initiatives come in the form of IT solutions – some address engineering needs (i.e. new CAD and CAE plug-ins and tools) and some address process flow (i.e. ERP) – but rarely do these initiatives drastically improve the costs associated with product development or improve the strategic standing of the company. Managers are still left with certain higher level questions that must be answered in order to make drastic improvements to cost and completion time:

- Are development process and roles optimally organized?
- Where should efforts for performance improvement be focused?
- What are the sources of risk in the development process, and how can they be controlled?
- What externalities frequently disrupt the process, and why are they so disruptive?
- What can be done to increase the robustness of the process to externalities?
- How can the process be structured to create and capture more value in the design chain?

The objective of this thesis is to offer a methodology to understand, model, and improve product development processes and to apply this methodology in the context of a large tier-one aircraft structures manufacturer. Although this research has a definitive focus on the development of aircraft, the methodology is meant to be generally applicable to development of other complex products as well.

Several analytical tools will be used. A design structure matrix (DSM) will be used to model the process by quantifying the performance characteristics of individual process tasks as well as the relationships and interdependencies between tasks. [14] [15] A discrete event simulation will be used to understand the time-related effects of external disruptions such a changing customer requirements. A Monte Carlo simulation will be used to predict probabilistically the outcome of a development process based on uncertain characteristics of the process. This is useful for understanding both the expected time for process completion and the level of risk in the

process, and will be the basis for process analysis. A failure modes and effects analysis (FMEA) will be applied to highlight process risks and ensure measures are taken to reasonably control them. A product DSM will be used to shed light on the complexities created by designing an aircraft across multiple organizations according to sectional architecture rather than systems architecture. Finally, potential recommendations will be presented and an organization-based "Three Lens" analysis offered to evaluate them.

1.2 Outline

- Chapter 1 orients the reader to the purpose and context of this thesis.
- Chapter 2 describes the product development process as a basis for later analysis.
- Chapter 3 summarizes the process and tools used to model the development process.
- Chapter 4 describes how various characteristics of the product development process are modeled, and how the simulation model itself is structured.
- Chapter 5 offers analysis of the nominal product development process through simple network techniques and through simulating the process.
- Chapter 6 incorporates exogenous requirements changes into the process simulation, and offers insight into managing for evolving requirements.
- Chapter 7 offers a tool for managing risk development process risk.
- Chapter 8 provides a holistic perspective of product development in the aircraft industry and identifies architectural features that increase complexity and risk.
- Chapter 9 offers recommendations for improving the product development process.
- Chapter 10 offers an approach for identifying the organizational suitability of an initiative.
- Chapter 11 summarizes findings and proposes opportunities for future work.

1.3 Context

This research was performed during a six-month internship as part of collaborative work between Spirit AeroSystems (Spirit) and the MIT Leaders for Manufacturing (LFM) Program. LFM is a partnership between the MIT School of Engineering, the MIT Sloan School of Management, and numerous large U.S. manufacturing firms. LFM's charter is to discover and teach the principles that produce world-class leadership in manufacturing and operations.

The internship took place during the six months following Spirit's third year after its divestiture from the Boeing Company. During Spirit's transition from having a cost center mentality to that of value creation for the customer, the role of product development also began to transition. Rather than simply serving as a means to defining the product for manufacturing, product development had become a basis for competition in terms of quality, cost, schedule, and even customer service. At the time of the internship, only one new propulsion product had been completely developed. Although this product exceeded expectations for quality, there were opportunities for improvement in the projects cost and schedule. Because of this, Spirit had decided to revisit its product development processes looking for areas to improve performance and for areas to restructure. This initiative served as the basis and context for the internship.

The internship was positioned at two levels within the company. At a grassroots level, the focus was to find and implement specific improvements in product development of Spirit's Propulsion Structures & Systems (PS&S) business unit. At the high level of the Chief Technology Office (CTO), the focus was to find strategic improvements in product development that would apply across Spirit's various business units.

This thesis is meant to synthesize the application of multiple tools to better understand, model and improve the development of propulsion products in the context of Spirit AeroSystems, but also in a manner applicable to development of complex products in general.

1.4 Company Overview

Spirit AeroSystems, Inc, headquartered in Wichita, Kansas, is the world's largest non-OEM designer and manufacturer of aerostructures for commercial, military, and business/regional jet aircraft. Spirit's people, capabilities, and technologies provide customers with products and services in the business segments of fuselages, wings, propulsion products, and aftermarket support. [12]

Spirit manufactures aerostructures for every Boeing commercial aircraft model currently in production including the majority of the airframe content for Boeing's 737 jetliner. Spirit is also the largest aerostructures supplier on Boeing's new 787 Dreamliner aircraft, which is now scheduled to enter service in 2010.

With principal manufacturing facilities located in Prestwick, Scotland, Spirit Europe, the companies wholly owned subsidiary, is the largest independent aerostructures supplier to Airbus providing wing components for the A320, A330, A340, and A380 aircraft families.

Spirit AeroSystems was established in 2005 when Onex Corporation acquired most of Boeing's Wichita-based, commercial aircraft product design and manufacturing capabilities. The company's headquarters are in Wichita, Kansas, with additional operations in the US, UK, and Malaysia. The facilities of Spirit's Wichita site have been producing airplanes and aerostructures since the 1920's. In 2006, Spirit Europe was created as a fully owned subsidiary through the purchase of BAE Aerostructures facility in Prestwick, Scotland. Spirit issued its first public stock offering in November 2006.

For the fiscal year of 2007, Spirit reported a net sales of $3.86B which was a 20% increase from the previous year. Net income in this year was $297M. Total assets of the company are valued at $3.34B. [12] Figure 1.2 below illustrates the breakdown of 2007 financial performance by business segment and major customer.

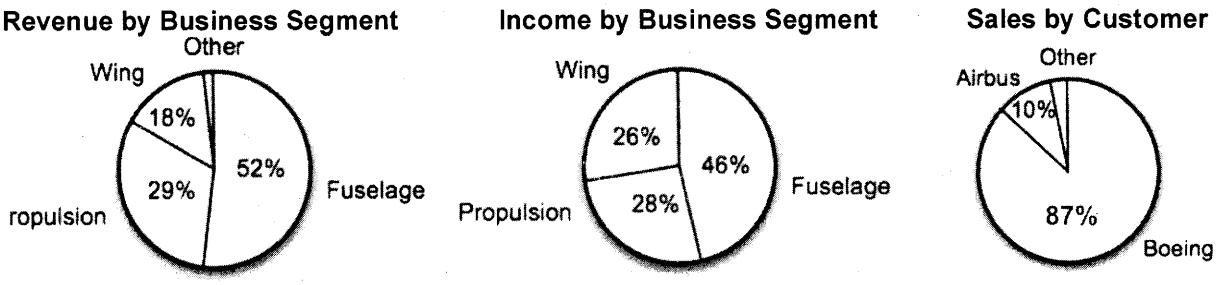


Figure 1.1 - Spirit's Sales Breakdown [12]

Spirit is organized according to several important characteristics, the combination of which defines the relationships between internal groups. These characteristics are functional hierarchy, business segment, geographic location, and customer.

At the top of Spirit's functional hierarchy, the business is arranged into five primary groups: technology, operations, finance, sales & marketing, and human resources. Spirit's primary businesses fall into four major segments: propulsion, fuselage, wing, and aftermarket support.

Spirit serves a variety of aircraft OEM customers. Since many of its customers are competitors, Spirit takes significant precautions to ensure isolation of the intellectual property developed for or owned by its customers. This inevitably leads to the formation of organizational groups around particular customers. In particular, Spirit's strong roots in Boeing are still prevalent, so the company has taken extreme measures to ensure proper isolation of intellectual property between its two largest customers Boeing and Airbus. Spirit customers include: Boeing, Airbus, Gulfstream, Cessna, Mitsubishi, Sikorsky, and others.

Locations in which Spirit owns and operates facilities include: Wichita, Kansas; Tulsa, Oklahoma; McAlester, Oklahoma; Kinston, North Carolina; Prestwick, Scotland; Samlesbury, England; and Subang Jaya, Malaysia.

This thesis is primarily focused on the part of the organization characterized by the following:
- *Function*: Product Development (within the Chief Technology Office)
- *Business Segment*: Propulsion Structures and Systems (PS&S)
- *Location*: Wichita, KS
- *Customer*: Various

1.5 Chapter Summary

This first chapter discussed the purpose and context of this thesis. The objective is to offer a methodology to understand, model, and improve product development processes and to apply this methodology in the context of a large tier-one aircraft structures manufacturer, Spirit AeroSystems. An overview of the company is presented to demonstrate the size and scope of the business. Research was conducted based particularly on the development of propulsion products, which is the focus of the following chapter.

2. DEVELOPING STRUCTURAL PROPULSION PRODUCTS

2.1 Propulsion Products Overview

Spirit currently produces two types of propulsion products – pylons (frequently called struts) and nacelles. The nacelle is comprised of seven primary parts – inlet, apron, fan cowl, engine, engine build up, nozzle, and thrust reverser. Spirit does not design or manufacture the engine itself. The pylon, which attaches the nacelle to the wing, is comprised of three primary parts – torque box, fairings, and systems. The design of these components and assemblies usually takes place in separate teams consisting of employees who are more familiar with one assembly than the others.

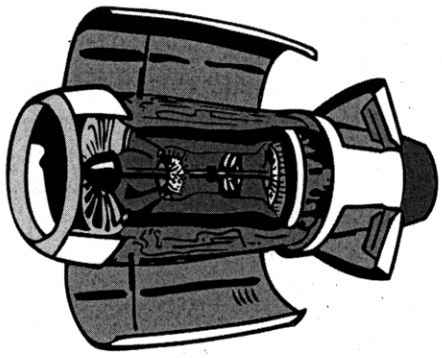


Figure 2.1 - Cutout Nacelle

Figure 2.2 - Commercial Airline Jet

Products can be further classified according to the type of aircraft to which they belong. Spirit's primary revenue is derived from commercial airline jet components. Typically, but not always, the commercial airliners employ under-wing nacelles attached with pylons. These propulsion components are physically large and serve as Spirit's financial and technical base stemming from the company's Boeing years.

As a means of diversification and growth after the divestiture, Spirit began pursuing customers in the market of business jets. Business jets are typically smaller than commercial airliners, generally supporting loads of less than 20 passengers. Business jets usually employ fuselage mounted nacelles, in which case no pylon is necessary. Diversification into this new market has given Spirit access to a much larger number of customers, where as there were only two customers in the commercial airliner market – Boeing and Airbus.

Figure 2.3 - Business Jet

The architecture of a propulsion product is represented in Figure 2.4 below. Interfaces between components are typically defined in the industry as either "structural" or "systems." Structural interfaces imply that some force or load is transmitted between the two components, thereby creating a dependency in the mechanical structure of the two components. Systems interfaces imply that either a signal or fluid is transmitted between the two components, thereby creating a dependency in the operational states of the two components. A typical systems interface is accomplished through connecting wires, cables, and tubing, whereas a structural interface is typically accomplished through rivets, fasteners, and other means of mechanical attachment.

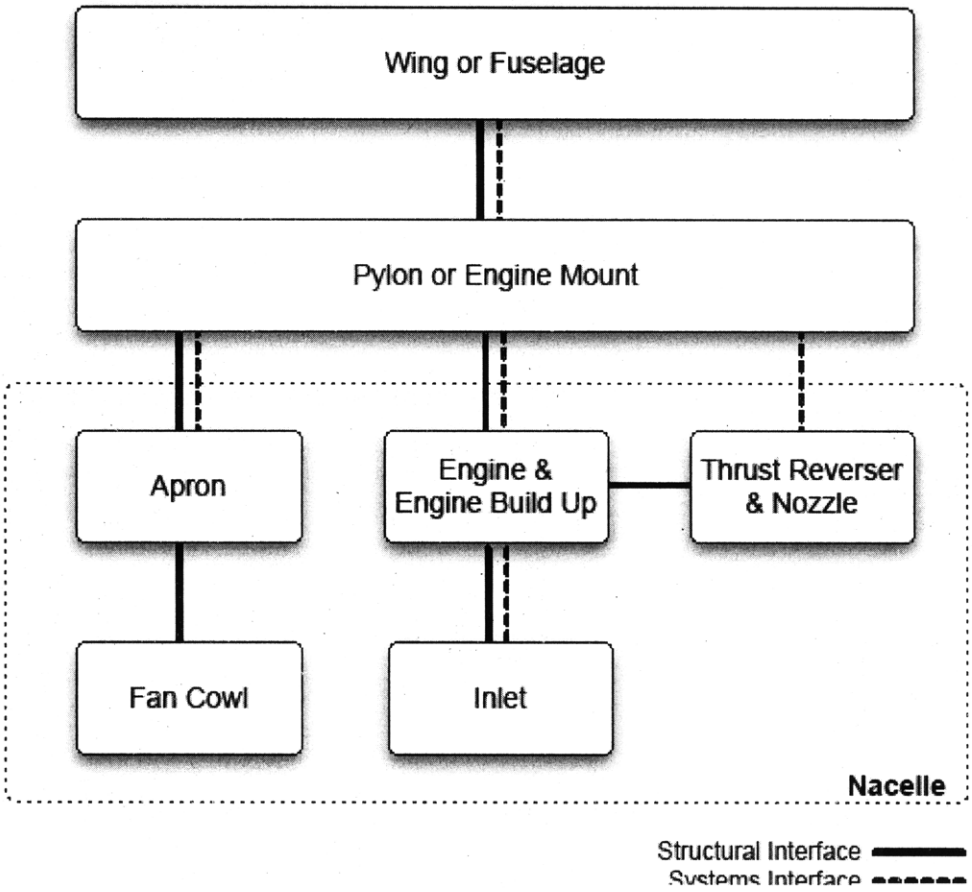


Figure 2.4 - Propulsion Product Architecture

2.2 Product Development Process Description

Design of propulsion structures and systems at Spirit is understood to be a highly iterative, functionally based process. The process begins with concept development, often called the joint concept development phase (JCDP) since the initial conceptual design is meant to be developed alongside the customer. Initial concept development begins early on taking into account preliminary customer requirements and creating an engineering design that will likely meet those requirements. During the JCDP, two other important processes are also taking place. Program planning begins in which the budget, schedule, and resources requirements are created. In addition, an integrated product team (IPT) is formed by selecting high performing individuals from each of the functional groups that will participate in the development process.

The IPT will continually meet throughout the process to review the design and ensure that nothing is being overlooked. Configuration management also begins in which the contractual agreements are formed and accepted between Spirit and the customer. These agreements specify attributes of the product, schedule, cost, and customer interaction that comprise the scope of work for the product program. More colloquially to those participating in the development process, configuration management refers to the customer's design specifications that the product must meet. The JCDP usually takes place completely on Spirit's dime and is viewed as the effort necessary to win a new product contract and iron out the scope of work.

At the finalization of JCDP, the detailed design begins. This is accomplished in teams of CAD designers organized by sub-assemblies and components. The design of the product is acknowledged as a highly iterative process. Many constraints are levied upon the design including weight limits and stress safety limits, which inherently drive the design in opposite directions. When considering manufacturability, supply chain limitations, and overall cost of the product, it is often difficult and time consuming to converge upon a design solution that meets all of the design constraints, let alone optimizes cost. The design group is typically the largest in a product program.

In order to validate the design, a substantial amount of analysis is applied. Since tools typically need to designed and fabricated in order to produce the designed components (especially where composite materials are concerned), testing of physical hardware is replaced by testing and analysis of simulated or modeled hardware. This modeling and analysis is performed by two separate groups – stress and propulsion analysis. The stress group performs all analysis of the capabilities of the design to safely handle various load scenarios and conditions. The propulsion analysis group comprised of experts in all other analysis techniques necessary to evaluate performance of the design ranging from computational fluid dynamic (CFD) methods to fire control analysis. There are two primary goals for stress and propulsion analysis, which sometimes impart different requirements on the groups. In addition to analyzing in order to advance the design, these groups must also provide analysis documentation of the final design in order to meet regulatory certification requirements. Usually, certification requires a more detailed and complex analysis than would be required to advance the design, especially in the early iterations. The stress group is typically the second largest on a product program at about 60% the size of the design group. The propulsion analysis group, however, is not typically assigned to a product program and is treated more as a shared company resource. The only significant link in the process between stress and propulsion analysis is that the resulting load conditions and heat transfer coefficients generated by propulsion analysis need to be incorporated into the stress engineers model. The iteration that takes place between the design group and the analysis groups is labeled as "mechanical model loop" in Figure 2.5 below.

There are three groups that participate in the development process which are primarily concerned with design of production capabilities and processes for the product. The tooling and supply chain groups are involved early in the design process. These groups together make decisions regarding whether to purchase/outsource component supply or to make components in house. These groups interact with the design group early on in the form of high level guidance regarding the feasibility and cost of producing a design. Later in the development process as tool designs and supplier purchase orders are being created, these groups may have more specific recommendations for design changes. Tooling and supply chain are both treated as shared resources in the company, however, one or two members from each group are typically assigned to a product program in order to ensure that their concerns are being considered in product design choices. The third group is the manufacturing engineering group, which is responsible for the design of entire manufacturing process, incorporating the designs

and decisions made by tooling and supply chain. The number of manufacturing engineering personnel on a product program is low initially, but increases near the end of the development process as product and tooling designs become more mature.

Once the product design is finalized, the initial tools are built, and production processes have been defined, fabrication of the initial product hardware begins. This initial prototype hardware is usually outfitted with additional instrumentation to provide feedback from testing. The prototype hardware is then integrated with the engine and tested. Testing is typically performed by the engine manufacturer. It is not usually expected that testing will return negative results, thus testing is viewed more as a final design quality check for certification rather than part of the design improvement iteration. Once all necessary test hardware has been produced and the design passes certification, operational production begins. During the initial phase of production, the product platform is still considered to be in product development since manufacture process improvement is frequent, but once manufacture is at a significant level of maturity, the program is handed over to the operations business function.

Product Development Process

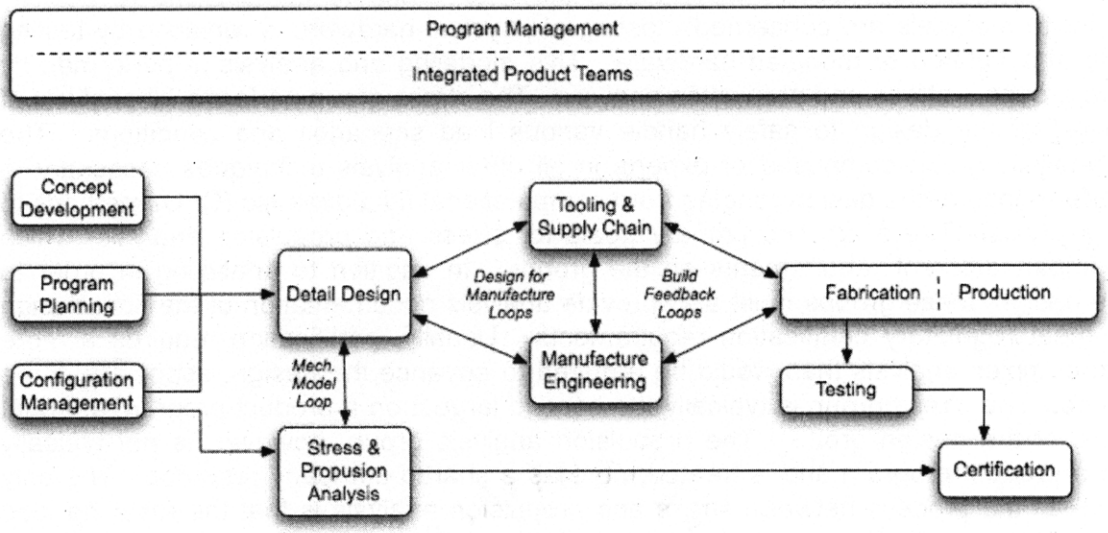


Figure 2.5 - Inter-Functional Information Flow in the Development Process

There are also two major design reviews in the product development process referred to as the preliminary design review (PDR) and the critical design review (CDR). The timing of these design reviews is left somewhat to the discretion of the program manager. Typically PDR is held after the first iteration of detail design and stress analysis, such that tooling and supply chain can indicate feasibility of the design moving forward. CDR is typically held when the design is considered to be about 90% complete, such that tool designs, purchase orders, and manufacture plans may also be evaluated. However, it is sometimes the case that CDR is held earlier in the design's progress since unforeseen changes sometimes occur afterwards which impact many other aspects of the design.

2.3 Design Progression

Feedback is critical for the advancement of any design. In most industries, this feedback typically comes from actual experience with the product. Incorporated into a product development process, this is commonly referred to as the design-build-test feedback loop. Prototypes can be made at a rough level of detail to provide this feedback early in the development process and advance the design very rapidly.

Testing in the aerospace industry, however, does not come easily for two primary reasons. Firstly, many of the components can not be easily prototyped since their size and uniqueness require special tools which need to be developed concurrently with the product design. Secondly, and most importantly, testing of an early stage design can lead to serious injury and death. In years as recent as the 1960's, aircraft test pilots have had incredibly high death rates.

Due to these reasons, the Federal Aviation Administration (FAA) as well as most aircraft design companies requires a substantial amount of design analysis before any hardware is built, tested, or flown. For most aircraft OEM's and suppliers, the design-build-test loop for design advancement has been replaced by a design-model-simulate approach. Elaborate numeric and computational methodologies are used to simulate and predict the behavior and performance of a design, rather than actually testing it in the real world. Much effort has been taken, however, to validate simulation processes against historic test data.

For an aircraft structure, analysis must be performed to gain feedback regarding various failure modes. In the case of propulsion structures, such failure modes may include the accumulation of ice or outbreak of fire. However, the failure mode requiring the most analysis is the yielding of the structure due to excessive loading. The method that addresses this failure mode is referred to as stress analysis. Using this approach, loading conditions (or load scenarios) are defined for the various operational states of an aircraft. Models are created which then translate the assembly level load conditions into component level load conditions. Additional models are then created which allow each designed component to be evaluated for its ability to withstand its specified load criteria. This information is then fed back to the designer, who uses it to improve the design. Figure 2.6 below illustrates the complex interdependence between the design and stress analysis tasks.

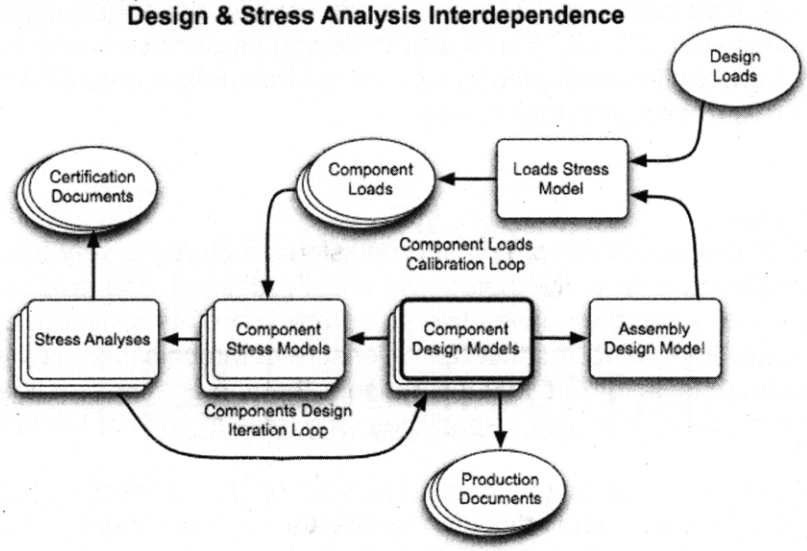


Figure 2.6 - Complexity in Design and Stress Analysis Relationship

Due to the complex level of interdependence in this process, a high degree of iteration is required to properly advance the design. As seen in Figure 2.6 above, the component design is used not only to create component stress models, but also to generate the component load conditions. When the information from the stress analysis is fed back to the designer, the design can advance. However the change of the design also changes the load conditions upon which the design change was based. Typically a given set of component loads is allowed to remain constant, such that the design may advance through several iterations. Once the design has changed significantly, the component loads will be recalibrated such that further analysis takes place with more accurate conditions.

A very important and relevant criteria in the advancement of design is the maturity of customer requirements – in particular, the maturity of the loads. Historically, there may have been as few as 5 load scenarios specified for an aircraft component. In order to compensate for the insufficiency of so few cases, more conservative factors of safety were used, which inevitably increased the weight of the aircraft. Today, with more sophisticated analysis tools thousands of load scenarios may be modeled and analyzed. This increase in analysis has allowed for lower factors of safety in the design (although regulatory safety requirements must always be met), and thus has lowered the weight of structural components. Because there are so many load scenarios, the design can only improve based on feedback from analysis of the loads by which the design performs worst. As the design changes, the most critical load scenarios may also change. To further complicate matters, the load scenarios are usually not well defined at the start of a new program. Instead, new load scenarios and changes to load scenarios arrive throughout the development process. The maturity of the loads by which the design is evaluated plays a critical role in the ability to improve the design against the final customer requirements. It should be noted, though that changes in load scenarios creates rework that is not inherent in the process (since the maturity of the load scenarios is mostly independent of the maturity of Spirit's design), but rather inherent in the scheduling of Spirit's design effort with respect to that of the aircraft OEM.

The final major element in design progression involves the interdependency between design of the aerostructure and the incorporation of various "systems" into that structure. The structures and systems designs are initially created separately and then integrated together. However, constraints on space, load paths, and center of mass usually make the integration of systems into the structure somewhat difficult. This leads to design iteration between the structures and systems. Additional rework for integrating systems and structures is created when the design of structures changes due to changing load scenarios.

2.4 Functional Groups

As seen in Figure 2.5, product development of propulsion structures is very functionally oriented at Spirit. Responsibilities and deliverables are initially defined and assigned to functional groups. Each functional group then drives toward completion of its deliverables using the tools and systems provided. Integrated product teams ensure communication between groups such that they don't operate in isolation. It is in this manner that the final design is achieved. Below is a list of brief descriptions of the roles, capabilities, and tools of each of the functional groups.

Program Management – oversees the entire lifecycle of the product development process including concept development, detail design, manufacture process design, and early stages of manufacture operations. Program management is responsible for all of the program planning including budget, schedule, and resource requirements, as well as establishment of customer contracts and customer relations. At the onset of a product program, management is

responsible for establishment of an initial integrated product team. Typical tools employed include program schedule, progress tracking, and critical path tools.

Integrated Product Team – is comprised of high performing individuals from each of the various functional groups and the financial group. The IPT's fundamental role is to ensure that all of the necessary perspectives are considered during the design of the product. The core members are located in close physical proximity to ensure adequate communication. Ultimately, the IPT is responsible for ensuring that the product meets the customer's requirements as well as the needs of the company.

Design – is responsible for creating and defining the attributes of the product. A three-dimensional model is created using high-end computer aided design (CAD) software. The design attributes of the model are eventually translated into two-dimensional drawings which are released for production. Design and release are controlled electronically, although several software systems are currently in place. The long term goal is to control design and release through an ERP system.

Stress – oversees the analysis of the design regarding capability to safely incur specified load scenarios and conditions. Loading conditions may be defined both by the regulatory agencies and by the customer. To perform this step, typically finite element analysis (FEA) software is employed. The stress engineer must create a new model using FEA tools based on the product design. Loading conditions, boundary conditions, and other physics are then specified within the new model. A solver then outputs the stress levels incurred throughout the model. The results of this analysis serve two purposes – to feed back to the design group for advancement of the product design and to prove to the certifying agencies that the design meets safety requisites.

Propulsion Analysis – oversees the analysis of the design regarding all safety issues other than stress. These include computational fluid dynamic (CFD) modeling and analysis, thermal analysis, anti-ice analysis, fire safety analysis, acoustics analysis, and others. Much like stress, the results of this group are needed for both design advancement as well as certification. Even though PS&S has a stronger need for propulsions analysis capabilities than the other business segments, the propulsion analysis group is treated as a shared resource across the company.

Tooling – is responsible for the creation of tools which are capable of effectively producing the product design and that meet manufacture/assembly requirements. This includes design and production of the required tools. Important factors that impact tool design are ergonomics, safety, maintenance, and process flow characteristics. In addition, tooling provides the design group with preliminary feedback regarding the feasibility and cost of product manufacture.

Supply Chain – oversees the sourcing and procurement of parts and assemblies that comply with the product design. This includes finding and qualifying suppliers for parts as well as performing make-buy decisions with tooling. Contractual structuring with suppliers also takes place through this group.

Manufacturing engineering – oversees the design of the manufacture process in which product components will be received, manufactured, and assembled. Manufacturing engineering defines the steps needed to produce the product with the aim of optimizing product flow through the manufacture process. Much of the process design is driven by choices made by tooling and supply chain.

Operations – is responsible for daily production of the product by executing the plans created by design and manufacturing engineering. This includes managing production employees, maintaining and troubleshooting tooling and machinery, and improving the flow of the product on the manufacturing floor.

Certification – is responsible for ensuring that the product meets the requirements for regulatory certification and for guiding the product through the certification process.

2.5 Intellectual Property

The fruit of Spirit's product development process is a design, the blueprints for the product and for the tools to manufacture and assemble that product. The creation of these designs can be viewed as the creation of intellectual property (IP). Currently, the business model at Spirit treats the development of every product as a new effort. This is largely due to the fact that Spirit does not own the IP for the product design that it creates, rather the customer or aircraft OEM owns the IP for the design of the product. The OEM perceives value in the ownership of the IP so that the IP is not shared with competing aircraft OEM's. However, Spirit would also perceive value in its ownership of the product design IP such that it would not have to begin each new development process from a clean slate and could, thus, offer lower cost and faster development times to its customers. The separation of product IP within Spirit is a very sensitive issue for two of Spirit's customers, the industry's largest OEM's. The IP that Spirit does retain is that for the design of the tools used to produce the designed product. This allows Spirit to concede the IP for the product design since it retains the information needed to produce the product itself. In this manner, the OEM ensures that no other competitor can copy the design of its products while Spirit retains the sole ability to produce the product which it designed.

2.6 Chapter Summary

The second chapter described both the architecture and the development process for propulsion products. The role of functional groups is described, as well as the relationships between these groups in the development process. The high degree of interdependency in the product and the development process, which dictates design progress, is discussed. The chapter ended with a note regarding the creation and capture of intellectual property. A methodology for documenting, modeling and analyzing the development process will be offered in the following chapter.

3. UNDERSTANDING AND MODELING PRODUCT DEVELOPMENT PROCESSES

3.1 Utility of a Product Development Process Model

A model is an abstract representation of something that exists in reality. It is never possible to create a model that fully captures every aspect of its subject, and there should never be reason to attempt to do so. The real purpose of model creation is to capture the important factors such that the results output from the model closely approximate reality. If this can be achieved, models can provide much insight into their subjects.

In the case of product development processes (PDP's), modeling can provide several valuable benefits. Firstly, a PDP model can provide a much more rapid response to stimuli than an actual development process of a real product. In fact, product development of propulsion products can typically take several years to complete, in which case feedback from an improvement initiative may take years as well. In addition, it would be difficult over this extended period of time to attribute changes in performance to a single improvement initiative since it is nearly impossible to isolate this one variable and hold everything else constant in a real life development process. Thus a model allows the manager to isolate single variables and receive timely feedback regarding the anticipated effects of a change or improvement initiative. A good PDP model can serve as a sandbox for experimenting with potential process changes.

Secondly, manipulating and experimenting with a real life product development process entails the assumption of risk. It is typically not easy to understand the impacts of a change in such a dynamic system, thus there is a chance that a change may yield negative results. Through the use of a model, both the expected outcome and the potential risks may be predicted such that better decisions can be made without as much costly experimentation with the real process. In addition, a PDP model can serve as a quantitative justification for ideas and improvements that would be difficult to obtain support for, since stakeholders would not feel that they are assuming as much risk.

Finally a relevant PDP model can provide a better basis for estimation of development time and cost before initiation of a real product development process as well as during an ongoing process. The information yielded by a model that simulates development of a potential product can be used to help managers make better business decisions and structure more appropriate contracts. During an ongoing process, a manager could use a model to better quantify the schedule impact of a product change order. Having the information that a model provides can be invaluable in making appropriate business decisions.

3.2 Design Structure Matrix (DSM)

The tool used as the basis for modeling the development process is the design structure matrix (DSM). The DSM is a modeling tool that represents the relationships and dependencies between components of a system, product, or process. In the case of a product development process, the DSM indicates the flow of information between development activities in the form of a matrix. [14] Information flow in development processes dictates not only workflow, but also the rate of design iteration and evolution. Therefore, DSM can allow for incorporation of process dynamics that are often overlooked when planning or predicting a product development process. The DSM was first introduced by Steward and captures coupling and dependency between the design tasks of a project. [15] The tasks listed along the left column of the matrix represent design activities that receive information, while the same tasks listed along the top row represent design activities that provide information. An off-diagonal mark located within the matrix denotes dependence and coupling between two design activities. Steward's original

model is also referred to as a Binary Design Structure Matrix because each cell in the matrix represents a binary choice of coupling and dependency. For example, the process depicted below, where each process task is represented by a letter, can be represented as a 6x6 binary DSM. [9]

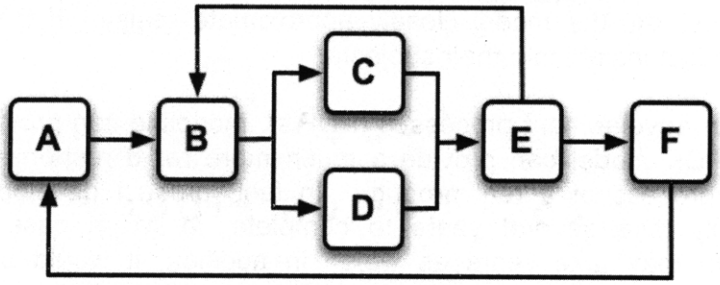


Figure 3.1 - Simple Example Process [9]

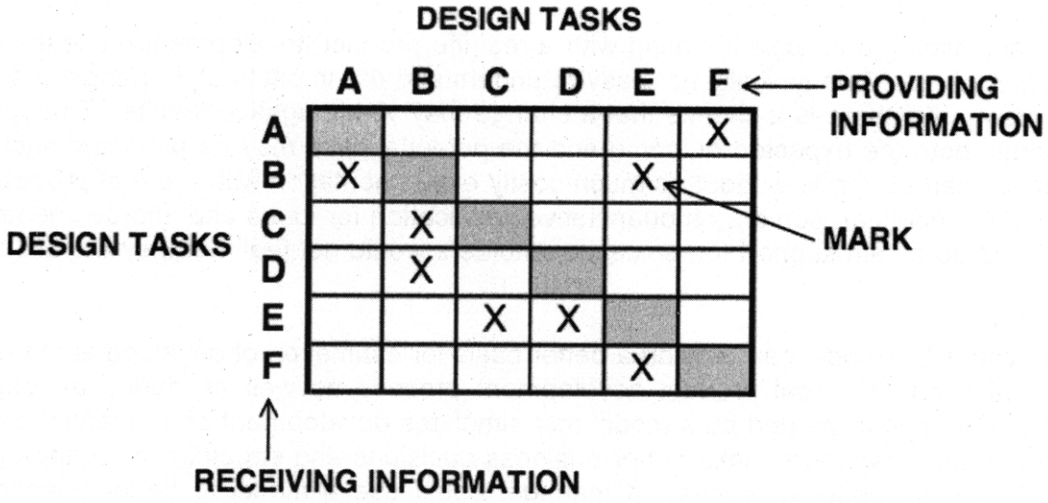


Figure 3.2 - Binary DSM for Example Process [9]

The mark located in the third row and second column denotes task C's dependence on task B to be completed before it can be executed. Looking down a task's column, one can easily determine those tasks that are dependent upon it. For example, looking down the fifth column, we see that task B and task F are dependent on task E to be completed. Looking across a task's row, one can easily determine those tasks upon which it is dependent. For example, looking across the fifth row we see that task E is dependent on task C and task D to be completed. Note that if the tasks in the DSM are ordered sequentially, the marks located above the diagonal represent feedback (information transferred from later tasks to earlier tasks) and the marks located below the diagonal represent feed-forward (information transferred from earlier tasks to later tasks).

The DSM may be treated as both a qualitative and a quantitative tool.. An extension to Steward's work was introduced by Eppinger, Whitney, Smith, and Gebala where the off-

diagonal marks are replaced with numerical measures of coupling and dependence (or some other metric that measures an inter-task relationship), while the on-diagonal mark measure task duration (or some other metric that characterizes an intra-task relationship). [6] This Numerical Design Structure Matrix captures task interrelationships at a much deeper level than its binary counterpart, in addition to capturing completion time. Although the binary DSM can be very useful to a project manager as a visual representation of the process, task dependencies, and risks, the numeric DSM provides a much deeper insight into the impact of the structure of the process on overall completion time, costs, and risks.

As an example, Figure 3.3 below shows a 6x6 numerical DSM representation of the binary DSM shown above. The number 2.7 located in the second row and fifth column denotes the relative strength of dependence of task B upon Task E. The number 7 located in the third row and third column denotes a duration of 7 days for Task C to be completed. [9] If human resources constrain the process, duration may be replaced by workload (in man-days) in order to allow for variable durations according to the staffing level of a given task. Workloads are used when applying the DSM to Spirit's development process.

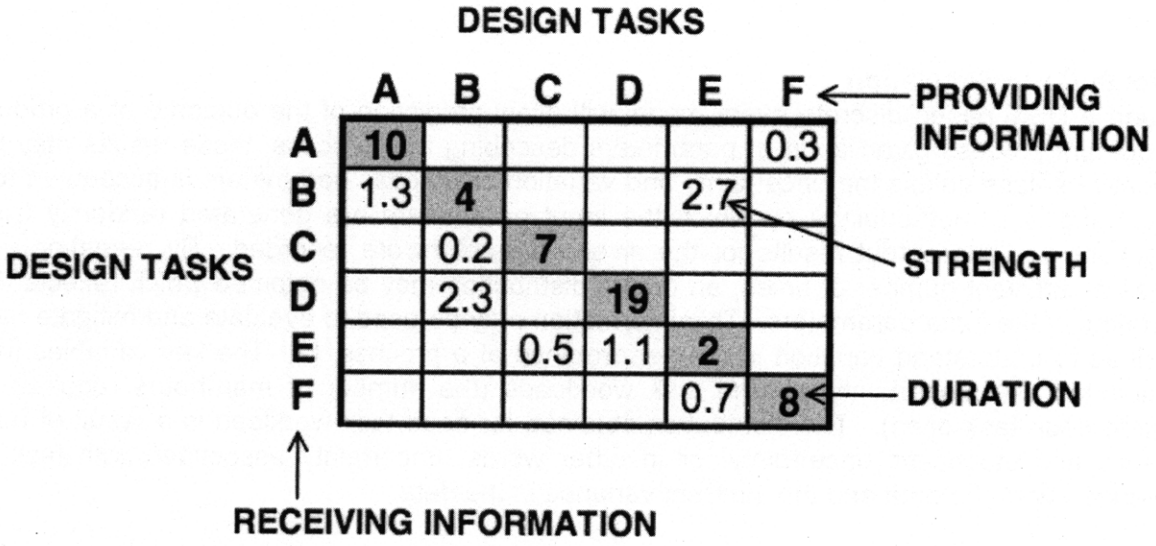


Figure 3.3 - numeric DSM for Example Process [9]

As evidenced from these simple examples, the DSM is an effective tools for capturing the structure of a system and visually representing it concisely and compactly. [9] A significant aspect of the product development process that numeric DSM neglects is quality. Firstly, the DSM assumes that the quality of the final design is fixed. In addition, the DSM does not account for variance in product features or technology from the development of one product to the next. Thus, the numeric DSM is most useful when applied to a stable product platform in which the underlying structure of the product and the process do not change. It is also most useful when applied to a process that develops products with similar design quality requirements.

3.3 Discrete Event Simulation

Discrete event simulation is a modeling methodology that simulates a process as a discrete-state, event-driven system of which the state evolution depends entirely on the occurrence of asynchronous discrete events over time. [4] In a discrete event model of a system, the state of

the system can be changed by events which occur at given points in time. However, the timing and impact of each event may depend upon the state of the system when each event occurs. A discrete event simulation operates in a manner in which events are executed one at a time, each time re-evaluating the state of the system and then the timing and impact of the subsequent event. Because the initiation and re-initiation of tasks, the completion of tasks, and the changing of requirements are all events in a development process which occur at a given point in time and which change the state of the development process itself, discrete event simulation should be a relevant tool for modeling this type of process. The manner in which the discrete event model operates for a product development process is by:

1. initiating the first task(s) in the process by staffing them;
2. finding the next task start, task completion, or requirements change based upon the current time and state of the process and its tasks;
3. advancing a clock by the time until the next event;
4. updating the system and task staffing to the new state;
5. repeating the previous 2-4 until all tasks have converged and no requirements changes remain.

3.4 Monte Carlo Simulation

Although a DSM-based discrete event model will allow prediction of the outcome of a product development process given a set of parameters describing that process, these results may be practically useless unless the uncertainty and variation of process parameters is accounted for. Monte Carlo is a methodology by which the input parameters are generated randomly from distributions and the output results for the given parameters are recorded. By repeating this process a sufficient number of times, an output distribution may be obtained which reflects the uncertainty of the input parameters. This information may be used to evaluate and mitigate risk, as well as to understand variation in the performance of a process. [2] The key variables that are modeled with uncertainty are the task workloads (the number of man-hours required to complete each task once). The distribution obtained for each task workload is a result of both epistemic and stochastic uncertainty, or in other words, uncertainty associated with lack of knowledge about the data and the inherent variance in the data.

3.5 Creation of a DSM-Based Simulation Model

The construction of a quantitative DSM model entails five primary steps. Firstly, all tasks which occur in the development process must be accounted for. This is not to say that every task must be at a high level of detail, but rather that every task must appear in the DSM even if it is recognized as a sub-task of a larger task. During this step the modeler must trade off detail against abstraction. Additional detail may increase the realism and accuracy of the model, but this comes at the expense of additional complexity and model processing time. The modeler must find the appropriate balance between detail and abstraction in order to achieve the greatest level of usability and usefulness.

Secondly, once all tasks have been identified, data must be gathered regarding the performance characteristics of each task in isolation. This is the information that is inherent in the task itself, regardless of its role in the process. When treating the task in isolation, as such, the terminology used is "once-through work." This entails that the task performer has access to timely, accurate information and is not required to rework or restart the task. It is often difficult to gather accurate data for once-through work since the work for tasks is never completed in isolation. Primary data collected for once-through work are the estimated completion workloads

(in man-hours) expressed as a probability distribution. Also collected are the task learning curves, which again are inherent in the task and not the task's role in the process. The typical maximum number of personnel assigned to each task must also be collected in order to reflect the actual human resources to which the development process may have access.

Thirdly, data must be gathered regarding the relationships of each task to all of the other tasks in the process. This is the information regarding the task which is inherent in the task's role in the process. The terminology used to describe the relationships between tasks is referred to as "dependency." This task dependency data can be difficult to gather since it forces the task performer to completely evaluate his or her role in the process, something that many task performers have never before done. The primary data collected for task dependencies are the strength of precedence and rework dependencies. Strength of precedence refers to the degree of concurrency which may take place between two tasks. Rework dependency refers to the degree of rework that a task must perform, given that the information from an input task has changed.

Next, externalities that impact the development process must also be identified. The information captured in the previous three steps is sufficient for modeling the ideal process. However, there may be other factors which are not inherent in the tasks or the process, but which may greatly impact the outcome of the development process. The externalities captured by this model are changing customer requirements (usually in the form of engineering change orders), unplanned customer audits and reviews, and human resource constraints. Each of these externalities creates unique effects on the outcome of a development process, especially since their impact may differ depending upon the timing of their introduction to the process. These externalities do add a significant degree of complexity to the model, however neglecting them would produce results which would not meaningfully approximate reality.

The final step in creating a quantitative DSM-based model is to define an algorithm which utilizes the data to predict an outcome. The methods used in this model are Monte-Carlo Simulation and discrete event simulation. Monte Carlo simulation will allow the model to run many times based on different probability selections for tasks. This should portray the variability in the outcome of the development process. The discrete event simulation will allow for the dynamics introduced by the timing of changing requirements to be properly captured. Once complete, the model should be calibrated against historical PDP data in order to ensure its accuracy.

One aspect not considered in the creation of this model is task error - the quality of the output of the design is considered to be solely dependent of the quality of the information input. In other words, errors in task completion are not incorporated into this model. Although mistakes happen in reality and affect the likelihood of rework and iteration, this data proved extremely difficult to gather. Since most task data was gathered from task performers, most were not willing to concede that mistakes are made. Thus this model will provide all iteration that is inherent in the design process and not the quality of work of the task performed.

3.6 Process Tasks

When contemplating, discussing or modeling a process, it is useful to divide the process into modules, referred to here as tasks. Sometimes the basis of this division is intuitive, as in the case of a typical manufacturing process, which is usually predetermined to perform steps one by one until a product is made. Product development, on the other hand, can be a very human intensive process in which steps and procedures are not always well designed or documented,

and sometimes occur in a de facto manner. It is often even difficult to monitor advancement of the design, which is really information rather than something physical. Thus, there is a high degree of discretion involved when documenting tasks in a development process.

Extreme detail can be captured by documenting thousands of tasks, or all of these detailed tasks could be pooled together to represent tasks at a more macro level. The tradeoff between detail and abstraction is also one between complexity and simplicity or usefulness and usability. A general rule of thumb was used in determining the appropriate level of detail – if a series of tasks could be grouped together such that other functional groups only interacted with the first and last task in the group, these tasks were listed as a single task. This level of modularization was intended to eliminate confusion when gathering task level data regarding dependencies. Thus, if a task in the middle of a group of tasks is dependent upon another functional group, this dependency could be isolated and not masked within an overly abstract task. Also, tasks that seemed to represent a large portion of the work were generally broken out into their subtasks, thus the workloads of each task would be on the same order of magnitude which would reduce sensitivity to error in the data gathered for each of those tasks.

3.7 Task Performance and Interdependence Characteristics

The methodology used predicts the outcome of a process on the basis of the characteristics inherent in each task performed and of the characteristics inherent in the role of the task in the process. This data was gathered in two separate phases to avoid confusion between addressing tasks in isolation and addressing task interdependencies. Below are the attributes of the gathered data described in detail.

Task Once-Through Completion Work – is the amount of work in man-hours required to perform a task one time with access to accurate information and with no rework. This data was collected at three points – the most likely, the pessimistic (corresponding to 90^{th} percentile), and the optimistic (corresponding to 10^{th} percentile). This data was later used as the basis for a triangular distribution characterizing the likelihood of completion work for a given task.

Task Learning Curve Factor – describes the decrease in required work necessary to complete subsequent iterations of a task. It is necessary to decrease the once-through completion work of a task with each iteration in order to account for learning.

Task Dependency – indicates that a task is directly dependent upon the information output from another task. Thus, task dependencies are used in a qualitative or binary DSM to indicate information flow from and to tasks.

Concurrency Factor – describes the degree in which two tasks must occur sequentially or concurrently. A concurrency factor of 1 would indicate that the tasks may occur completely in parallel or that they may begin at the same time. A concurrency factor 0 would indicate that the tasks must occur completely sequentially or that the dependent task must wait for the input task to finish entirely before beginning.

Rework Factor – indicates the impact of reworking input tasks upon a given task. A rework factor of 1 would indicate that any percentage of rework of the input task would be completely passed to the dependent task. A rework factor of 0 would indicate that reworking of the input task would not create rework for the dependent task.

Iteration Likelihood – is the probability of the reworking of an input task causing a given task to rework. For the purposes of this model, all dependencies are assumed to have a 100% iteration likelihood, except for dependencies classified as "failure modes". Each failure mode has some probability that it will occur in the development process and may only occur once during the process. Typically failure modes have a low likelihood of occurrence, but may cause vast amounts of late rework.

3.8 Data Collection

Collection of accurate data for a development process model is crucial for the relevancy of the model, but can pose one of the largest challenges. Data collection for this model was undertaken in five phases. The first phase was process documentation. This phase primarily focused on identification of tasks performed in the development process. The method used for data collection in this phase was mostly informal discussion with key individuals in the process. Contacts, which were deemed to have some expertise, were assigned to contribute information for each task. In total, forty-six contacts were listed, each having an expertise in as few as one task, or as many as seventeen. Each of these contacts was given an opportunity to provide feedback regarding the detail, accuracy, and comprehensiveness of the list of tasks. Additionally, all contacts were initially given an orientation to the model methodology, purpose, impact, and timeline in order to create awareness about the end use of the collected information, as well as to create more "buy in" for the project.

The second phase consisted of gathering once-through task performance characteristics. Task performers were asked for three types of data. Firstly, they were asked to offer a brief description of their task, which would be used to verify that the collected data was indeed for the task which was specified. Secondly, they were asked to offer expected once-through work man-hours. This data requested was based on pessimistic (within 90% confidence), realistic (within 50% confidence), and optimistic (within 10% confidence) expectations. Finally, the contacts were asked to provide data reflecting the learning curves inherent to their tasks. The duration of the interviews was between to be 5 to 15 minutes for each task. After this data was collected and reviewed, additional interviews and discussions took place regarding data that was not anticipated and was somewhat surprising. Managers from each functional group checked the quality of the data collected for each task and adjusted the data so that is was consistent and could potentially be scaled in aggregate for model calibration later.

The third phase consisted of gathering task dependency characteristics. Functional group managers were asked for three types of data. Firstly, they were asked to identify all tasks upon which the specified task is directly dependent for information and to qualify these dependencies as either "necessarily concurrent" or "not necessarily concurrent" tasks. Secondly, they were asked to assign a concurrency factor to each of the tasks upon which the specified task was dependent. This factor was interpreted as the degree of precedence that one task had over the other, or the amount at which the tasks could occur in parallel. Finally, the functional managers were asked to assign a rework dependency factor to each of the dependencies. Rework dependency was interpreted as the degree to which the specified task must be reworked given a complete change in information from a dependent task. This data was gathered during a one-day workshop where each functional manager was present and inter-functional dependencies could be easily discussed as necessary. This approach helped to ensure that dependencies were not overlooked and that confusion could be easily addressed. To facilitate the workshop, each attendee was given a printout of the binary process DSM and was asked to quantify the dependencies associated with their functional group by writing these factors next to each binary

relationship. After this data was collected and reviewed, additional interviews and discussions took place regarding data that was not anticipated and was somewhat surprising.

The fourth phase consisted of determining and quantifying significant externalities. Determination of the major externalities occurred by informal discussion throughout the other data gathering phases. However, effort was made to formally discuss and document these issues, as well as quantify them. This took place solely through interviews and discussion with key personnel and managers.

The final phase consisted of data calibration. In this phase, key individuals and managers were asked to look over the data collected and assess the accuracy. For the data that was identified as inaccurate, interviews were scheduled to revisit the justification of the data. This phase is necessary to compensate for data that may have been provided in an overly conservative, overly liberal, or erroneous basis. An additional calibration phase was performed based upon historical PDP data in order to ensure that output of the model was reasonable.

In addition, data was collected from program managers in the various business segments of the company. This data, however, was of a more qualitative nature to understand the underlying issues associated with developing products in this industry. Each program manager was asked to present on the perceptions of the problems and the lessons learned from their development processes. This information was later evaluated by a separate team in order to create recommendations for changes and restructuring of the process.

3.9 Chapter Summary

The third chapter described the Design Structure Matrix as a very powerful tool for documenting, understanding, and modeling the development process. An overview of DSM is given since it will serve as the basis of analysis and modeling. Discrete event simulation and Monte Carlo simulation are described as the modeling methodologies for the development of a product development simulation tool. The various types of data required for the simulation model are discussed, as is the methodology for data collection. The means by which this data is used to simulate the development process is the focus of the following chapter.

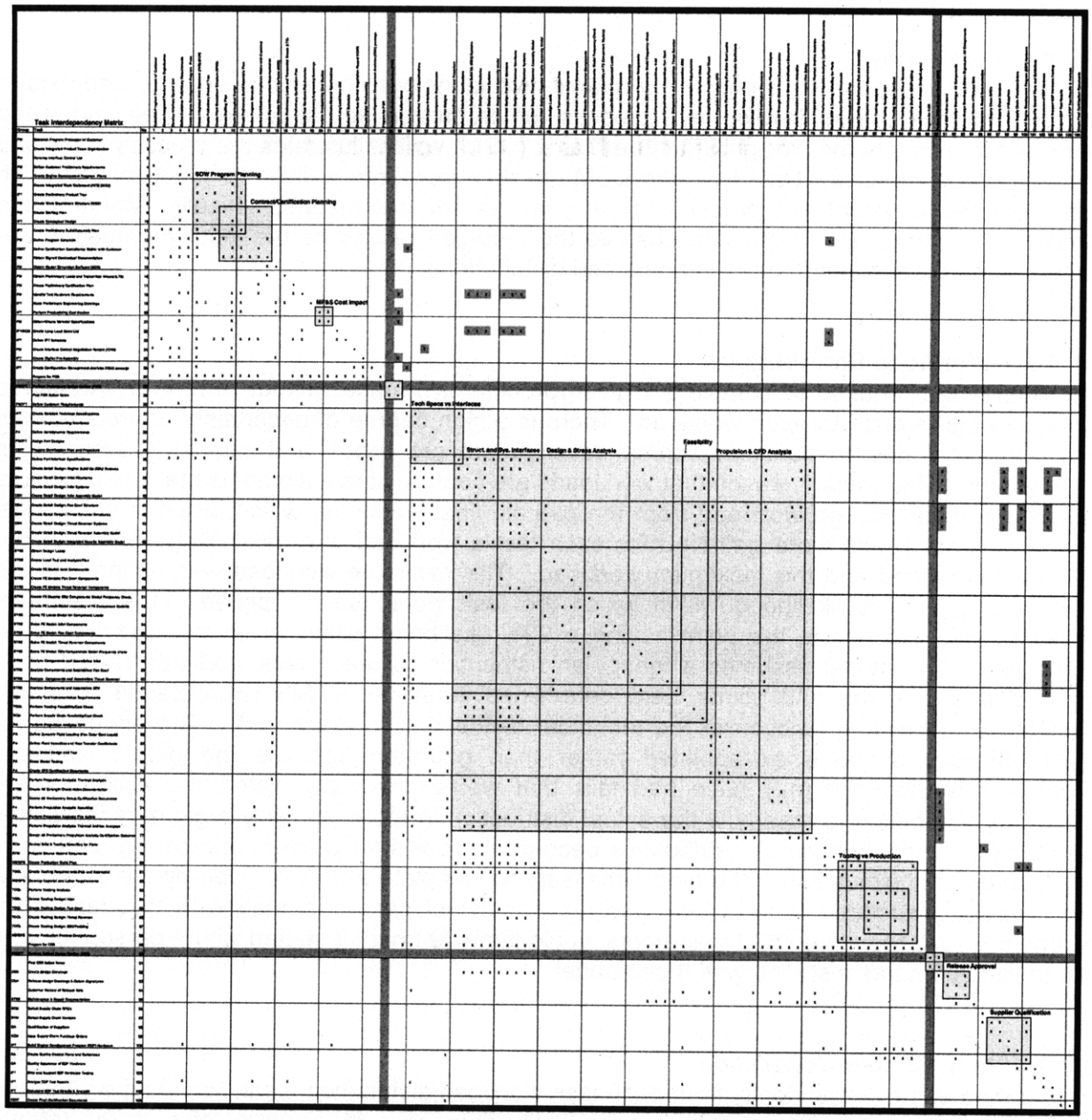


Figure 3.4 - Binary DSM for Development Process of a Nacelle

(See Appendix C for an enlargement of this figure.)

4. MODEL DESCRIPTION

4.1 Model Overview

As discussed in the previous chapter, a quantitative model was created in order to understand the performance of the development process for propulsion aerostructures. The underlying basis of the model is the Design Structure Matrix (DSM) which illustrates the process tasks and their interdependencies. Data from the DSM are then input into a discrete event model which simulates the development process for a given set of parameters. Finally, Monte Carlo simulation is performed in order to observe the modeled results as the input parameters are varied randomly according to their understood distributions. [2]

4.2 Task Workload Distributions

The man-hours required to complete a given process task once without rework or iteration is called the task's once-through workload. There is a high degree of uncertainty associated with this parameter due to variation across development processes and due to uncertainty in estimating it. The primary reason that workloads are so difficult to estimate is that it is difficult to isolate the once-through workload from the overall, final workload, which includes substantial rework. Three points were gathered for each task's workload: the most likely workload, the minimum workload and the maximum workload. The minimum workload was defined as the point having a 10% likelihood within which the task would be completed. The maximum workload was defined as the point having a 90% likelihood within which the task would be completed. The model assumes a triangular distribution for each task workload based upon these points, and the end points were determined iteratively in MS Excel based upon the cumulative distribution function of the triangular distribution for each task. The endpoints for these distributions were extrapolated rather than gathered because the actual workload probability distributions may have had tails that would drastically distort the ability for the triangular distribution to resemble the actual distribution in the most likely regions of the actual distribution. These workload distributions become the basis for random number generation of the Monte Carlo simulation. The time necessary to complete a task is given by the product of the remaining workload for that task and the number of personnel staffed to that task. For example, a task requiring 1600 man-hours would last 400 hours if staffed with 4 personnel, or it would last 200 hours if staffed with 8 personnel.

4.3 Random Number Generation

Rather than sampling randomly from each task workload distribution, Latin Hypercube sampling (LHS) was employed. This approach drastically decreases the number of samples (Monte Carlo iterations) required to reduce the variance of the output mean compared with sampling randomly. Thus, the time required for convergence of the final development process performance estimators is also drastically reduced, making the model much more time efficient. [10] One thousand samples were used for this simulation, which required about two hours to run.

The LHS process, shown in Figure 4.1 below, divides the sample probability density function into equally likely intervals and randomly samples within each interval. Thus, the number of samples is equal to the number of intervals by which the distribution is divided. Calculating the division points of the distribution is accomplished by setting the difference between the output of the cumulative distribution of each division point and the previous division point equal to 1 divided by the number of samples. One sample is taken randomly from each interval, and these samples are then randomly sequenced. [5]

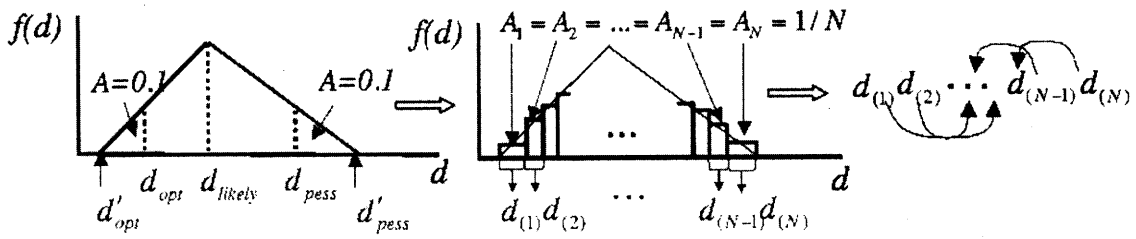


Figure 4.1 - Latin Hypercube Sampling Process [5]

4.4 Event Types

Three types of events are acknowledged in the discrete event model, each of which has ramifications to the state of the system: task initiation, task completion, requirements changes. Task initiation implies that the task is queued to receive staff as soon as possible. Task completion implies that subsequent tasks now have new information available and that the staff for that task is reallocated to a human resources pool to work on something else. Requirements changes imply that the percent of completion for a given task is actually less than previously perceived, and that the task must initiate rework if it had previously completed. One important attribute of requirements changes is that they are time dependent rather than process dependent. That is to say that task initiations and completions occur as an outcome of the previous state of the process, while requirements changes occur at a fixed point in time irrespective of the point in the process. Incorporation of requirements changes can therefore be interpreted as the incorporation of process externalities.

4.5 Next Immediate Event Determination

One major step in the discrete event model is the method to determine the next immediate event in which to staff and execute. This turns out to be a relatively costly computation based upon the number of process tasks and task dependencies. In order to calculate the next immediate event, first all subsequent tasks of all current tasks in progress are listed. All of these subsequent tasks whose other input tasks have not reached completion are removed. Then the remaining subsequent tasks are ordered based upon the amount of time remaining for their single input task that is not completed, and the top one(s) with the least time to initiation are kept. To this is added the nearest task completion and the nearest requirement change. The list is again truncated to the event(s) occurring soonest. Thus, the next immediate event and the time remaining for that event are calculated.

4.6 Human Resource Constraints

Two types of human resource constraints are accounted for in the model. The first is a task human resource constraint, which is the maximum amount of staff allocated to that task. The second is the maximum staff available within each functional group. The human resource allocator in the model attempts to provide each initiated task with its maximum staff level. However, if the staff pool for the functional group to which a given task belongs cannot provide staff at that time, it staffs the task as highly as possible until more staff become available. If multiple tasks are competing for available human resources, the human resource allocator prioritizes based on the chronology of first initiation of tasks (the lowest task number). The allocator will not, however, remove staff from a task in progress in order to put it on a new task.

Several important weaknesses of the model fall into the category of treatment of human resources. Firstly, resistance of staff transitions is neglected. This allows staff to be transferred in any number at any time to a new task with no cost in time. In reality, moving human resources does not occur with perfect efficiency. Secondly, there is no time or process dependency in the staff available for each functional group. In reality, the staffing level of functional groups is determined by management and may vary over time as the process ramps up or ramps down. Finally, there is no resolution of expertise within each functional group. Instead each functional staff member is deemed equally qualified as the others in that group. Nevertheless, the human resource allocator does provide enough functionality to yield meaningful results.

4.7 Learning Curves

Learning within the development process is modeled through learning curve factors for each task. After each iteration of a task, the workload for that task is reduced by a learning curve factor until it has met its point of maximum learning. So as the process proceeds, the amount of work required to complete a given task may be substantially reduced from its original value. [5]

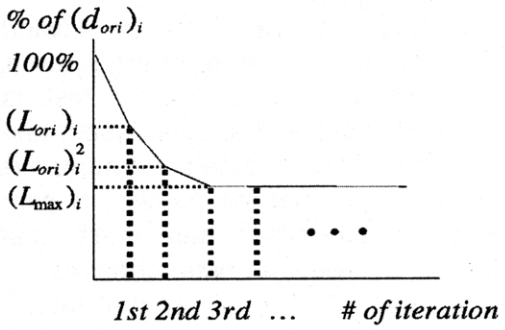


Figure 4.2 - Task Learning Curve

4.8 Concurrency

The degree to which tasks may occur in parallel is an important factor in determining the duration of the product development processes. The model assumes that tasks which do not exhibit dependency may occur completely in parallel. For tasks that do exhibit dependency, the concurrency factor provides the extent to which the dependent task can occur in parallel to the input task. For example, if task B has a concurrency dependency of 0.5 upon task A, then task B must wait until task A is 50% complete in order to begin. Additionally, concurrency in the model only occurs during the first iteration. This models the effect that once work is complete, it will not begin again until it receives changed information, and that changed information is only passed on once the task is complete. This may be viewed as a scenario of de facto information flow which is typical in a large development process in which most tasks are performed out of the immediate vicinity of its dependent tasks. Rework concurrency was perceived to be minimal in the actual process as well. Also, there is no cost of concurrency assumed since the once-through workloads for each task should have accounted for this. Thus a task that is begun in parallel to an input task does not have to update its work as the input task completes its work.

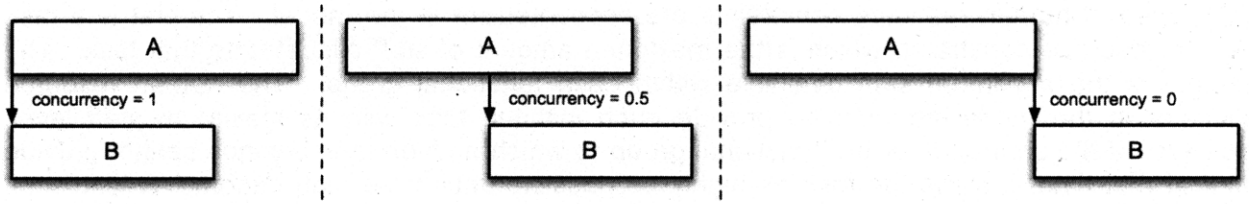


Figure 4.3 - Dependent Tasks with Varying Degrees of Concurrency

4.9 Rework

Task rework and iteration are colloquially viewed negatively during the product development process. Many companies judge the effectiveness of their development processes by "first pass" design quality, viewing subsequent rework as waste. However, this is a narrow view of rework and iteration. Although it is true that rework due to mistakes should be minimized, some rework is inherent in the process due to the product architecture. This rework is primarily caused by interdependency in the development process and should be facilitated rather than eliminated. For the purposes of this model, the rework dependency factor is defined as the fraction of rework generated for a given task when its input task is reworked completely. An underlying assumption that is applied in order to model these relationships is that all tasks are homogeneous. That is to say that if the rework dependency of task B upon task A is 0.5, that task B is subject to 50% of whatever fraction of rework task A is exposed to. Rework caused by error is neglected from the model since it is perceived to be relatively small compared to inherent rework and that data was nearly impossible to obtain in the given timeframe.

4.10 Failure Modes

There were certain dependencies that were not typically planned for since they had a small likelihood of occurrence for a given development process. These dependencies were deemed process "failure modes" which may cause substantial amounts of rework. Most of these failure modes occur later in the process when design maturity is perceived to be high. Examples typically include inability to manufacture the design or failure of certification testing. These failures were modeled with a likelihood of occurrence in each program and with rework and concurrency factors if they do occur. Much of risk management focuses on reducing the likelihood of failure mode occurrence and the amount of rework from their occurrence.

4.11 Convergence

Since nearly all of the dependencies have a 100% likelihood of occurrence, the discrete event model could continue infinitely to pass miniscule amounts of rework throughout the system. In reality, once tasks begin to converge either the small differences can be flushed out in a meeting, or the small differences become negligible and the process is stopped. This model assumes a convergence criteria for each task. If the queued rework fraction does not surpass the convergence criteria, then the task is not initiated. Once all tasks are in a converged state and there are no more requirements changes, the modeled process has completed. The choice of convergence criteria is difficult since convergence often depends upon the details for a given task. For our model, the convergence criteria is constant across all tasks.

4.12 Model Logic

Figure 4.4 below depicts the model logic at a relatively high level. MATLAB was chosen as the programming language since the matrix structure of variables is aligned with the DSM structure and can drastically reduce computational time. The process begins at the upper right portion of the figure with data pertaining to the Monte Carlo simulation. If the number of simulated process trials is less than the number specified, a new trial will be initiated based on the data pertaining to the development process. Each trial is a complete run of the discrete event model, which iterates (in the large left-most loop in the figure) until there are no more events for that trial. When the trial is complete, the observed metrics from that trial are recorded and the simulation process continues. Once the required number of trials is reached the results are formatted and displayed. See Appendix A for MATLAB code created for this model.

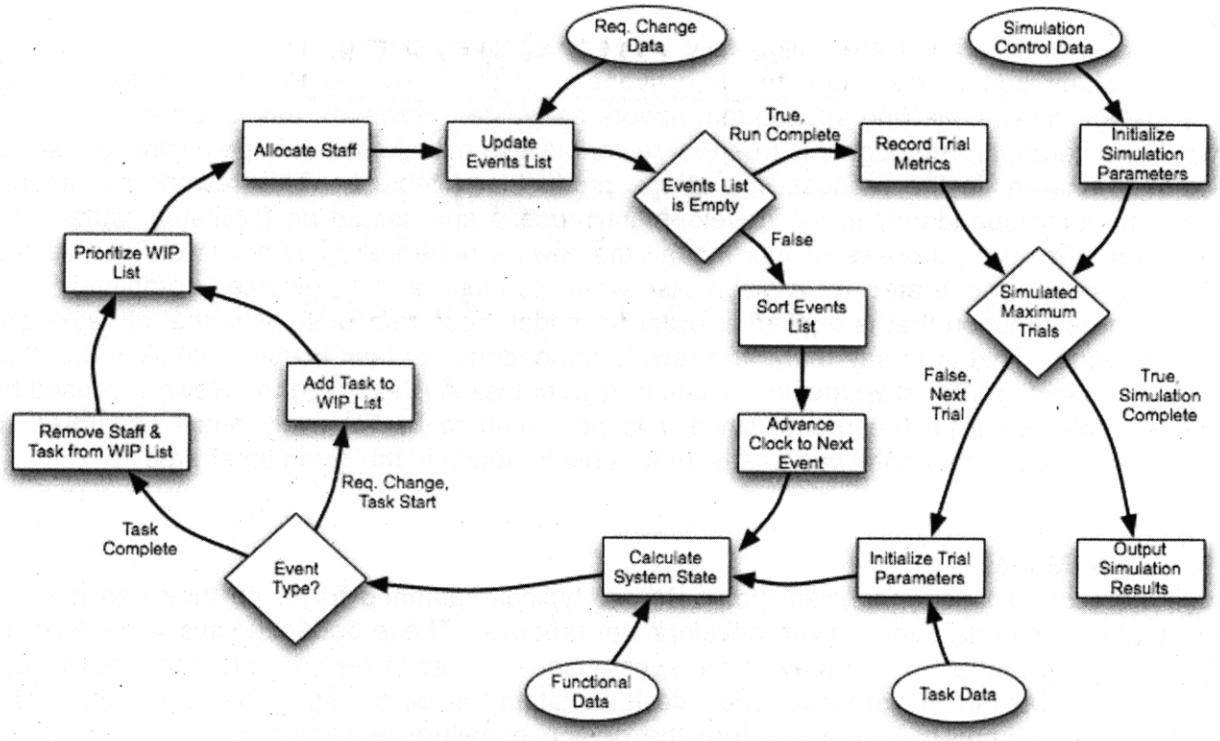


Figure 4.4 - High Level Model Logic

4.13 Input & Output

All of the input data for the model is aggregated into a single Microsoft Excel spreadsheet with workbooks for each type of data. The data quantifying dependencies between tasks was transposed into a list by an excel macro for easier input into the model. A scalar was given to each type of raw data. This scalar was used to calibrate data in aggregate against performance of actual development processes. Data concerning the characteristics of functional groups and human resource constraints is also present. Finally, time dependent requirements changes (or external engineering change orders) are compiled into a list for input to the model.

Several metrics are recorded for each simulated process trial and are formatted for output. The time to completion and the total process workload are recorded for each run. This data is plotted in histogram form to show the distribution of possible schedule and cost outcomes (workload is a proxy for cost since the wage rate data for each functional group was not made available). In addition, the staff levels over the duration of each process trial were recorded such that the distribution of staffing levels over time is also estimated.

4.14 Chapter Summary

The fourth chapter described in detail the manner in which the gathered data can be used to simulate the development process. The incorporation of characteristics of the process such as task learning curves and external considerations such as human resource constraints are described. Also more programmatic issues such as the treatment of uncertainty is discussed. Finally the simulation model's logic is described along with the format of input and output data. The following chapter offers analysis of the development process of propulsion products at Spirit using the qualitative binary DSM, various network analysis techniques, and finally the output from the simulation model.

5. PRODUCT DEVELOPMENT PROCESS ANALYSIS

5.1 Modeling Information Transfer

The Design Structure Matrix (DSM) shows that the development process is a network of interdependent tasks. The interdependencies cause iterative loops that allow the design to evolve to its final state. Planning tools that do not account for iteration and process dynamics become less relevant as the level of interdependency in the process increases. For example, it does not make sense to discuss the critical path of a complex development process that contains a high degree of interdependency. In a simple process where all information flows downstream, only tasks that fall on the critical path affect the total cycle time of the process. However, in a complex process, any task with interdependency could potentially affect the total cycle time. In order for the manager to understand where to focus attention in a complex process, evaluating the degree of process interdependency of each task should indicate the tasks that are most critical to the process network.

5.2 DSM Interpretation

Figure 5.1 below depicts the DSM for the development process of a jet nacelle. (See Appendix C for an enlargement of this figure.) The large grey bars divide the process into its three phases: concept development & project planning, detail design, and testing & certification. The preliminary design review (PDR) and critical design review (CDR) act as the stage-gate between development activities in the three phases.

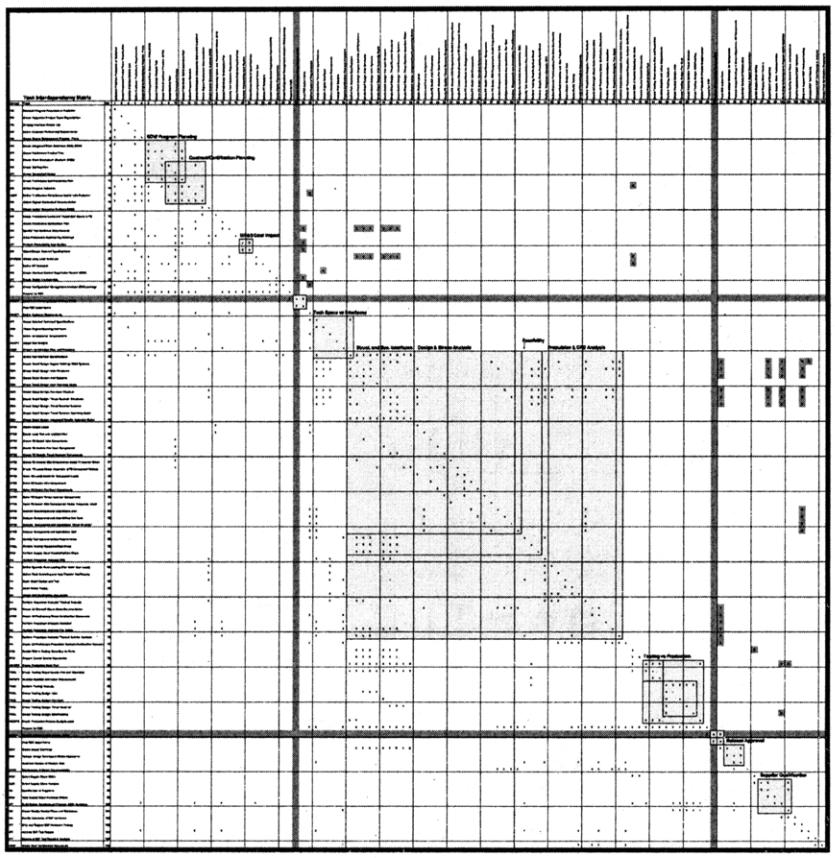


Figure 5.1 - Binary DSM for Development Process of a Nacelle

Although much of the information flow is downstream in the process (dependencies below the diagonal) there is significant information feedback (dependencies above the diagonal) which results in iterative loops. A box is drawn around each of these loops, and each is assigned a descriptive name. Some loops iterate independently, while others are coupled to other loops which causes task convergence to occur more slowly. Dependencies which occur above the diagonal and are not included within a planned iterative loop are deemed process failure modes. These dependencies are usually not likely to occur individually within the development process, and therefore are not included in typical iteration. However, collectively, the occurrence of one or several of these failure modes is likely and, therefore, should not be neglected. When interpreting the DSM, it should be noted that although the tasks are listed in order of which they first occur, the diagonal is not an effective proxy for the timeline of the project.

In the first phase of the development process, three iterative loops were identified as shown in Figure 5.2 below. The first loop, "Statement of Work/Program Planning", indicates that in order to create a plan for the development process, the scope and depth of work required by the customer must be understood. However, the work required by the customer may also depend upon the ability of the company to meet certain cost and schedule targets which is a product of planning. Thus these, tasks must iterate in order to converge to a final solution.

The second loop, "Contract/Certification Planning", indicates that the plan for the certification process depends upon the requirements agreed upon in the contract. However, the contracted requirements are influenced by the plan for certification in order to be realistic. Moreover, this iterative loop overlaps with the "SOW/Program Planning" loop, which could result in additional iteration of either loop.

The last loop in the first phase, "Materials, Processes, & Specifications (MP&S) Cost Impact", the cost of production depends upon developing and certifying new MP&S, but MP&S must also be constrained by the cost targets of production. These three iterative loops should not be neglected, but because they do not contain a high number of tasks or large workloads, they should not drastically impact the total development process outcome directly.

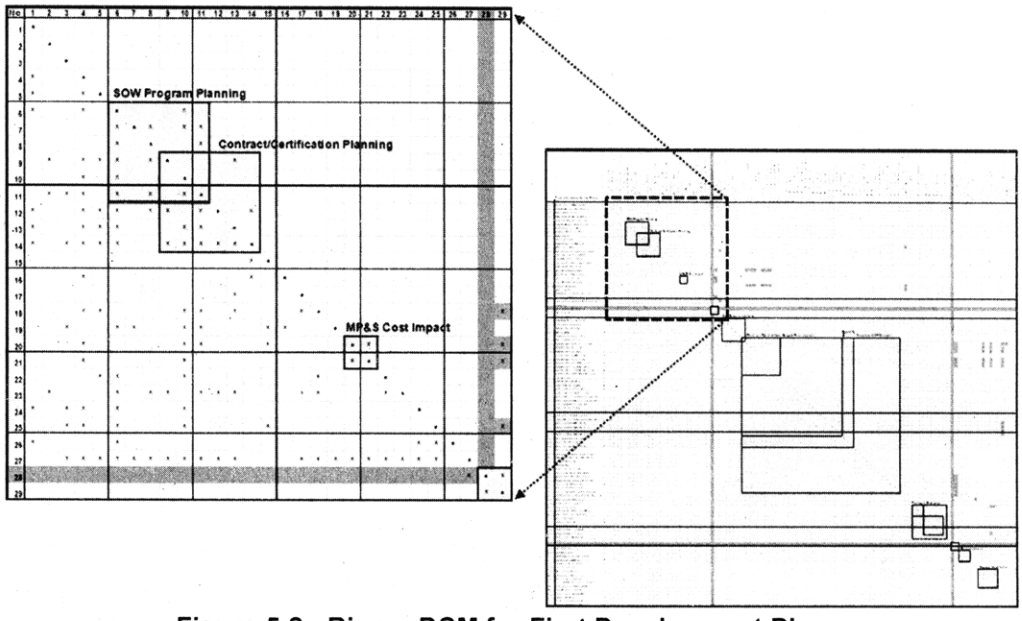


Figure 5.2 - Binary DSM for First Development Phase

In the second phase of the product development process, nine iterative loops were identified. Figure 5.5 below shows the first six. The first is simply iteration that occurs between preliminary design review and the action items that come out of the review in order to pass into the next phase. The second loop, "Technical Specifications & Interfaces," indicates that component specifications influence the component interfaces, and interfaces influence the specifications as well. These two loops are relatively minor.

The third loop, "Structures & Systems Interfaces," indicates that the design of the various subsystems influences the design of other subsystems through interfaces. Thus, no subsystem design can occur in isolation from the other subsystems. The dependency within this iterative loop is driven by the product architecture. Perhaps the most significant dependencies driving iteration result from the integration of systems and structures within a given subsystem. Better integration of these design activities may result in substantial improvements to the performance of the development process.

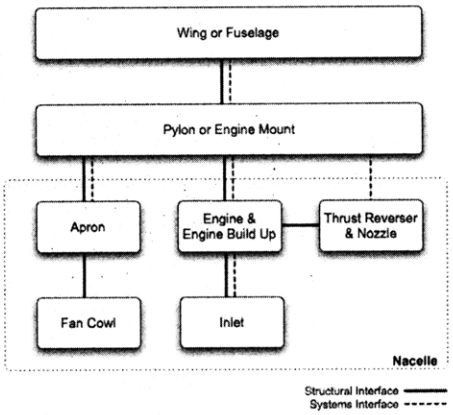


Figure 5.3 - Product Architecture

The fourth loop, "Design & Stress Analysis," indicates that the design of structures is subject to stress analysis in order to ensure the capability of a given design to withstand the specified forces required of it. The structures are designed, evaluated, and redesigned through an iterative design optimization process. The ultimate goal of this process is to minimize the weight of the product while meeting unit cost constraints and ensuring the strength of the design to safely withstand its operating environment. This iterative loop is typically viewed as the most costly since it involves the majority of development staff and contains a relatively high degree of rework dependency. Changing load requirements can manifest into very high costs and long delays since they can retrigger substantial iteration in this loop.

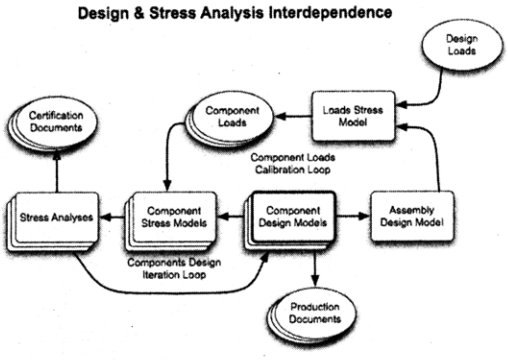


Figure 5.4 - Design-Stress Relation

The fifth loop, "Design Feasibility," indicates that the design of the subsystems must undergo evaluation to determine the producibility and the alignment of the design with the supply chain. This loop is essentially "design for manufacture and supply" in which manufacturing staff, tooling staff, and supply chain staff give early feedback regarding their perceptions of the design in order to minimize downstream problems in the future. Usually, this feedback is given in integrated product team (IPT) meetings, which occur throughout the development process rather than through formal documentation or formal reviews.

The sixth loop, "Design & Propulsion Analysis", indicates that the design must be analyzed for performance against various fluid and thermal related requirements. This analysis is usually performed at a system level rather than at the component level, so it does not display the complex dependency and does not require the significant staff levels of the "Design & Stress Analysis" loop. However, the tasks performed tend to be technically intensive which may drive significant human resource constraints in this loop.

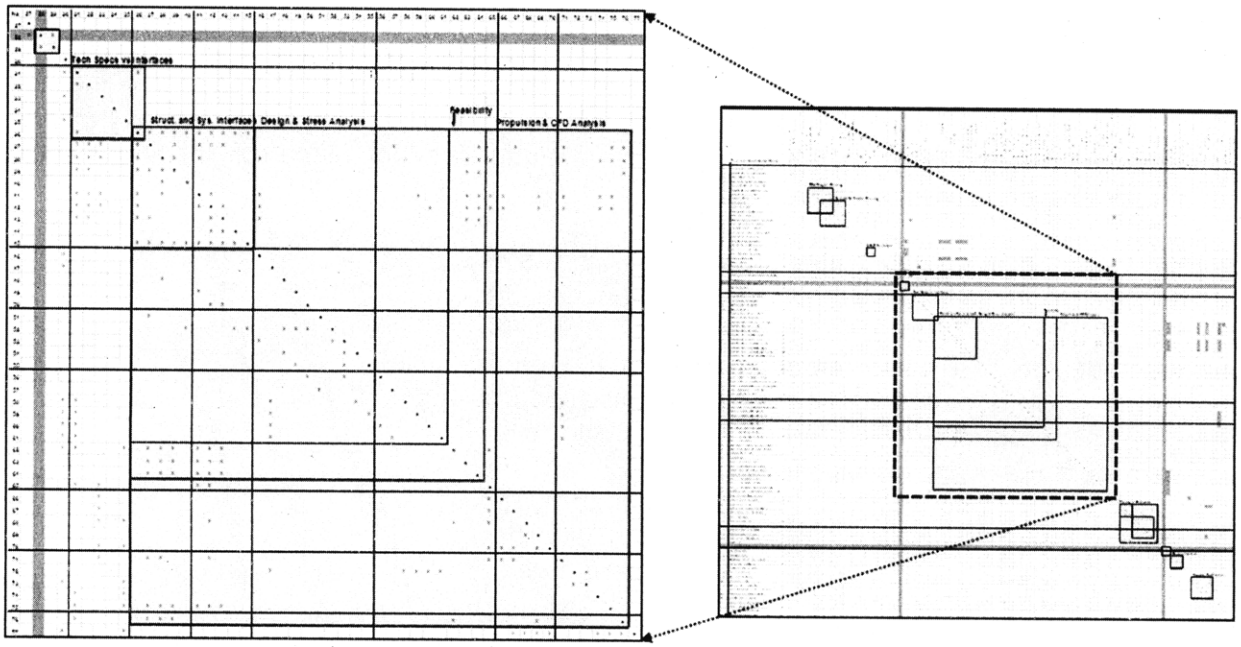


Figure 5.5 - Binary DSM for Second Development Phase

The primary iterative loops in the development process are those contained in the detail design phase. Specifically, there are four loops for design of structures which iterate mostly independently with minimal direct dependency upon each other, except through the structures design. Figure 5.6 to the right shows a simple representation of these loops. This complexity creates difficulty in managing the process since all four loops may occur simultaneously and the design may be evolve based on feedback from any one of them. Moreover, the design must be frozen for a significant portion of the analysis, which erodes the relevancy of the analysis as the design changes during that period. The overlapping of several iterative loops necessitates more iteration than in simpler processes.

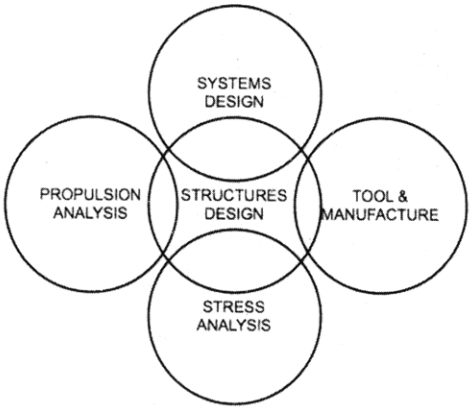


Figure 5.6 - Major Iteration Sources

The last three loops in the second phase of the development process facilitate the development of production tools and the manufacturing process. The first loop, "Production Requirements," indicates that the requirements for the tooling and the manufacturing process are dependent upon each other. The second loop, "Tool Design & Analysis," indicates that the tools themselves must undergo analysis to evaluate their ability to meet the production requirements. The final loop, "Tooling & Manufacturing Design" indicates that the manufacturing process design depends upon the tools, but also that the tooling design may depend upon the manufacturing process design. Although it is possible that the tooling design may impact the product design, the "Feasibility" loop is meant to provide tooling feedback, while isolating tooling design from product design in the overarching process.

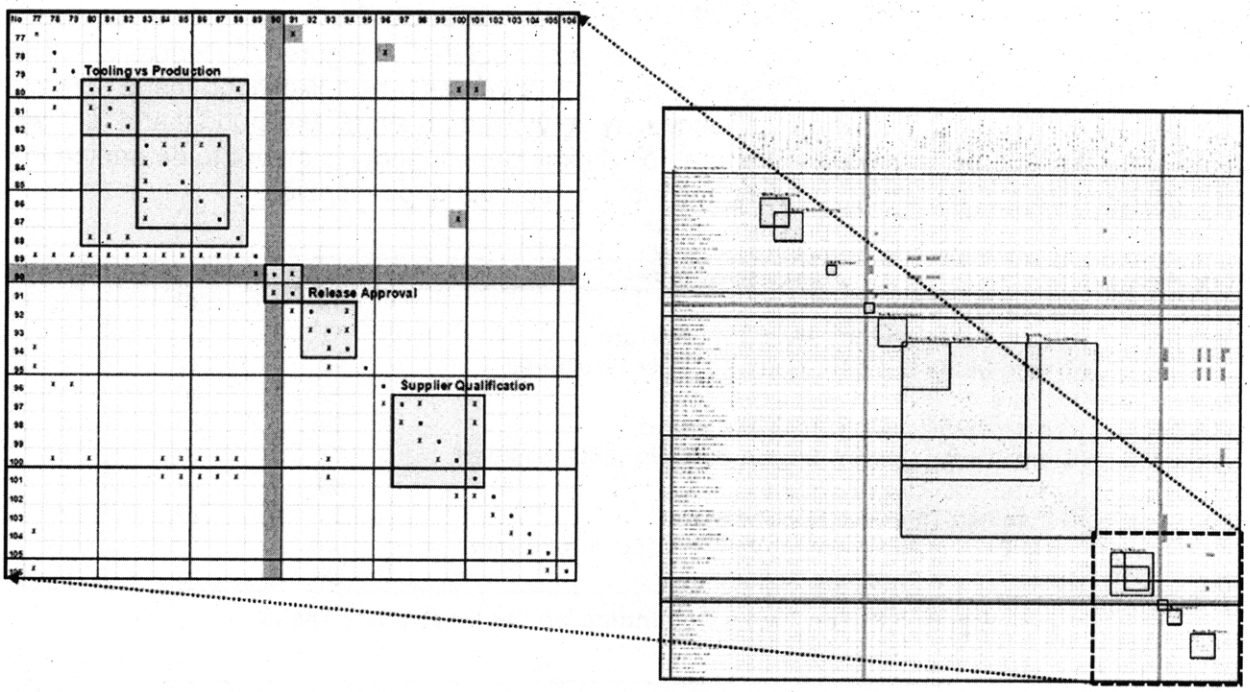


Figure 5.7 - Binary DSM for Third Development Phase

The third phase of development contains three iterative loops. The first is simply iteration that occurs between critical design review and the action items that come out of the review in order to pass into the next phase. The second loop, "Release Approval," indicates that the creation of the engineering drawings and the approval process for those drawings cause iteration. The final loop, "Supplier Qualification," indicates that the supplier selection process and the supplier approval process may iterate if chosen suppliers cannot meet the specified requirements.

Interestingly, the testing and certification of the product, which occurs in this phase, is not planned to cause design iteration. The "Design & Stress Analysis" loop and the "Design & Propulsion Analysis Loop" are meant to ensure positive test results with a high degree of certainty. However, the outcome is not always positive since there is always at least some small likelihood of failure. Therefore, the final development phase contains a significant number of failure modes which may retrigger previously completed process tasks.

5.3 Task Coupling Analysis

Coupling analysis is a simple tool that uses the information captured by the DSM to quantify interactions. The analysis can be performed at two levels - the individual task level, the lowest level of abstraction, and the group task level, which aggregates the individual task level data to create higher levels of abstraction. [9]

By observing the information flow and dependencies for each task, it is possible to identify roughly which tasks have the largest network effect in the development process. For complex development processes dominated by iterative loops, the performance of the process is typically most sensitive to tasks with the largest network effect. The ideal method for evaluating the network effect would be to perform a sensitivity analysis of the simulated development time for each design task and its parameters. But, simply adding a task's dependency data should yield approximate results. This measure is referred to as the "degree" of each task with respect to

the network of tasks. [13] The top results achieved by counting the input, output, and total dependencies from the binary DSM for each task are tabulated in Figure 5.8, Figure 5.9, and Figure 5.10 below. A high input dependency may be viewed as a high likelihood that a task will be reworked, while a high output dependency may be viewed as a high likelihood that a task will cause other tasks to be reworked. The total (or volume dependency) is meant to determine the total network effect of a task by summing the input and output dependencies.

Task Group	Task Description	Task No	Input Count	Rank
IPT	Prepare for CDR	89	38	1
DESIGN	Create Detail Design: Fan Cowl Structure	41	21	2
DESIGN	Create Detail Design: Thrust Reverser Structures	42	21	2
IPT	Prepare for PDR	27	20	4
DESIGN	Create Detail Design: Inlet Structures	38	20	4
DESIGN	Create Detail Design: Engine Build Up (EBU) Systems	37	19	6
ME/OPS	Create Production Build Plan	80	17	7
IPT	Define Part Interface Specifications	36	17	7
IPT	Build Engine Development Program (EDP) Hardware	100	14	9
PM	Identify Test Hardware Requirements	18	13	10

Figure 5.8 -Ten Tasks Depending on the Most Other Tasks

Task Group	Task Description	Task No	Output Count	Rank
DESIGN	Create Detail Design: Thrust Reverser Structures	42	23	1
DESIGN	Create Detail Design: Inlet Structures	38	21	2
IPT	Create Detailed Technical Specifications	31	20	3
IPT	Create Conceptual Design	10	20	3
DESIGN	Create Detail Design: Engine Build Up (EBU) Systems	37	20	3
DESIGN	Create Detail Design: Fan Cowl Structure	41	20	3
DESIGN	Create Detail Design: Thrust Reverser Systems	43	19	7
DESIGN	Create Detail Design: Inlet Systems	39	18	8
PM	Define Customer Preliminary Requirements	4	17	9
PM	Create Integrated Work Statement (IWS) (BOM)	6	17	9

Figure 5.9 - Ten Tasks Upon Which the Most Other Tasks Depend

Task Group	Task Description	Task No	Volume Count	Rank
DESIGN	Create Detail Design: Thrust Reverser Structures	42	44	1
DESIGN	Create Detail Design: Fan Cowl Structure	41	41	2
DESIGN	Create Detail Design: Inlet Structures	38	41	2
IPT	Prepare for CDR	89	39	4
DESIGN	Create Detail Design: Engine Build Up (EBU) Systems	37	39	4
DESIGN	Create Detail Design: Thrust Reverser Systems	43	31	6
DESIGN	Create Detail Design: Inlet Systems	39	30	7
IPT	Define Part Interface Specifications	36	28	8
IPT	Build Engine Development Program (EDP) Hardware	100	23	9
IPT	Create Detailed Technical Specifications	31	22	10
IPT	Create Conceptual Design	10	22	10

Figure 5.10 - Ten Tasks that Interface with the Most Other Tasks

These results show that the design tasks interface with the most other tasks and are therefore most likely to influence the performance of the development process. These results can be further refined by multiplying each counted dependency by its rework factor. [13] The top results achieved by summing the input, output, and total rework dependencies from the numeric DSM for each task are tabulated in Figure 5.11, Figure 5.12, and Figure 5.13 below.

Task Group	Task Description	Task No	Input Measure	Input Percent	Rank
DESIGN	Create Detail Design: Inlet Structures	38	4.2	2.6%	1
DESIGN	Create Detail Design: Thrust Reverser Structures	42	4.1	2.6%	2
PM	Obtain Signed Contractual Documentation	14	4	2.5%	3
DESIGN	Create Detail Design: Fan Cowl Structure	41	3.9	2.4%	4
PM	Identify Test Hardware Requirements	18	3.9	2.4%	4
IPT	Define Part Interface Specifications	36	3.6	2.3%	6
DESIGN	Create Detail Design: Engine Build Up (EBU) Systems	37	3.4	2.1%	7
IPT	Define IPT Schedule	23	3.2	2.0%	8
IPT	Perform Producibility Cost Studies	20	3.1	1.9%	9
PM	Define Program Schedule	12	2.9	1.8%	10

Figure 5.11 - Ten Tasks Depending the Most on Other Tasks

Task Group	Task Description	Task No	Output Measure	Output Percent	Rank
IPT	Create Conceptual Design	10	8.9	5.6%	1
PM	Create Integrated Work Statement (IWS) (BOM)	6	6.1	3.8%	2
DESIGN	Create Detail Design: Thrust Reverser Structures	42	5.6	3.5%	3
PM	Define Customer Preliminary Requirements	4	5.4	3.4%	4
DESIGN	Create Detail Design: Engine Build Up (EBU) Systems	37	5.1	3.2%	5
DESIGN	Create Detail Design: Inlet Structures	38	5.1	3.2%	5
DESIGN	Create Detail Design: Fan Cowl Structure	41	4.7	2.9%	7
IPT	Define Customer Requirements	30	4.15	2.6%	8
PM	Create Engine Development Program Plans	5	3.8	2.4%	9
DESIGN	Create Detail Design: Thrust Reverser Systems	43	3.8	2.4%	9

Figure 5.12 - Ten Tasks Upon Which Other Tasks Depend the Most

Task Group	Task Description	Task No	Volume Measure	Volume Percent	Rank
DESIGN	Create Detail Design: Thrust Reverser Structures	42	9.70	3.0%	1
IPT	Create Conceptual Design	10	9.70	3.0%	1
DESIGN	Create Detail Design: Inlet Structures	38	9.30	2.9%	3
DESIGN	Create Detail Design: Fan Cowl Structure	41	8.60	2.7%	4
DESIGN	Create Detail Design: Engine Build Up (EBU) Systems	37	8.50	2.7%	5
PM	Create Integrated Work Statement (IWS) (BOM)	6	6.70	2.1%	6
CERT	Define Certification Compliance Matrix with Customer	13	6.30	2.0%	7
DESIGN	Create Detail Design: Thrust Reverser Systems	43	6.00	1.9%	8
PM	Define Customer Preliminary Requirements	4	5.50	1.7%	9
IPT	Define Customer Requirements	30	5.45	1.7%	10

Figure 5.13 - Ten Tasks that Exhibit the Most Input and Output Dependency

These results confirm that the design tasks are the most likely to influence the process performance. However, several new tasks are brought forward that the binary DSM was not able to capture. For example, creating an appropriate conceptual design becomes the second most important in terms of causing additional iteration as shown in Figure 5.13. Also, defining the certification compliance matrix with the customer becomes a critical task that must be managed well. Finally, defining preliminary and final requirements can play a critical role in the process. This is especially important since customer requirements evolve over time, which as shown here, can drastically impact the development process. Weighting each task by its typical staff level or its typical completion time could further refine the coupling analysis for future work.

A more advanced method for evaluating the network impact of each task is to measure the distance (e.g. number of tasks) that separates each task from all of the other tasks. This measure is referred to as the distance modularity of each task and is given for the number of tasks (n) by the formula below. [13]

$$M(\mathrm{IT})_i = \frac{\text{Actual distance disconnectivity}}{\text{Maximum distance disconnectivity}} = \frac{\sum_{j=1, j\neq i}^{n} d(i,j)}{n(n-1)}$$

This measure can validate the accuracy of the "degree" measure, since immediate tasks must also be connected to other tasks in order to have a significant impact. [13] The MATLAB code used to calculate the distance of each task is shown in Appendix B. The results from analyzing the DSM for "distance" are depicted in Figure 5.14 below. The top 37 tasks are shown in Figure 5.14 below. These tasks have very similar network distances that are significantly shorter than the others.

Task Group	Task Description	Task No	Input Distance	Output Distance	Total Distance
DESIGN	Create Detail Design: Thrust Reverser Structures	42	42%	35%	38%
DESIGN	Create Detail Design: Fan Cowl Structure	41	42%	35%	38%
DESIGN	Create Detail Design: Inlet Structures	38	42%	35%	38%
IPT	Define Part Interface Specifications	36	42%	35%	38%
DESIGN	Create Detail Design: Engine Build Up (EBU) Systems	37	42%	35%	38%
DESIGN	Create Detail Design: Thrust Reverser Systems	43	42%	35%	39%
DESIGN	Create Detail Design: Inlet Systems	39	42%	35%	39%
PA	Perform Propulsion Analysis: Fire Safety	75	42%	35%	39%
IPT	Create Detailed Technical Specifications	31	43%	35%	39%
PA	Perform Propulsion Analysis: Thermal Analysis	71	42%	35%	39%
PA	Perform Propulsion Analysis: CFD	65	43%	35%	39%
PA	Perform Propulsion Analysis: Thermal Anti-Ice Analyses	76	42%	35%	39%
TOOL	Perform Tooling Feasibility/Cost Check (DFM)	63	42%	35%	39%
DESIGN	Create Detail Design: Integrated Nacelle Assembly Model	45	42%	35%	39%
SCM	Perform Supply Chain Feasibility/Cost Check	64	42%	35%	39%
STRESS	Analyze Components and Assemblies: Thrust Reverser	60	43%	35%	39%
STRESS	Analyze Components and Assemblies: Fan Cowl	59	43%	35%	39%
STRESS	Analyze Components and Assemblies: Inlet	58	43%	35%	39%
DESIGN	Identify Test Instrumentation Requirements	62	42%	35%	39%
DESIGN	Create Detail Design: Thrust Reverser Assembly Model	44	43%	35%	39%
DESIGN	Create Detail Design: Inlet Assembly Model	40	43%	35%	39%
PA	Define Dynamic Fluid Loading (Fan Outer Duct Loads)	66	43%	35%	39%
PA	Define Fluid Velocities and Heat Transfer Coefficients	67	43%	35%	39%
PA	Perform Propulsion Analysis: Acoustics	74	43%	35%	39%
PA	Scale Model Testing	69	43%	35%	39%
PA	Scale Model Design	68	42%	36%	39%
DESIGN	Obtain Engine Interfaces	32	43%	35%	39%
STRESS	Solve FE Model: EBU Components: Modal Frequency check	57	43%	35%	39%
STRESS	Create FE Models: EBU Components: Modal Frequency Check	51	43%	36%	39%
STRESS	Solve FE Model: Thrust Reverser Components	56	43%	36%	39%
STRESS	Solve FE Model: Fan Cowl Components	55	43%	36%	39%
STRESS	Create FE Models: Thrust Reverser Components	50	43%	36%	39%
STRESS	Solve FE Model: Inlet Components	54	43%	36%	39%
STRESS	Create FE Models: Fan Cowl Components	49	43%	36%	39%
STRESS	Create FE Model: Inlet Components	48	43%	36%	39%
STRESS	Solve FE Loads Model for Component Loads	53	43%	36%	39%
STRESS	Create FE Loads Model (Assembly of FE Component Models)	52	43%	37%	40%

Figure 5.14 - Most Integrated Tasks by Network Distance

This analysis did not take into account the dependency weightings, since it is only meant to validate the findings of the coupling analysis based on task degree. The distance measure of tasks confirms that the Design tasks are central to the development network. In addition, the Stress and Propulsions Analysis (PA) tasks are also shown to be very central to development of aerostructures.

5.4 Functional Group Analysis

Individual task data can be aggregated by functional group in order to show functional dependency. Using rework dependencies from the numeric DSM, information flow between and within functional groups can be assessed. The charts in Figure 5.15 to the right approximate the input, output, and total dependency that occurs across functional lines. The integrated product teams (IPT's) interact the most with other functional groups. This is reassuring since the IPT is not actually a functional group, but rather a group consisting of members of the various functional groups. The fact that the IPT as a group interacts with the most other functional groups indicates that the tasks that require the most functional integration are the responsibility of the IPT. Thus, IPT's seem to have the authority necessary to fulfill the charter for which they are intended. Again, the design group shows up as having a high degree of interdependency with the other groups. The stress group should also be acknowledged as needing to communicate heavily with other groups since it is heavily involved in both the design and certification processes. Finally, program management exhibits a high degree of cross-functional dependency, which should be expected.

Group	Input Measure	Input Percent	Rank
IPT	38.3	24%	1
DESIGN	28.7	18%	2
STRESS	25.8	16%	3
PM	22.6	14%	4
PA	15.3	10%	5
SCM	8.8	6%	6
TOOL	7.8	5%	7
ME/OPS	7.5	5%	8
CERT	4.7	3%	9

Group	Output Measure	Output Percent	Rank
IPT	36.1	23%	1
PM	35.2	22%	2
DESIGN	35.1	22%	3
STRESS	17.7	11%	4
PA	13.9	9%	5
ME/OPS	5.9	4%	6
CERT	5.7	4%	7
TOOL	5.2	3%	8
SCM	4.8	3%	9

Group	Volume Measure	Volume Percent	Rank
IPT	74.4	23%	1
DESIGN	63.8	20%	2
PM	57.8	18%	3
STRESS	43.5	14%	4
PA	29.2	9%	5
SCM	13.6	4%	6
ME/OPS	13.4	4%	7
TOOL	13.0	4%	8
CERT	10.4	3%	9

Figure 5.15 - Functional Info Exchange

The DSM in Figure 5.16 below shows the percent of total process dependencies of each group upon the others. The numbers the show up along the diagonal indicate intra-functional dependency, while those off of the diagonal indicate inter-functional dependency. Although the largest dependencies do occur within a group, which is to be expected if grouping is effective, intra-functional dependency accounts for only 43% of the total process dependency as represented by the DSM. This assessment of intra-functional dependency should be taken in context of the level of detail and aggregation of tasks within the DSM. Thus, intra-functional dependency accounts for 43% of process dependency, only at a relatively high level.

	PM	IPT	DESIGN	STRESS	PA	TOOL	ME/OPS	SCM	CERT	
PM	9%	3%	1%	0%	0%	0%	0%	0%	1%	14%
IPT	9%	10%	2%	0%	0%	1%	1%	1%	1%	24%
DESIGN	1%	4%	7%	2%	3%	0%	1%	0%	0%	18%
STRESS	0%	3%	3%	8%	1%	0%	0%	0%	0%	16%
PA	1%	1%	2%	0%	5%	0%	0%	0%	1%	10%
TOOL	0%	0%	3%	0%	0%	1%	0%	0%	0%	5%
ME/OPS	0%	1%	1%	0%	0%	1%	1%	0%	0%	5%
SCM	1%	0%	3%	0%	0%	0%	1%	2%	0%	6%
CERT	1%	1%	0%	0%	0%	0%	0%	0%	0%	3%
	22%	23%	22%	11%	9%	3%	4%	3%	4%	100%

Figure 5.16 - Functional Group Interdependence

5.5 Nominal Completion Time, Cost & Staff Levels

Please note that the data and graphical information provided in this section has been scaled to represent data and risk typical in industry, rather than that of the host company. Therefore, this data may not be used to evaluate the host company, and may not be associated with it in any way. It may only be used educationally for the purposes of comprehending a method for understanding, modeling and improving product development processes.

The output from the DSM-based simulation model shown in this section show that typical variability in the process tasks and variability in the process failure modes can cause non-negligible variation in the cost and completion time of the baseline process. For example, the histogram in Figure 5.17 below depicts the likelihood of completing the development process as a function of time. Feasible completion times range from 500 to 700 business days, a range of 200 business days. Also, there seem to be at least two modes. The first represents the possibility of having minimal or no failure modes occurring during the process, while the second represents the possibility that failure modes create substantial rework. Although each of the failure modes individually may not be likely, the impact of the failure modes collectively seems to be significant.

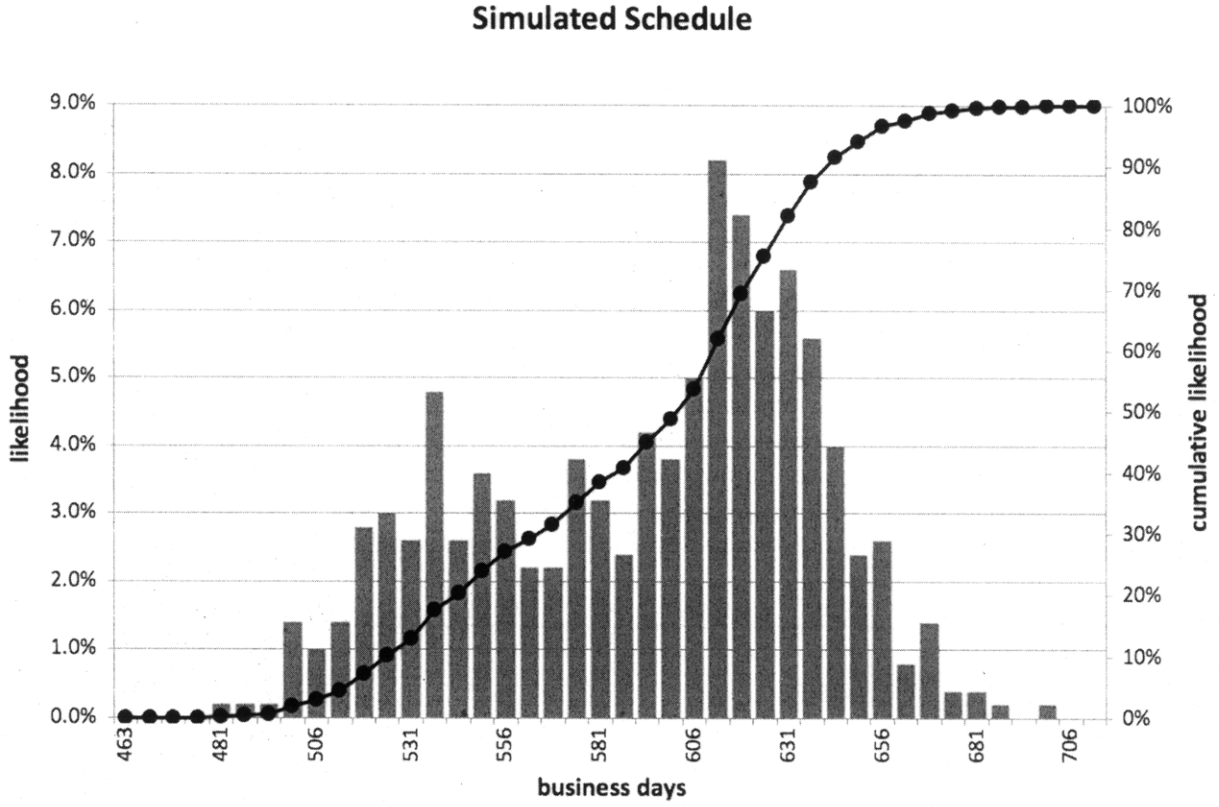


Figure 5.17 - Simulated Nominal Completion Time Distribution

The predicted total workload required, which is a proxy for cost, is shown in the histogram in Figure 5.18 below. At first glance, the total workload variability seems less significant that that of completion time. However, the figure below actually depicts the workload required by the process. One assumption of the model is that human resources are provided as needed by the process. This assumption is somewhat invalid in reality since human resources cannot be

easily removed from a development process. Thus when idle time is added to the workload, the distribution will be much more correlated with the time of completion of the project.

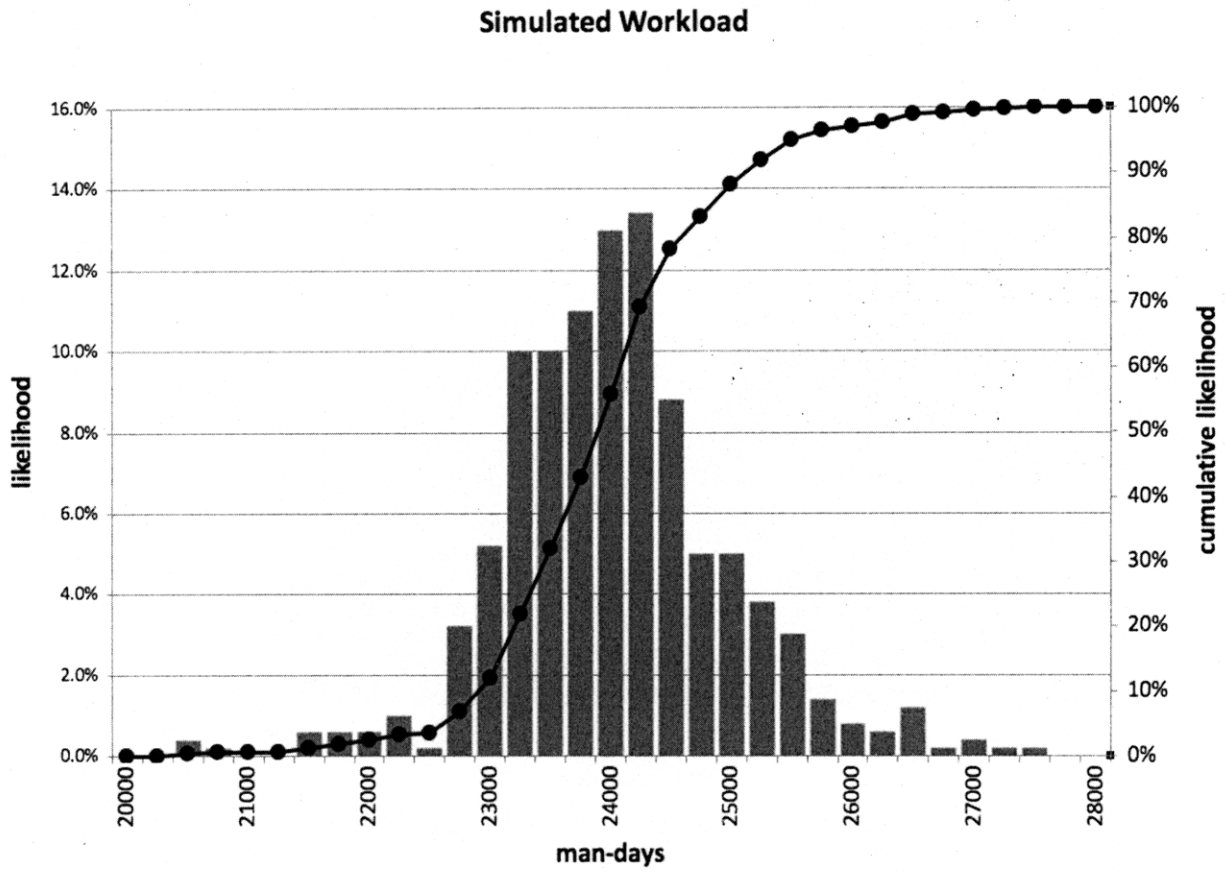


Figure 5.18 - Simulated Nominal Required Workload Distribution

The staffing profile of the simulated baseline process is shown in Figure 5.19 below along with the likelihood of process completion. This can be interpreted as the expected number of staff required by the process over the course of the process. The mean plus one and two standard deviations are also shown, in order to roughly depict variance. Towards the end of the process, as the possibility of completing the process becomes more substantial, the staff level percentiles become less relevant, as the staff levels become less adequately represented by the normal distribution. This figure shows that initially, there is little variation in the staff required. Thus the first iteration of design work and analysis can be easily planned for. However, as various iterative loops begin and as failure modes begin to occur, early predictions of staffing requirements become less relevant. This shows that the staffing of a program at certain points in time may have a standard deviation as high as 10 people. So a flexible human resource and planning structure could have major benefits in reducing the cost of a program. There also seem to be several peaks throughout the program that coincide with typical design iteration and with the occurrence of failure modes that occur near the end of the program. Constraining the human resources to a point below that demanded by the actual process would have the effect of lowering the peaks and extending the timeline. This may, however, increase the timeline more than just by the area under the curve since iterative loops may be substantially slowed.

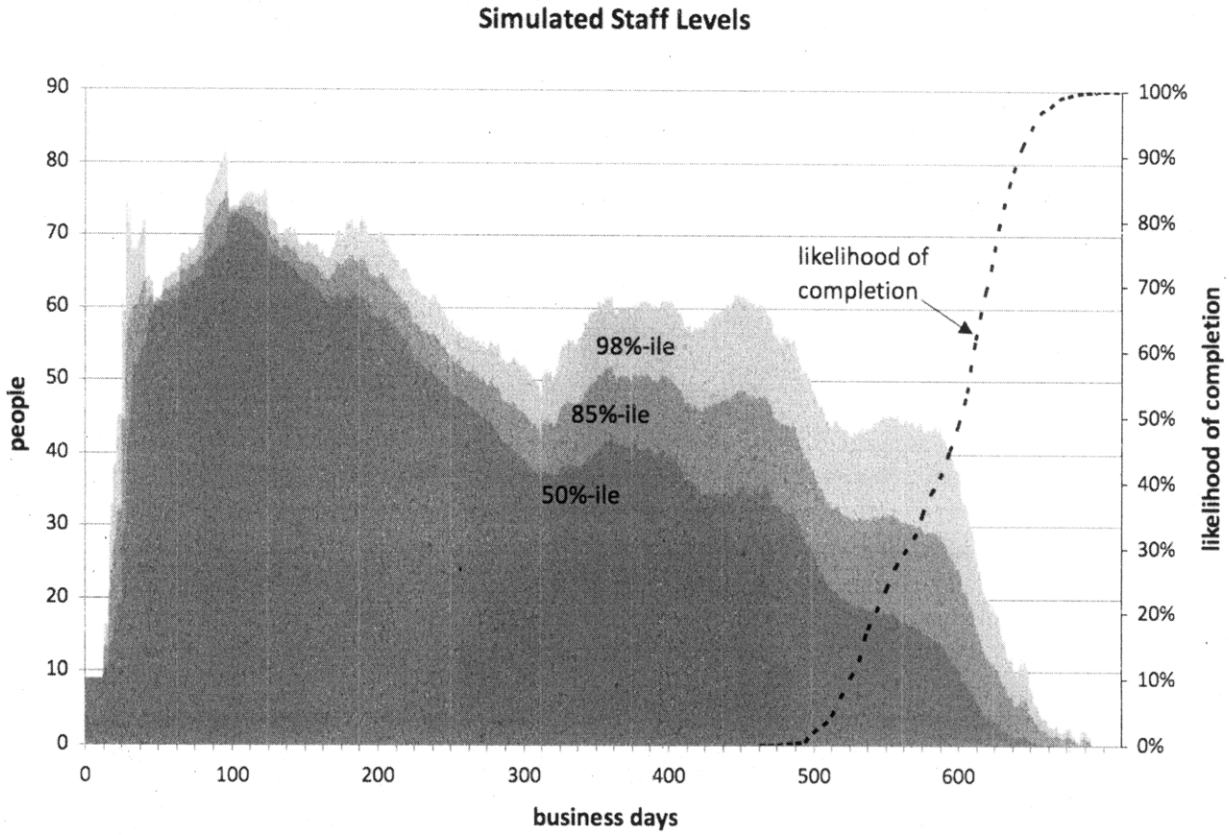


Figure 5.19 - Simulated Nominal Staff Levels Required by the Development Process

The simulation results depicted in the above figures show that the baseline development process without the impact of external drivers exhibits a substantial degree of variation, which is inherent in the process tasks and dependencies. There may be ways to improve process tasks, improve the speed of iteration, and reduce the risk associated with the process that could drastically impact development process performance.

5.6 Chapter Summary

The fifth chapter analyzed the development process for propulsion products at Spirit using a variety of techniques. A detailed description of the process dynamics is given by discussing each iterative loop in the process. Control and facilitation of these dynamic loops will dictate the performance of the process. Next network coupling analysis tools are used to identify the specific tasks that are likely most crucial to the performance of the process. A similar network analysis gives insight to how the various functional group depend on each other. Finally, the output from the simulation tool described in the fourth chapter is presented and analyzed. The analysis in this chapter is based solely upon the process characteristics. The following chapter will offer analysis of the development process incorporating externally changing requirements.

6. MANAGING DEVELOPMENT WITH CHANGING REQUIREMENTS

6.1 Source of Changing Requirements

More and more frequently, product development responsibilities are being pushed down the supply chain in many industries. This disintegration of the design chain for a product may have many advantages including reduced cost structure and better design for manufacture. Specific to the large aircraft industry, the ability to create additional jobs in various countries may secure aircraft sales to those countries. One significant disadvantage in disintegrating the design chain of a highly interdependent product is that informal communication, which is necessary to efficiently complete the development process, is replaced by formal communication across the various participating organizations. Specifically, iteration that occurs across organizations is manifest through changing customer requirements. Moreover, if adequate high level design work and analysis has not been preformed by the OEM or system integrator before offloading system and sub-system design responsibilities to suppliers, the suppliers' requirements will change even more substantially. In a disintegrated design chain for an interdependent product, customer requirements will always change in order to align and integrate the development efforts of all the organizations involved.

6.2 Types of Change

The most disruptive factor of the product development process is changing customer requirements. Throughout the course of a development program, it is understood that the requirements will change, but the timing of these changes is difficult to predict, and the impact of the changes is difficult to quantify, even upon completion of the development effort. In addition, changing requirements can create a sense of frustration on the part of the process participants and a sense of panic on the part of the process managers. Creating a product development process that is more robust to changing requirements will enable creation of more accurate proposals and schedules, staffing programs at reduced cost, and more consistently meeting the deadlines of the customer.

The effect of changing load scenarios seems to be the largest cause of total rework for Spirit during product development. As program staffing is ramped up, design and analysis iteration begins, thus advancing the product design against customer specified loads and other requirements. Once the design begins to reach maturity, however, the customer (aircraft OEM) "always" changes the load scenarios. The changing load scenarios render much of the previous work and design iteration invalid, in effect creating substantial amounts of rework. The primary cause of these customer driven loads changes seems to simply be advancement of the analysis of the overall aircraft design. The fact that the requirements are developed concurrently with the detail design effort rather than in advance of it inevitably leads to changing requirements.

In an ideal PD process, customer requirements would be finalized before development starts. However, the increased need to shorten development time of aircraft has driven a higher degree of concurrency. Aircraft OEM's now develop the overarching aircraft design simultaneously with its subsystems. This high degree of concurrency results in the release of immature requirements to design suppliers as a basis for early development. As development and analysis of the aircraft and its subsystems progress, the OEM is able to release more mature requirements to its design suppliers. As such, updated requirements are received at Spirit, rendering some of the previous work invalid and creating great amounts of rework.

With the understanding that customer requirements do not change randomly, but as a result of the maturing of the overall aircraft design and analysis, these requirements may be qualified

with a maturity rating. This rating should then be used as a basis for the amount of work that should be exerted for a given set of requirements. In this manner, a program can be scheduled and staffed such that no more work, additional detailing or analysis is performed than is justified by the maturity of the requirements.

6.3 Time Dependency

The two major factors dictating the impact of a requirements change are the timing of the change with respect to the state of the process and the extent of the change. If a requirements change occurs early within the development process, the impact is not substantial since little work had been initiated at that point. However, if the requirements change occurs later in the process, the impact is greater since the change renders a large amount of previous work invalid.

To demonstrate the significance of the timing of a requirements change using the simulation model, a single change was issued that completely changes the load requirements that the structure must withstand. When this change is made at the start of the process, no difference is perceived from simulating the processes without the change. But when the change is made after the process has completed, a very substantial amount of rework is required causing major cost and schedule overruns as shown in the Figure 6.1 below. The impact shown should be viewed as the maximum potential impact since the change occurs at a perceived finalized state of the process. The impact from a change in requirements will be reduced at earlier states of the development process.

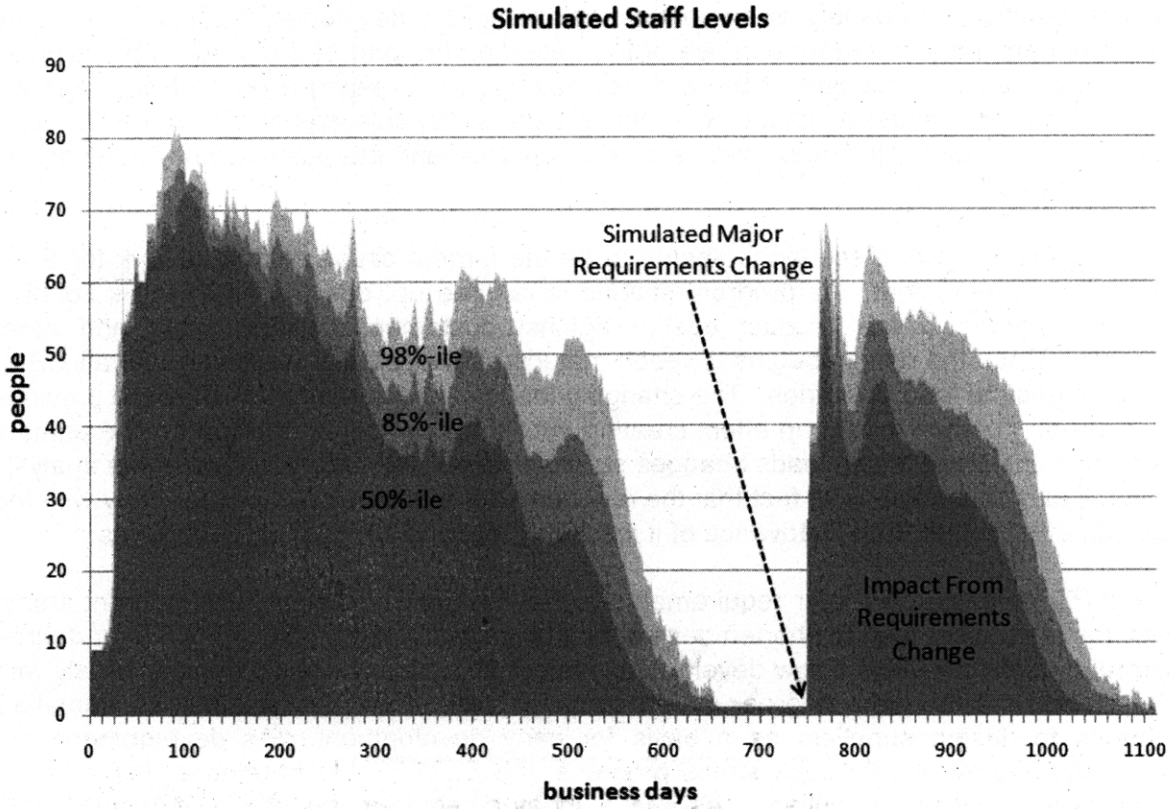


Figure 6.1 - Maximum Simulated Impact of a Single Major Requirements Change

6.4 Externality Adjusted Completion Time, Cost & Staff Levels

Please note that the data and graphical information provided in this section has been scaled to represent data and risk typical in industry, rather than that of the host company. Therefore, this data may not be used to evaluate the host company, and may not be associated with it in any way. It may only be used educationally for the purposes of comprehending a method for understanding, modeling and improving product development processes.

Changing customer requirements can drastically change the cost and schedule of a development process, as shown in the previous section. Therefore, it is necessary to modify the baseline process results from the previous chapter to reflect the change that occurs in reality. A set of 24 requirements changes, shown in Figure 6.2 below, was compiled to reflect the timing and extent of changes that may occur during a typical development process. Changes were induced for requirements regarding surface geometries, external interfaces, aerodynamic performance, load scenarios, and configuration. The extent of change was generally reduced as the development process advanced. Variability of the timing of incoming changes was not considered, although this would likely result in increased variability in cost and schedule. All 24 changes were induced for each run of the Monte Carlo simulation, which allows us to understand the potential effects of a set of deterministic, exogenous changes.

Index	Task	Task Description	Time (hrs)	Impact
1	15	Obtain Master Dimension Surfaces (MDS)	1000	50%
2	15	Obtain Master Dimension Surfaces (MDS)	2000	25%
3	16	Obtain Preliminary Loads and Transmittal Sheets (LTS)	1000	50%
4	16	Obtain Preliminary Loads and Transmittal Sheets (LTS)	2000	25%
5	16	Obtain Preliminary Loads and Transmittal Sheets (LTS)	4000	25%
6	16	Obtain Preliminary Loads and Transmittal Sheets (LTS)	6000	25%
7	30	Define Customer Requirements	1000	50%
8	32	Obtain Engine Interfaces	1000	50%
9	32	Obtain Engine Interfaces	3000	25%
10	32	Obtain Engine Interfaces	5000	25%
11	32	Obtain Engine Interfaces	6000	25%
12	32	Obtain Engine Interfaces	7000	25%
13	33	Define Aerodynamic Requirements	1000	50%
14	33	Define Aerodynamic Requirements	2000	25%
15	33	Define Aerodynamic Requirements	3000	25%
16	33	Define Aerodynamic Requirements	5000	25%
17	33	Define Aerodynamic Requirements	7000	25%
18	46	Obtain Design Loads	1000	75%
19	46	Obtain Design Loads	2000	50%
20	46	Obtain Design Loads	3000	50%
21	46	Obtain Design Loads	4000	40%
22	46	Obtain Design Loads	5000	30%
23	46	Obtain Design Loads	6000	20%
24	46	Obtain Design Loads	7000	20%

Figure 6.2 – List of Simulated Requirements Changes

Figure 6.3 below shows a histogram of the distribution of the time required to complete development activities. The average duration has increased by almost a factor of two due to external changes. Interestingly, the variation in completion time is actually reduced. This is due to the fact that the total time required is implicitly derived from the timing of the last requirement change, which entails final set of requirements. Since the final change was fixed in this model, the variation of completion time was reduced. However, uncertainty or variation in the timing of the final requirements change would cause additional variation in completion time.

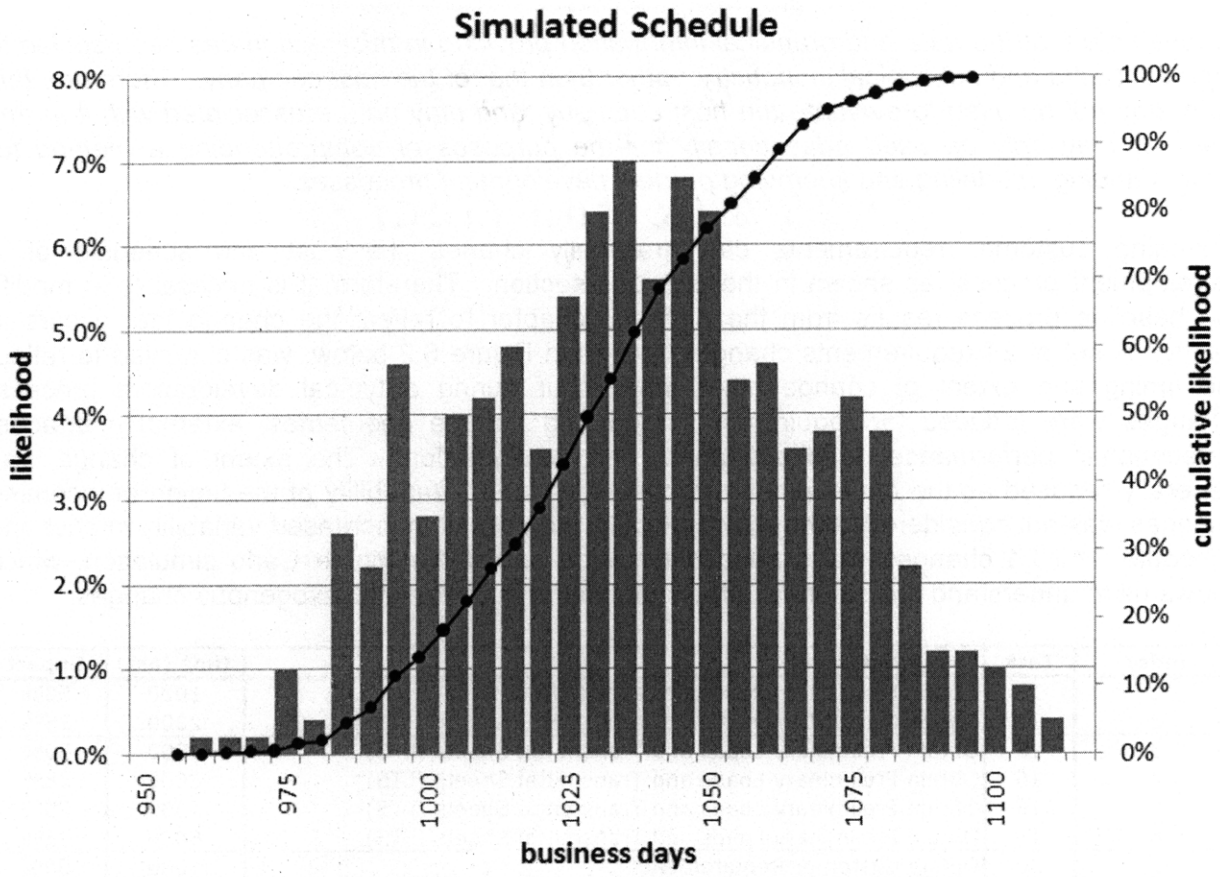


Figure 6.3 - Simulated Completion Time Distribution with Typical External Change

Figure 6.4 below, depicts the histogram of the distribution of work in man-days required by the process. Workload, like schedule, has also increased by almost a factor of two. More importantly, the 80% range of potential workloads has almost tripled indicating increased variation. This is important because if human resources are inflexible, then the actual worked hours on a program will mask opportunities for cost reduction. But if human resources are easily placed and removed from the development process, this may result in lower cost.

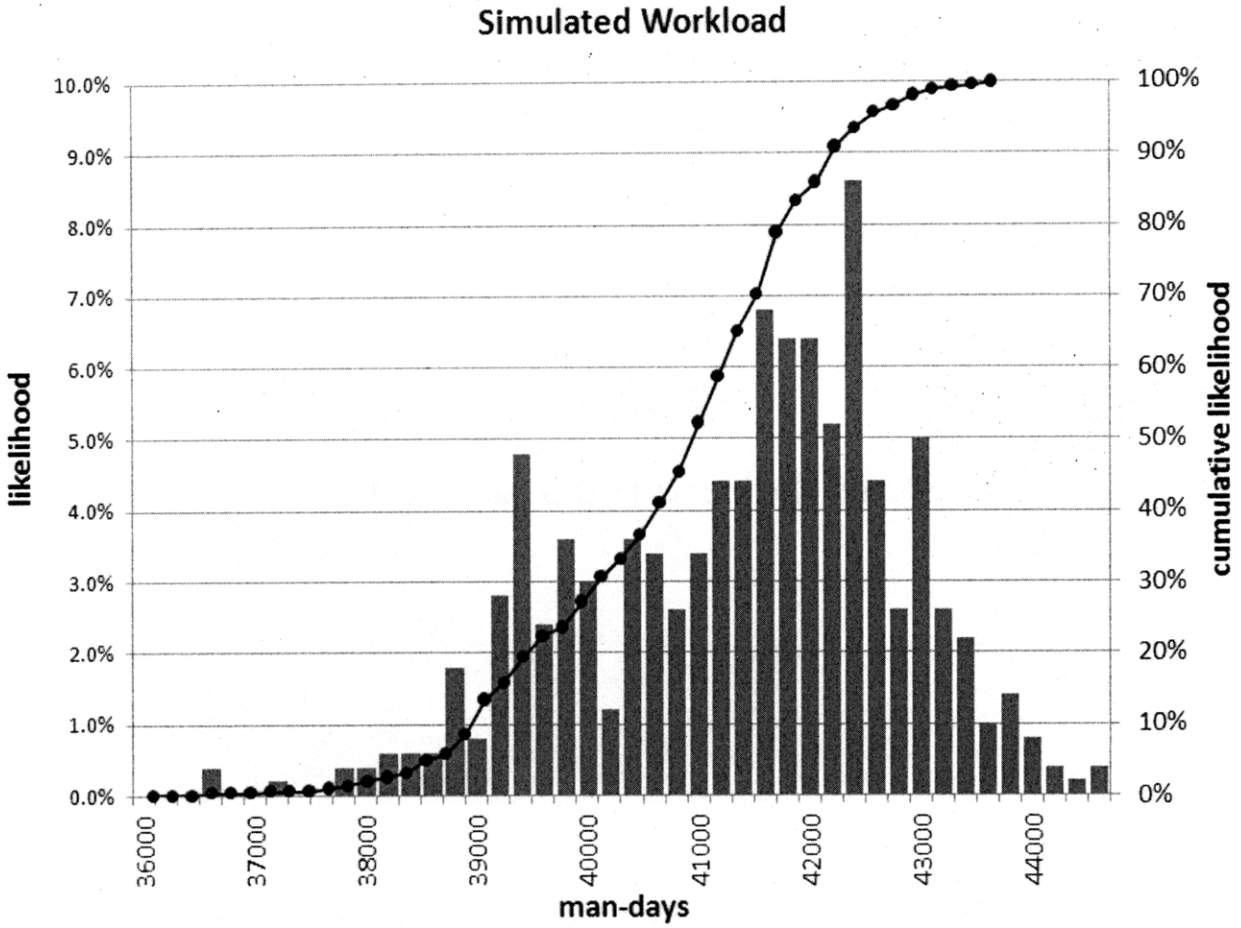


Figure 6.4 – Simulated Required Workload Distribution with Typical External Change

Finally, Figure 6.5 below depicts the staffing requirements demanded by the development process over time. The changes incurred drive staff peaks later in the process, whereas early changes are not as disruptive. Intuitively, the process cannot converge until the last requirements change has taken place. Again, flexible human resources would be able to take advantage of the dips in staffing demand. Maintaining peak staff levels throughout would allow the process to meet the same completion time, but with a higher cost. Maintaining valley levels throughout would allow the process to minimize cost, but at the expense of the schedule. The white line indicates the expected baseline staff levels without incorporating change. This graph shows with certainty that requirements changes may have a drastic impact on the development process and pose a unique and difficult management challenge.

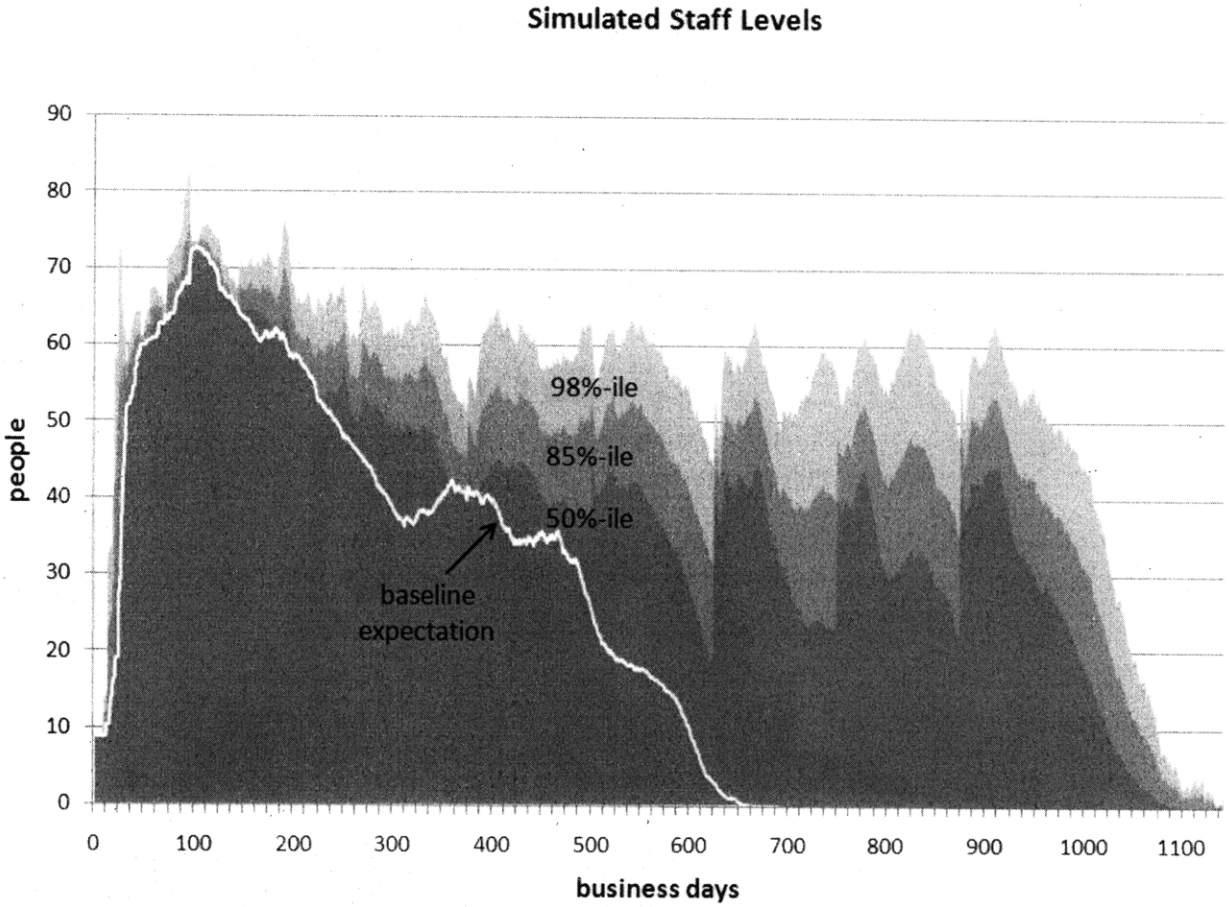


Figure 6.5 - Simulated Staff Level Requirements with Typical External Change

6.5 Wasted Development Effort

Although much iteration required to advance the development of Spirit's products is inherent in the products, a significant amount of iteration and rework stems from changing customer requirements – most typically changing loads. At first glance, it may seem as though the wasted work in the product development process lies in the vast amounts of rework created by changing requirements. Many industries judge the effectiveness of their PD processes by "first pass" design quality, viewing subsequent rework as waste. However, when rework is caused by changing customer requirements rather than engineering mistake, the rework cannot be seen as waste since it advances the design against the final set of requirements which were previously unknown. In effect, the real waste lies in the work that was initially completed, much of which may have been irrelevant with respect to the final customer requirements.

Consider the simple staffing profile in Figure 6.6 below. In this process there are no iterative loops, just a ramp up, a steady state and a ramp down of staff. The time denoted by t_C denotes the time at which the detail design phase is complete. Figure 6.7 indicates that development progress for the simple staffing profile follows an S-shaped curve, which is simply the accumulation of the area (man-hours) under the staffing profile as time proceeds.

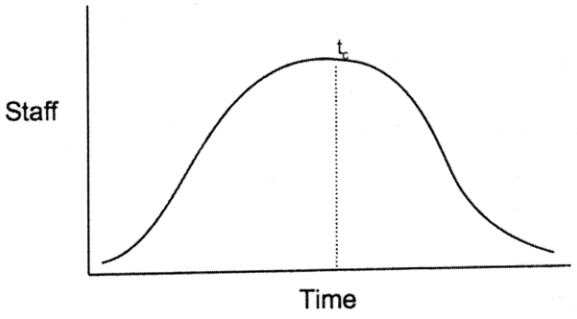


Figure 6.6 - Simple Staff Profile

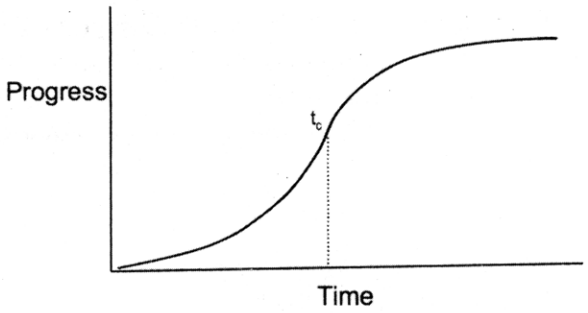


Figure 6.7 - Simple Process Progress

Now consider how changing customer requirements three times during the development process might prolong the completion of detail design. The design cannot be complete until the requirements are finalized. The staffing profile has now been extended such that is does not ramp down until after the final set of requirements is received, as shown in Figure 6.8. The perceived progress against the current set of requirements is drastically reduced each time the requirements change, as shown in Figure 6.9. It is often perceived that the overrun due to the changing requirements is waste, since it could have been avoided if the final requirements were simply known at the beginning of the development process.

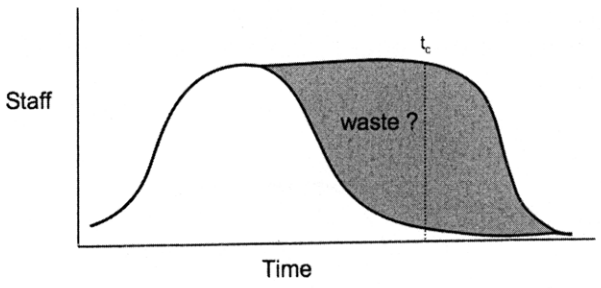


Figure 6.8 - Staff Profile with Change

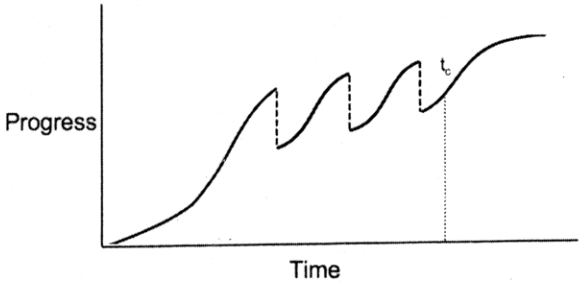


Figure 6.9 - Progress with Change

However, it is usually impossible for the final requirements to be known at the beginning of the process. Also, it is not appropriate to qualify work that advances the design against the final requirements as waste since that work may actually be the most valuable. For separate organizations developing interdependent parts of a design concurrently, it may be helpful to understand the evolution of customer requirements in terms of their maturity, as shown in Figure 6.10 below.

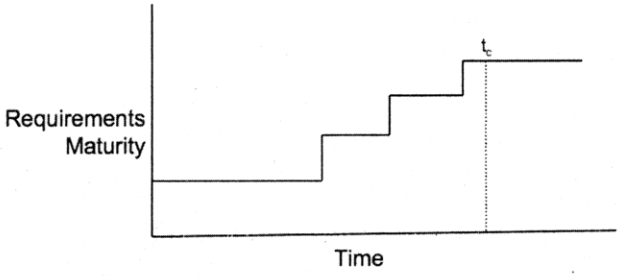


Figure 6.10 - Requirements Maturity with Change

Through this lens, the design process advances the design only as much as is justified by the maturity of the final requirements, or the predicted relevancy of the current requirements to the anticipated final requirements. This paradigm views waste as the work that was done in excess of what is justified by the maturity of the requirements as shown in Figure 6.11 and Figure 6.12.

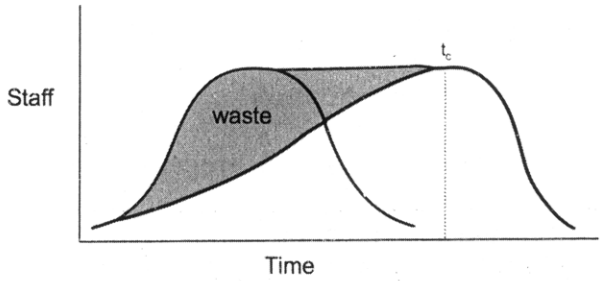


Figure 6.11 - Potentially Wasted Effort

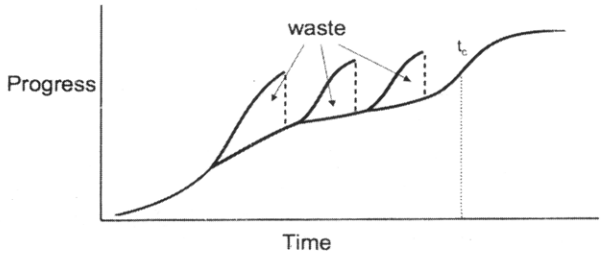


Figure 6.12 - Potentially Wasted Progress

There is one important caveat to this perspective. The extent to which the maturity of the requirements depends upon the maturity of the design dictates the ability of the process to staff based upon the requirements maturity. If the advancement of the maturity of the requirements depends upon the maturity of the design, then excess progress of the design could reduce the likelihood of future requirements changes. In this case, excess staffing may be justified. Figure 6.13 shows the case in which requirements maturity is completely dependent on the design maturity, thus rendering necessary an early staff ramp up. Figure 6.14 depicts how the optimal staffing ramp up depends on the degree of dependency of the requirements maturity upon the design maturity.

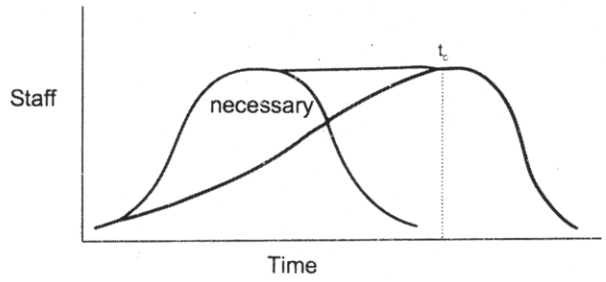


Figure 6.13 - Potential Necessity of High Staffing

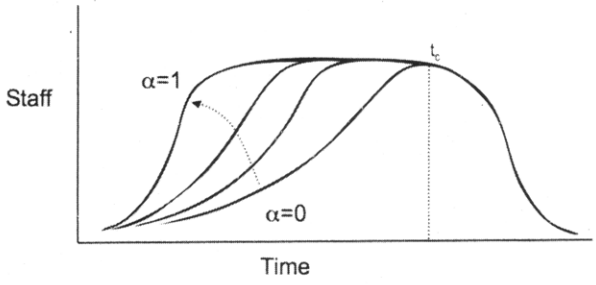


Figure 6.14 - Optimal Staffing Levels

Effort is made in chapter 9, "Restructuring and Improving the Development Process," to develop quantitative measures for evaluating the maturity of requirements and incentivizing open communication regarding requirements maturity between suppliers and the OEM.

6.6 Creating a Robust Development Process

Change is a reality of product development in the aerospace industry. Because the overall aircraft design and each of its interdependent subsystems are developed both separately and concurrently, change cannot be eliminated. Therefore, the most suited process for product development is not one of speed under ideal or planned circumstances, but one of robustness to change. In the current process at Spirit, change is somewhat disruptive. Psychologically, change may create panic for management and frustration for engineers. A development process that is robust to change means that change is expected and planned for, and that excess effort toward meeting immature requirements is minimized. The goal of a robust development structure is to ensure that change typical on new programs does not disrupt the

process and does not result in overruns that may have been avoided. Recommendations for creating a robust development process are reserved for chapter 9, "Restructuring and Improving the Development Process."

6.7 Chapter Summary

The sixth chapter discussed and analyzed the impact of changing customer on the performance of the development process. The two factors dictating the impact of a change are the timing and the extent of the change. A set of requirements changes meant replicate changes that may actually occur on a program was introduced to the simulation model. The results showed that changing requirements is likely the largest factor in determining the performance of the development process. Finally, a staffing policy to optimize the cost of development is offered based upon the maturity level of requirements and the dependency of requirements maturity upon the maturity of the product design. With a more substantial understanding of the development process and its sensitivities based upon the analysis of this and the previous chapters, a method for managing risk will be offered in the following chapter.

7. MANAGING RISK

7.1 FMEA Methodology Overview

The Failure Mode & Effects Analysis (FMEA) is a procedure for documenting and analyzing potential failure modes within a system for classification by severity or determination of the effect of failures on the system. Typically, two types of methodologies exist, the product and the process FMEA. The product FMEA is often used to improve reliability and safety of products being developed. The process FMEA is typically used to improve the reliability of manufacturing processes. However, Browning notes that the task-based DSM is well suited to process FMEA. [1] The FMEA can be applied to each of the failure modes of the DSM in order better understand development process risk. The FMEA documentation may also be used to prioritize and organize risk management activities. This process typically should be undertaken by a team of four to eight people. Too few people will not provide enough accuracy in capturing and quantifying failure effects, while too many people results in ineffective meetings.

The first step in conducting an FMEA is to identify the failure modes of the process. Although the DSM already provides this information, the team should familiarize itself with these failure modes and ensure that the list is complete. Using the DSM structure, each failure mode should be identified with a dependency between two tasks, an input task and an output task. A brief summary is also useful to better describe what the failure actually is.

A failure effect is defined as the result of a failure mode on the whole development process. Separate effects should be listed separately under the same failure mode. A rating should be given to each effect, quantifying the predicted impact on the cost and schedule of the development process. It is important that the FMEA team establish the meaning of ratings before proceeding to ensure consistency throughout the process. Example ratings for severity are:

0.0 – negligible impact on product development cost and schedule
0.2 – some impact on the product development cost or schedule, but not likely to be perceived by the customer/OEM
0.5 – large impact on the product development cost or schedule, which will likely require some re-planning or re-negotiation with the customer/OEM
1.0 – critical impact on the product development cost or schedule, which may put the future of the program or the company at risk

To determine the overall severity of a failure mode's effects, the severity of each effect should not be simply added individually since the scale is not necessarily linear (i.e. five effects with severity of two should not aggregate to a severity of rating of ten). Instead the highest severity rating should be used unless a more appropriate formula is agreed upon.

A failure cause is defined as the event or source which initiates the occurrence of a failure mode. Independent causes should be listed separately under the same failure mode. An occurrence rating should be assigned to each failure cause. Occurrence rating is defined as the likelihood of the given failure cause occurring during the development process. Example ratings for occurrence are:

0.0 – zero possibility of occurring
0.5 – fifty percent likelihood of occurring
1.0 – definite likelihood of occurring

The overall occurrence from a failure mode's causes is given by one minus the likelihood that no failure causes occur, as shown in the equation below.

$$O_{Total} = 1 - \prod_{1}^{n}(1 - O_n)$$

Typically in an FMEA, controls should be taken into account in assessing the ability of the system to detect impending failure and to preemptively mitigate its consequences. However in human based systems such as a product development process, such controls are difficult to automate and are unreliable. Therefore, controls created in a product development process are incorporated only through reduction of occurrence or severity and the detectability rating is not considered.

The total risk associated for any single failure mode is obtained by multiplying the severity of each failure mode by its occurrence. This risk priority number may be used to evaluate the total level of risk in the development process (e.g. number of failure modes surpassing certain thresholds), to prioritize risk management efforts, and to determine the expected impact of risk which can be extrapolated from running the simulation model with and without the presence of process failure modes.

The FMEA should be documented, preferably in spreadsheet form such as that of MS Excel. This will allow for easy communication of results and easy prioritization of risk reduction and mitigation initiatives. A DSM-based process FMEA should reference the process dependency to which each failure mode refers. Below is a list of headings which should be present:

- Index – an identifying number for each failure mode
- Output Task – the task which may trigger a failure mode causing unplanned rework
- Input Task – the task which may be reworked if the failure mode occurs
- Failure Mode Description – a brief summary describing circumstances of the failure mode
- Effect(s) of Failure Mode – a potential consequence of the occurrence of a failure mode
- Severity – the impact of the consequence of the failure mode
- Cause(s) of Failure Mode – a potential cause which will induce the failure mode
- Occurrence – the likelihood of the occurrence of the cause of a failure mode
- Risk Priority Number – a rating for the total risk associated with a failure mode
- Recommended Actions – a list of actions which would reduce the occurrence or severity associated with a failure mode
- Responsibility - the person or group responsible for undertaking the recommended action

There are several common practices in prioritizing failure modes. One practice is to adopt a color-based grouping approach. This approach determines threshold risk priority numbers which classify the failure modes into green, yellow, and red groups. Red failure modes are considered extremely risky, while green ones are considered reasonable. Yellow failure modes are somewhere in between. A major benefit from this approach is that the total significant risk of a program may be easily communicated as the number of red, yellow, and green failure modes for the process. Also, the red classification may be psychologically motivating for group members. There are criticisms of this approach, however. The first is that threshold limits may be determined somewhat arbitrarily, which may not set well in a more quantitative environment, where numbers are typically not used subjectively. Another is that fixed threshold limits are not appropriate since the severity and occurrence data for a failure mode may be developed relationally against the other failure modes. So, the data may be scaled high or low since it was developed subjectively, in which case fixed thresholds may not portray actual levels of risk.

Another common practice is maintaining a top ten list of failure modes to address. This approach may be beneficial since it is very conducive to continuous improvement, regardless of the state or risk. However, if the total level of risk is very high, more effort may be justified to address additional failure modes. If the total level of risk is low, the reduction in risk from the efforts addressing the top ten failure modes may not justify the costs.

A final, and potentially most appropriate, common practice is for a manager to evaluate the best, and most cost effective way to reduce risk. The risk associated with some failure modes may be more easily reduced than others. This approach should account for the manager's integration of tacit knowledge and cost-benefit analysis.

7.2 Reducing Risk and Mitigating Failure Mode Effects

There are many ways to reduce risk and mitigate its effects. Simply increasing the awareness of specific risks alone can make a substantial impact in reducing risk. This is because many task performers are unaware of the system risk, so once they become aware they will naturally act to reduce it. Other, more proactive methods are described below.

One method is to freeze the design even though iteration may have naturally occurred, and to label the failure as generational learning. For example, if it is found that a specific design feature is costly to manufacture, rather than reiterating the design, this rework may be artificially withheld and the feedback will be withheld for future products. This approach is best suited when the failure mode is for a non-critical aspect of the design (e.g. not safety related) and when iteration is very costly (e.g. if the failure mode occurs once the design is mostly complete).

Another technique is to rapidly advance the beginning iterations of each of the development tasks in order to trigger failure modes as early as possible. This has the effect of reducing the severity of the failure mode since it would not impact the cost or schedule to the extent that it would if the process advanced more slowly. This technique is best suited for processes with substantial iteration, since the rapid advancement may cost more relatively in processes that only iterate once or twice. This may have the effect of increasing the expected cost or schedule while reducing the risk of substantial overruns.

Another approach is to restructure the process to reduce the risk or to eliminate the failure mode completely. An example of this approach would be to incorporate prototype testing into the standard development process, rather than using it as a final quality check on the design for certification. If testing is incorporated into the process earlier, the risk of testing failure would be drastically reduced, although the expected time and cost to complete the process may be extended.

Finally, some companies simply assign each failure mode to a team. Thus, that team is responsible for reducing the likelihood and severity of their assigned failure modes. These teams are given flexibility to address their failures in the ways that they deem most effective. This is similar to generating awareness, but it takes it a step further by assigning responsibility and incentives. The most mature organizations maintain a failure reporting and corrective action database in which occurrence of actual process failures and effects are recorded, and corrective actions are created upon further analysis.

7.3 Chapter Summary

The seventh chapter overviewed the failure mode and effects analysis in the context of failure modes identified through the process DSM. Suggested methods were offered to reduce risk and mitigate failure mode effects. Unfortunately, schedule constraints did not allow for sufficient time during the internship for the author to perform an FMEA that would be adequate for presentation here. Before suggesting recommendations for improving the development process, a more holistic view of aircraft development is offered in the following chapter.

8. PRODUCT DEVELOPMENT INTEGRATION IN THE AIRCRAFT INDUSTRY

8.1 Increased Outsourced Development

Recently, OEM's in the large aircraft industry have felt increasing pressure to disintegrate their business and to increase outsourcing. Over the past decade, this has lead to increased outsourcing of both product supply and development. Two primary forces have driven this disintegration – cost reduction and offsets. Pressure to reduces costs forces companies to develop suppliers in lower wage environments. "Offsets" is the term given to the agreement between the aircraft OEM's and the governments of potential aircraft customers to develop or build part of an aircraft in that country in exchange for government subsidies for aircraft purchases from that OEM. Recently, Boeing adopted a mostly outsourced model for development and supply of its 787 Dreamliner.

The Dreamliner is now scheduled to deliver two years later than planned - the culprit, in large part, is the new unprecedented outsource model that Boeing had adopted for the program. [18] Diving headfirst into this new model, Boeing outsourced roughly 70% of Dreamliner parts which would have previously been built in-house. The new model not only increased externally supplied content of the aircraft, but also offloaded a substantial amount of design work to those suppliers. Although the impact on the development schedule is obvious, it has yet to be seen how the new model will play out once full-scale production is underway.

Although many argue that Boeing's lack of control over its dispersed supply chain led to the problems it is currently experiencing, [7] the underlying problem may be much more structural in nature – that Boeing disintegrated the design effort of a highly interdependent product, failing to understand the impact of this forced modularization upon the development process.

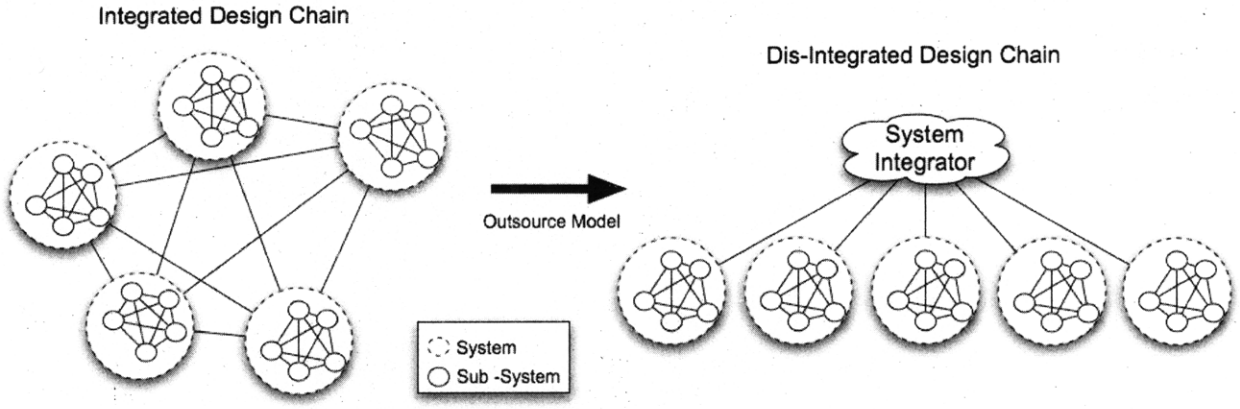


Figure 8.1 - Effect of Outsourcing on Design Chain Architecture

8.2 Interdependent Design & Process Integration

When designing a highly complex product, it is critical to understand the interdependency between different parts of the product. Typically, parts which are highly inter-dependent can be grouped into sub-systems, and sub-systems which are highly interdependent can be grouped into systems. Systems represent the highest level of grouping of the whole product. The point of grouping parts into systems and sub-systems is that interdependent designs must collaborate in order to reach a feasible solution. For this reason, there must be a certain level of integration between parts, sub-systems, and systems which depends on the degree of interdependency between these groups. As seen in Figure 8.1 above, vertical integration provides an architecture with open collaborative channels. Under an outsourced model, collaboration

between interdependent sub-systems takes place through the system integrator and is inherently a model with more resistive collaborative channels. Figure 8.2 below depicts grouping responsibilities of design tasks along the lines of simplified systems and sub-systems.

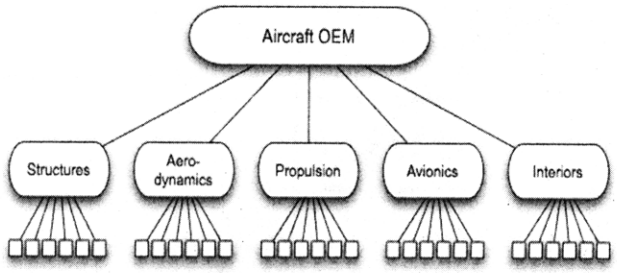


Figure 8.2 - Example Systems Integrator Design Chain

8.3 Sections & Systems Architecture

The problems resulting from the disintegration of the design chain at the system level are further complicated by the grouping of design tasks based on sections rather than on systems. The sections grouping breaks the structures system into parts, as they should be supplied. The production and assembly of an aircraft is not an iterative process, so supply of aircraft parts may be dissected based on labor and logistics costs. Although sections grouping optimizes the supply of parts assemblies, superimposing this structure onto the development process drastically complicates the process architecture. Not only is collaboration between the highly interdependent sections indirectly routed through the OEM, but other systems must now collaborate with multiple structures providers in alleviating inter-systemic issues. Figure 8.3 and Figure 8.4 below depict the sections view of an aircraft and the associated sections-based product DSM. [16]

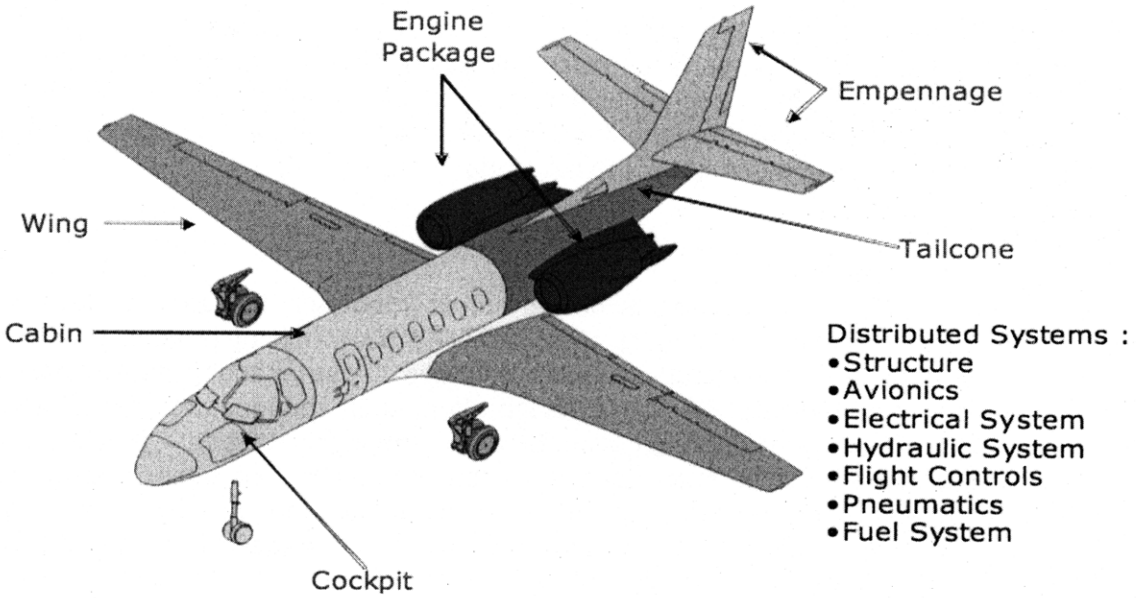


Figure 8.3 - Example Sections Architecture of a Business Jet

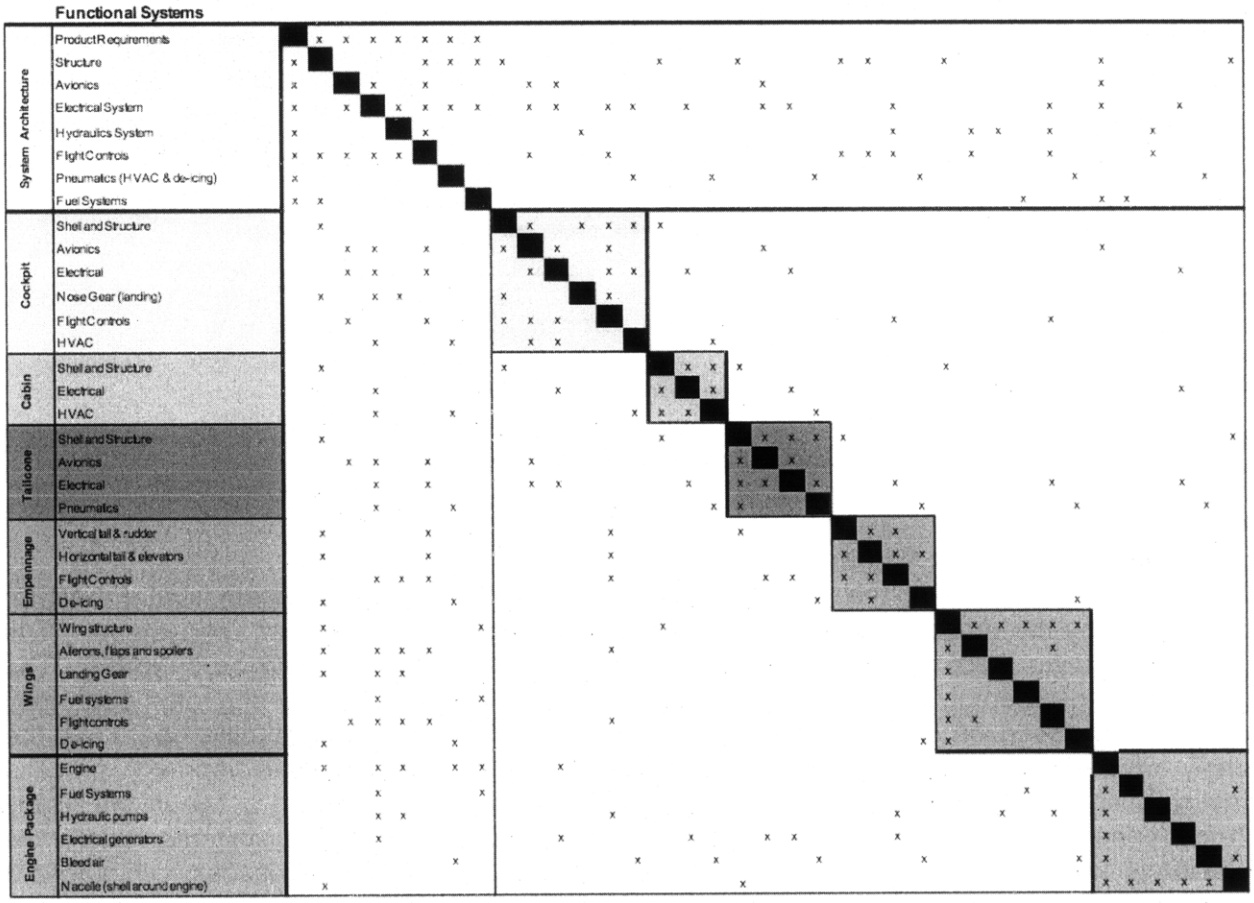


Figure 8.4 - Binary Product DSM for Example Sections Aircraft Architecture

This product DSM shows that the sections architecture drastically increases the number of inter-organizational dependencies and interfaces versus grouping based on functional systems. Tripathy further notes that modular decomposition for jet aircraft is not possible since the functional systems are distributed throughout the aircraft, touching almost all sections, and since each section contains elements of most of the functional systems. [16]

8.4 Mutating Requirements

If the role in the development process is now systems integrator, the aircraft OEM is now responsible for communicating the impact of an evolving over-arching product design to its suppliers. As the design of a particular supplier changes, those changes may impact other parts of the design. In a vertically integrated design effort, such changes are internal and may occur quickly and informally through iteration between the design tasks. However, in an outsourced model, this sort of iteration does not occur informally, but rather through formal requirements changes. The formality of this process is compounded by other business processes that must also occur such as re-evaluation of contracts for scope creep and estimation of the cost impact of changing requirements. Thus, mutating requirements and engineering change orders replace informal collaboration, drastically impeding the design evolution process. Figure 8.5 below shows the degree of interdependency in a sections-based design chain, which may act as sources for changing requirements.

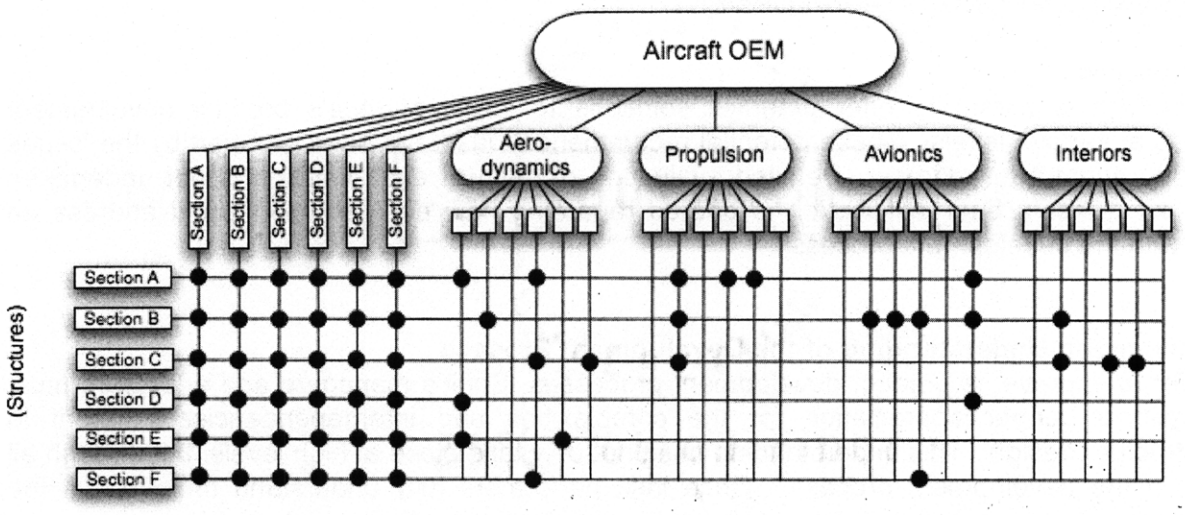


Figure 8.5 - Effect of Sections Architecture on Systems Integrator Design Chain

8.5 Cost of Concurrency

Many believed that the new outsourced model would allow for an expedient development schedule, since suppliers would be responsible for their own designs allowing more concurrency in the process with fewer human resource constraints. The planned schedule for the Dreamliner assumed a schedule reduction of 33% over previous similar development processes. This belief demonstrates a significant misunderstanding of the complex, interdependent nature of both aircraft products and the aircraft development process.

The primary problem with this schedule assumption is that not all tasks should be performed concurrently. If two design tasks that normally should be performed in series are instead performed in parallel, this introduces the need for additional iteration between the tasks, which could have otherwise been avoided. Overlapping sequential tasks has been described as "a core technique for saving development time [11], but Cho and Eppinger point out that it is generally acknowledged that overlapping tasks may save time, but is more costly than the traditional sequential approach, since it induces the need for additional iteration. [5] Moreover, the reduction in schedule assumes that the overlapped tasks may iterate freely when needed. But when iteration must occur between separate organizations through changing requirements, any time savings will likely be wasted, and development may take substantially longer. Thus, two design tasks can sometimes be completed more quickly when performed in series than by overlapping them.

8.6 Chapter Summary

The eighth chapter described how an increased trend in outsourcing product development has redefined the manner that design iteration can occur. The effects of the disintegration of the design chain are compounded by the division of design responsibilities along sections of the aircraft rather than systems, rendering the development process inefficient. This perspective offers insight into the source of changing customer requirements that disrupt the development processes of the suppliers. The following chapter will offer recommendations to address both the high level integration problems depicted in this chapter and the low level issues associated with the development processes of suppliers.

9. IMPROVING PRODUCT DEVELOPMENT

9.1 Overview

This section is meant to offer potential solutions to improve Spirit's product development process at various levels. These potential recommendations have been grouped by the issues that they attempt to address. Detailed evaluation of each of each solution is not undertaken here, but rather a brief justification is offered regarding how each solution could address an issue with the development process.

9.2 Advancing Understanding of the Development Process

In order to improve its product development processes, Spirit's managers and engineers must first gain a deeper appreciation for the complexities and interdependencies inherent in aerostructure design. This understanding must not only take place at high levels, but through all levels in the development process. Once task performers fully understand their role in the development process, they will adapt to minimize risk for themselves and their colleagues.

Institutionalize DSM

The Design Structure Matrix (DSM) is a very useful tool for depicting the flow of information within a complex process in a very organized and simple graphic. The DSM highlights sources of iteration that define the complexity of the process allowing employees to better understand their role in the process and managers to better understand how to control and improve the process. In order to increase the accuracy of simulation results, as well as to have more accurate numeric DSM representation, task characteristics and dependency data should be monitored. With more accurate input data, the simulation model could be used as basis for product development cost and schedule estimation. Ulrich and Eppinger note that facilitating information exchange is for complex development processes is critical to its performance. [17] With a better understanding of information flow paths, employees will be able to communicate more effectively in the development process.

Institutionalize the Process FMEA

The Failure Mode & Effects Analysis (FMEA) is a useful tool for understanding and managing risk. The standardization of the FMEA would result in a better understanding of systemic risk by all of the employees participating in development efforts. The FMEA's may be originally informed through subjective input and simulation results, but with the creation of a process failure modes and corrective action tracking system, the FMEA would become more informed by historical data allowing for long term calibration.

9.3 Integrating Interdependent Tasks

Because interdependency is inherent in its development processes, advancement of product design at Spirit is highly iterative. This iteration should be facilitated since it can't be eliminated. The best way to facilitate iteration is to ensure that the interdependent tasks are highly integrated. Since such a high degree of interdependency is across functional lines, it is very important to remove inter-functional barriers so that iteration occurs with resistance.

Minimize Functional Grouping on Programs

In Wichita, the functional grouping within programs at Spirit is very strong. By recreating a program organization based off of the product architecture rather than the functions, design iteration is more likely to be both expedited and reduced since staff performing interdependent tasks are more tightly coupled. This sort of product-based grouping is not uncommon in

complex development programs, and is already essentially enacted at Spirit's Prestwick site. By integrating functions at a grass-roots level, interdependent tasks in the development process will be more tightly coupled and may iterate more rapidly or even concurrently.

Create Functional Centers at the CTO

If functional grouping is minimized at the program level, it will still be necessary to ensure that the functions are still represented and maintained within the organization. In order to achieve this, it is recommended that functional centers are created within the CTO either under or alongside the program management center in the CTO. These centers will ensure proper career development opportunities, training, software selection, and best practices across the organization. There should also be links established between each program and the functional centers such that employees have a place to go for functional questions related to the program.

Split Role of Stress in Design & Certification

Since there are two customers of stress analysis in the development process – design and certification – which have different but related needs, the stress group should be split such that it can better meet the needs of each of those customers individually. In this manner, stress for design will not be obliged to perform certifiable analysis on each iteration and may therefore adopt tools and practices which enable faster iteration and design evolution. Certification analysis would then be viewed as a one-time, effort-intensive quality check of the final design. This would allow design analysis to adopt a more appropriate amount of risk in order to save time and effort, much like the role of rapid prototyping in a design-build-test process.

Be Aware of Integrated CAD/CAE Tools

The state of technology of computer aided design and engineering tools is constantly evolving. In more recent years, there have been major advancements in the integration of stress analysis tools into the CAD environment. Although these integrated tools are not yet mature enough to apply to aerostructure products, they will reach that state. When these tools are mature enough, the largest iterative loop in the development process will be reduced to nearly nothing providing significant reductions in the development cost and schedule. One significant issue, however, is that the organizational structure of the development process at Spirit is not suited for this level of integration. Spirit will need to rethink the roles of design and stress engineers and how they interact when this technology reaches a maturity tipping point.

Innovate in Structures/Systems Integration Practices

The interdependency between systems and structures design inherently causes design iteration. Innovation in integrating systems into the structural design would not only be a basis for product differentiation and competitive advantage, but could also eliminate iteration within the development process. Because of the dual benefit of better integrating systems and structures, investment in this area is very likely to produce a substantial return.

Outsource Only Independent & Non-Iterative Development Tasks

There do exist particular aspects of the development process which may be outsourced to a low cost engineering service provider without significantly increasing the complexity of the development process. Such tasks are characterized by: minimal interdependency to other tasks in the development process, minimal likelihood of iteration, easily communicated statement of work and instruction, and relatively large workload. If such characteristics of a task are met, then it should be considered for low-cost outsourcing. Two such items include creation of final product drawings (based upon CAD designs) and certification analysis & documentation of the final design.

9.4 Managing for Change

Managing for change means controlling the development process with the understanding that changing requirements are part of the process. This includes maintaining a smoothly functioning change management processes, but also includes preparing the process and its participants for change. Assuming that progress will proceed perfectly as planned is not only a poor assumption, in the design of aerostructures it is probably the worst assumption.

Create a Scope & Change Program Manager Position

The most simple way to prioritize managing for change would be to create a position for a scope and change manager at the highest level of the development program. This person would then be responsible for interfacing with the OEM on all issues relating to changing requirements. Incentive structures would be developed by this person to encourage better communication of requirements maturity and to help the OEM understand the necessity of flushing out requirements as early as possible.

Develop a Wiki to Manage Change

Communicating requirements changes and engineering change orders throughout the development process can become very tricky when the number of participating employees is large. A wiki-style interface could allow for the employees to have easy, searchable access to all of the external changes impacting the development effort. The OEM could also be given access to request changes through this system, essentially using the wiki as a platform for managing the change process. Finally, if horizontal development partners may be given access to this change system, then all participating organizations could become aware of impending change upon their processes due to external changes, essentially demystifying the source of changing requirements.

Staff Based on Requirements Maturity

With the knowledge of the customer's perception of the maturity of their requirements, a different approach may be taken with regards to staffing a program. Since work completed against mature requirements is more valuable than work completed against immature requirements, the maturity rating of the requirements should serve as a basis for staffing a new product development program. In order to minimize staff levels throughout a program, effort should be made to delay staff ramp-up of a program until customer loads have matured. This recommendation is driven by two factors – customer-driven load scenario maturity dictates the ramp-down of a program, and the load scenario maturity is mostly independent from the maturity of Spirit's design. Although the customer would like to see early ramp-up in order to minimize their perceived risk on Spirit's end, the customer's perceived risk mitigation comes at Spirit's expense of inability to meet internal cost and cycle time targets.

Create Incentives for Early & Expedient Changes

The timing of a requirements change can drastically reduce or increase the systemic impact of that change on the development process. Based on this understanding, contractual agreements should create incentive for the OEM customer to flush out requirements changes as early as possible. This would not only incentivize the OEM to allow more informal communication between development organizations, but would also encourage the OEM to stop overlapping sequential development tasks across the development organizations, resulting in less iteration and fewer requirements changes.

Create Incentives for Communicating Requirements Maturity

In order to foster more accurate and open communication between Spirit and its customers regarding the maturity of requirements, development schedule, and actual progress, both parties must have a financial incentive to do so. If such an incentive does not exist for the customer, that customer will always mitigate schedule risk by representing its requirements as more mature than they actually are. This mitigation of the customer's risk comes at great expense to Spirit since it would then staff the program at levels beyond those justified by the maturity of the requirements.

Cost sharing, if properly structured, would incentivize the customer to accurately depict the maturity of the requirements, since they would bear the cost of overruns associated with overly optimistic portrayal of the level of maturity. Overly pessimistic portrayal of requirements would result in prolonging the product development schedule more than would have otherwise been necessary. The proper structure for cost sharing should be based upon the accuracy of the communicated maturity of the loads. Not only would this structure for cost sharing force OEM's to pay for overruns that were caused by them, but it would create incentive for both parties to eliminate unnecessary cost in the first place.

Give Notice to Downstream Tasks of Impending Change

One very specific action that managers can take to reduce the impact of change is to communicate that change is impending to the downstream tasks of the task directly affected by the change. This would allow the process to better brace against change by preemptively planning for the effects of specific changes as early as possible.

Increase Human Resources Flexibility

The baseline development process for aerostructures contains a significant amount of variability, and changing requirements further compounds it. If the staffing of a given development process is inflexible and human resources cannot be easily added or removed, there will be very significant waste. The ability to shift human resources as the process requires them will significantly reduce development costs. Although it is not realistic to achieve a perfectly flexible human resource pool, any increase in flexibility will reduce costs.

9.5 Making the Process Robust to Change

Change is a reality of product development in the aerospace industry. Because the overall aircraft design and each of its interdependent subsystems are developed concurrently, change cannot be eliminated. Therefore, the most suited process for product development is not one of speed under ideal or planned circumstances, but one of robustness to change. In the current process at Spirit, change is very disruptive. Change creates panic for management and frustration for engineers. A development process that is robust to change means that change is expected and planned for, and that excess effort toward meeting immature requirements is minimized. The goal of the proposed recommendations and development structure below is to ensure that change typical on new programs does not disrupt the process and does not result in overruns.

Create Requirement Maturity Ratings

The degree of maturity of customer requirements is currently roughly acknowledged by the classification of loads as either preliminary, design, and final. Three tiers of maturity, however, do not offer enough resolution to differentiate between incoming changes that may differ greatly in maturity. Several criteria (listed below) can be used to quantify the maturity of a given set of loads.

- *Wholeness* – the extent to which all scenarios have been translated into load conditions;
- *Feasibility – the likelihood that the current design and loading conditions allow for a feasible solution and does not require a substantial aerodynamic and conceptual redesign;*
- *Set Accuracy – the percentage of all loading conditions within the entire set of loads that should not see any future change;*
- *Average Load Accuracy – the average extent to which the magnitude (and sign) of each load condition is accurate (accuracy variance may also be a useful metric).*

These characteristics of loads maturity may be explicitly used by Spirit and the OEM, or a more general set of maturity ratings may be developed based on minimum values of these characteristics. If a general set of maturity ratings is implemented, a high enough number of ratings must be chosen in order to provide adequate resolution to distinguish between loads of greatly different maturity and to justify staffing changes during a program. In either manner, the accuracy of the communicated maturity of each set of loads can be evaluated by comparing each load set against the final set of load conditions at the end of the program. The accuracy of communicated loads maturity should be used as a basis for cost sharing of program overruns. It should be noted that these criteria (except feasibility) can and should be tracked, even without any sort of cost sharing structure.

Monitor, Predict & Communicate Requirements Maturity

Every effort should be made to understand the evolution of requirements. The most basic form of this is subjectively estimating the maturity of requirements. A more advanced approach involves evaluating the requirements maturity progression of historic development processes, and using this as a basis for predicting the maturity of a current development process. This is possible since the final requirements of historic programs are known, so all preliminary and intermediate requirements can be measured against the final ones. The best approach, however, would be to communicate with the OEM customer in order to understand how they perceive the maturity of the current requirements.

Relax Requirements

One method for increasing the robustness to change is to relax certain design requirements based upon the maturity of the requirements. This methodology is analogous to assigning tolerances to or specifying a range for the requirements. For example, if the requirements are immature, then they could be said to have loose tolerances. The tightness of the tolerances dictates the amount of effort that should be spent to achieve the target requirement. This has an effect of relaxing the constraints on the solution space for the design such that no more effort is exerted to achieve the target requirements than is justified by the requirements maturity.

One way to accomplish this is to relax the target weight for the design based upon the maturity of the requirements. The preliminary design target weight may be increased by a large factor, which would gradually be reduced as the requirements mature until the late stages of product development when the design weight is reduced to exactly the target weight. This methodology would create a process that is much less sensitive to incoming changes, and therefore, would not be radically disturbed by major changes early on. In addition, this would allow programs to delay ramp up to the later stages of design since relaxing design weight would result in a simpler engineering problem. A delayed ramp up would create a situation where most of the work is performed against mature requirements rather than wasting the effort against immature requirements.

If the requirements relaxation method is to be used, then a "proof of concept" should be developed after concept selection. The proof of concept should be used to indicate that the chosen concept has a high likelihood of meeting the target customer requirements. Developing a proof of concept should eliminate the risk that relaxing requirements may delay feedback of poor concept selection and should allow the OEM customer to be more comfortable with this approach.

Ramp Fidelity

Similar to relaxing the requirements for the product, the fidelity of the design can also be adjusted to correspond to the maturity of customer requirements. Fidelity refers to the extent of detail present in the design. In other words, early in the development process when requirements are immature or preliminary, the design effort should be very conceptual. As the requirements mature, more detail can be added until the requirements and then the design are finalized.

Phase Detail Design

Currently there are three general phases in Spirit's product development process – joint concept development phase (JCDP), detail design, and release & certification. Spirit could greatly benefit from imposing additional structured phases onto the process, particularly in detail design. Since the maturity of the customer's requirements can vary greatly in the detail design, it would be appropriate to subdivide that portion of the process into additional phases such that the process acknowledges that excessive detail and analysis are not justified early on. Additionally, staffing policy can now be referenced against the timing of the phases since the addition of more phases provides more resolution. For example, tooling staff may be needed in the initial detail design phase for general feedback and the final design phase for detail feedback, but less so in the middle phases when the details are being flushed out. Other policies could also be created in reference to the phases. For example structures and systems may design independently in the first detail design phase, but should focus on integration in the second phase. By breaking the development process into smaller blocks, more specific direction can be given to process participants regarding what they should be doing and what they should be prioritizing. More control can also be exerted over the rate of iteration of the different interdependent process steps. Finally, the additional phases will provide more opportunities to review the design against intermediate expectations and adapt subsequent phases to specific areas.

9.6 Capturing Intellectual Property

Intellectual property (IP) is important for an OEM because it dictates the starting point for development efforts. If, for example, Spirit retained no IP, development would start from a blank slate. With higher retention of IP, less redundant effort is required for subsequent development efforts. The potential recommendations below correspond to increasing levels of IP retention. Two important caveats to these recommendations are that Spirit must invest in its own development initiatives to create standard, reusable designs and that the OEM customer may have preferences regarding the extent of standard design usage.

Pre-Engineer Standard Component Designs

Spirit could invest in the development of standard components that could be used in multiple products. This is a very low level of IP retention that could easily eliminate redundant tasks across the products. To a certain extent, standard component designs are already used, but further expansion is assuredly feasible.

Create Load-Path Platforms

Taking standard components a step further, Spirit could standardize conceptual designs for its products. Each of these preliminary designs would essentially consist of low-fidelity load-path geometries. One major benefit of this approach is that it standardizes at a higher level than just component designs, which would allow for development of CAD tools specific to each load-path platform to streamline the design efforts required to add design fidelity and integrate the various component designs. Another advantage of this approach is that the costs to create these platforms should not be excessive, making this a low risk investment.

Create a Product Catalog

The most extreme form of IP generation could result in creation of a product catalog. Aircraft OEM's could select from the catalog the design that best fits its needs, and Spirit could tailor that design to meet the OEM's specific needs. This approach would appeal to OEM's since it could drastically reduce the costs and schedule of development, but the OEM's may not be partial to using designs similar to those of their competitors. There also may be a significant amount of risk in developing high fidelity designs that customers may not want.

9.7 Architecting Product Development of Aircraft

Chapter 8, "Integrating Product Development in the Aircraft Industry," showed the extreme complexities incurred from developing aircraft across multiple organizations and from dividing design responsibilities based on sections of structures rather than systems. That analysis shows the significant benefits of vertically integrating design and development efforts of a complex product. However, there are other factors, the most significant of which is the prevalence of offsets, that force the OEM's to outsource. Even if only manufacture were outsourced, the design for manufacture activities with suppliers would become intrinsically dispersed. Since complete vertical integration is not reasonable, the best option may lie in a better architecture for product development of aircraft.

Design by Systems, Supply by Sections

The first aspect of the proposed architecture for developing a large aircraft is to design by systems, rather than sections. This change particularly applies to the development of the structures in the aircraft. Thus a single organization would be responsible for developing the aircraft structures. The design of the structures should take place much more rapidly since the sections will be developed in an integrated effort. The supply, however, should be maintained in sections in order to minimize production costs. As previously stated, there is significant interdependency with design of the production process and tools with the design of the product. Thus, each sections supplier should coordinate with each systems designer in order to incorporate design for manufacture into the development process and to allow the suppliers to design the tools and manufacturing processes somewhat concurrently with the product design. This architecture should alleviate major issues with disintegrated interdependency in the design chain, while allowing the OEM's to produce products in their customers countries to ensure sales there. One disadvantage of this approach is that the OEM's would need to select a single design firm for each system, which may not ensure the equal footing for most tier-one competitors that currently results in significant cost-based competition. Since Spirit has the capability to design all structural components of a large aircraft, it is positioned to move into the system designer role. Although this role may require organizational restructuring to operate as both a structures system designer and a sections supplier, such a role could drastically increase the company's value to an OEM who prefers to play the systems integrator role, further ensuring future strategic business.

9.8 Chapter Summary

The ninth chapter offered recommendations to improve product development at various organizational levels. Recommendations are presented in the following categories: advancing understanding of the development process, integrating interdependent tasks, managing for change, making the process robust to change, capturing intellectual property, and architecting product development of aircraft. The recommendations would require varying degrees of effort and have varying likelihoods for successful implementation. The following chapter will offer an approach for identifying the organizational suitability of an initiative, which can serve as a basis for selecting recommendations or for developing tactics to increase the likelihood of successful implementation.

10. IMPLEMENTING CHANGE IN AN ORGANIZATION

10.1 Three-Lens Methodology

A particularly useful method for evaluating the practicality of a proposed improvement initiative or change involves evaluating it through three "lenses" – structural appropriateness, political alignment, and cultural suitability. Through the three lenses, obstacles to successful implementation are identified in the context of the specific organization. The results may be used to prioritize proposed initiatives by ease of implementation. The results may also be used as a basis for developing implementation tactics that will most likely yield success for a chosen initiative. Either way, it is important to evaluate proposed changes in the context of the specific company, since organizational dissonance alone may prevent successful implementation. [3]

10.2 Structural Appropriateness

The structural (or often called "strategic design") lens analyzes the proposed initiative from the perspective of how information flows through an organization. The structural lens determines how the architecture of the organization affects the way that tasks are accomplished and value is added in an organization. The structure dictates the rules by which employees abide in order to complete their tasks. In order to control activities, managers create work groups, links between groups, goals and responsibilities for the groups, and rewards and incentives to motivate success. The level of understanding of the structure by the employees can often impact the performance of the organization. [8] The degree to which an initiative can be deployed within the structure of the organization determines its structural appropriateness. Initiatives that require new groups and linkages may not be structurally appropriate for the organization. This, however, may indicate weaknesses in the current organizational architecture to face future market necessities regardless of the proposed initiative.

The organizational structure at Spirit is characterized by its manufacturing roots and mentality. Spirit has historically viewed product development not necessarily as a value added activity, but rather as a necessary cost that must occur before manufacture can begin. This is largely a relic of the role that the facility played as a Boeing site prior to the divestiture. In the past, this paradigm and structure has led Spirit to prioritize manufacturing improvement more so than product development improvement.

The current execution of product development takes place largely in a de facto manner where participants in the process continually react to changes in inputs in order to complete their deliverables. This has led to somewhat of a fire fighting mentality among engineers and managers, where process implementation is prioritized above process improvement. This is not unique to Spirit and seems to be common within the industry. Such a mentality leaves very little time to create standard processes and tools, which could reduce the fire fighting that occurs in subsequent efforts. Highly functional grouping is another former Boeing trait, where employees that participate in the product development process are grouped primarily based on their skill sets rather than the part of the product architecture. Any improvement initiatives must acknowledge these characteristics of the organizational structure pertaining to product development in order to increase the odds of success.

10.3 Political Alignment

The political lens analyzes the proposed initiative from the perspective of who maintains power within the organization and how the initiative aligns with their goals and perceptions. Employees with varying interests will resist or promote the initiative based on how they perceive

it will impact their interests. The level of power of those promoting versus resisting the initiative, may very well determine the success of the initiative. Individuals and groups may obtain their power through a variety of channels such as the organizational hierarchy, individual competence, and their ability to convince or motivate others in the organization, but all of these sources of power may be classified into two categories – bestowed power and de facto power. Bestowed power is given through the organizational structure, while de facto power is obtained on a cultural basis. The degree to which stakeholders with both bestowed and de facto power will promote the initiative determines its political alignment. Initiatives that require shifting sources of power from certain individuals to others may not be aligned with the current political situation. It is possible, however, for bestowed power to be shifted throughout an organization, so lack of political alignment may not necessarily prevent success of an initiative.

At Spirit, most significant power is bestowed from the organization. This is evident in part by the proliferated presence of organizational charts in cubicles, on walls, and in presentations. One major reason for the weighting toward bestowed power is the major presence of both manufacturing and engineering unions at the company, which explicitly indicate assignment of powers through contracts. These unions have left an indelible mark on the company regarding definition and division of responsibilities, which even finds its way into the mode of operation in management. For example, much care is always taken not to step into the responsibilities of one's colleagues, even when not mandated by contract, which can make cross-functional change difficult to coordinate and integrate throughout the company.

With regard to product development, pieces of power lie at three different tiers of the organization. First, the engineers retain essentially veto power for any change initiative since they understand how a change might impact the technical and safety aspects of product design. That said, this group is very comfortable with the status quo since they are typically well paid and possess a high degree of job security as part of the engineering union. In addition, they have very little concrete incentive to improve the cost or cycle time of a product development program since there is little to no significant financial reward for better performance. This group does not typically envision a better performing process and is generally skeptical when it comes to change or improvement. However, since this group does in effect have veto power, it's important to gain their appreciation for any change or improvement initiative. The key personnel for this group are the engineering leads and the technical fellows. These are the engineers with the most technical competency, experience, and process insight, who have the ability to motivate and convince other engineers in their organizations.

The second group, middle management, is probably the least empowered to enact change since the engineering leads control the technical aspects of the design process and the chief technology office controls the official development process and tools. However, this group is the most in touch with the problems, since they are evaluated against the performance of the development process and interact with the process participants on a daily basis. In addition, they are better positioned to see how a problem might affect various parts of the process and to understand the root cause of the problem. Generally, a manager may carry clout with a few engineering leads since management typically came up through engineering, but that clout is typically limited to the function that the manager had previously served in. This group may be the most capable of solving process problems, but seems to be the least empowered to do so.

The final group of power is the leadership, or upper management, including program managers, company directors, and vice presidents. This group carries power both politically and culturally at Spirit. Program mangers have the power to commission and lead change, however, they are typically very overwhelmed with other short-term issues which may prevent them from inventing

and implementing improvements. Directors and VP's are well positioned to commission improvement activity. Generally, this group may be too distanced from the development process to offer much specific insight as to improving it, even though they have the most power and resources to solve issues. In order for any significant change to occur, the clout and resources of upper management, the holistic understanding of the middle management, and the technical knowledge and buy in of the engineering leads will all be required.

10.4 Cultural Suitability

The cultural lens analyzes the proposed initiative form the perspective of its suitability regarding the norms, assumptions, values and informal relationships within the organization. The culture of a company is not explicitly created by individuals or leaders, but is rather the aggregation of the organization's behavior. Many aspects of the culture can be cultivated by its leaders, for example by removing staff who conduct business dishonestly or by rewarding individuals who exhibit technical competence. However, it is impossible to completely control an organization's culture. Initiatives that require significantly changing the norms, assumptions or values rather than reinforcing the current ones may not be culturally suitable. Lack of cultural suitability may pose the most difficult obstacle to overcome since culture is very difficult to change as it is somewhat a product of the company's cumulative history. However, it is often possible to change aspects of an initiative to make it more culturally suitable without compromising the purpose of the initiative.

Since Spirit's divestiture from Boeing, much has changed regarding the business plan, but the company's culture has been more slow to react. This is largely due to the fact that the divestiture happened very comfortably with large contracts in hand and without much crisis. Through the cultural lens, Spirit is an older, massive company with a highly experienced, but aging workforce, and with firm roots. This engineering culture does not lend itself easily to change, and is extremely skeptical of improvement initiatives that seem to be the next "flavor of the month." This is especially true of initiatives that do not seem to take into consideration the technical aspects of product development or which tend to oversimplify the process. For change to occur within the product development process, much effort must be taken to prove why and how the change can improve the process, rather than "marketing" the change to this skeptical technical audience.

In addition, the engineering organization seems to have an appreciative self-perception. Many engineers view the company as a "world leader" in general product design, which may be true in certain areas, but may lead to complacency where improvement opportunities lie. The lack of urgency and sense of comfort of the engineering organization has largely prevented Spirit from enacting significant change. For product development process improvement to occur substantially, more urgency is required than is portrayed through the comments of one technical executive comparing product development improvement at Spirit to "Tiger Woods remaking his golf swing."

There is also somewhat of a cultural divide between engineering and management. Organizationally these groups are connected, but they sometimes have very different priorities and even values. Many engineers are very skeptical of management because they fear that safety or engineering integrity may be compromised. This leads to a nothing-less-than-perfect mentality in engineering where many don't trust that the process will ensure that the best and safest design evolution will occur. The lack of trust in the process sometimes lead to over-engineering. One of the primary reasons that engineers are skeptical of management and improvement initiatives is that they sometimes perceive a lack of technical understanding and

oversimplification by management. Recommendations accompanied by a technical or quantitative analysis would go a long way in appealing to this group.

10.5 Chapter Summary

The tenth chapter described an approach for assessing the organizational suitability known as the Three Lens methodology. This approach seeks to understand the organization in terms of an organization's structure, politics, and culture. A brief analysis of Spirit's organization through the three lenses is offered as an example, but as an outsider's perspective, this view of Spirit should not be taken as complete or necessarily accurate. The following chapter offers the final conclusions from this thesis.

11. CONCLUSION

11.1 Key Learnings

As a result of the work for this thesis and the research opportunities provided by Spirit, several key learnings should be taken:

- *Linear planning tools are not appropriate in developing complex products.* In development, the plan is an approximation. If only downstream flow of information and causality is incorporated, the plan is a poor approximation. Interdependencies in the product and process may cause substantial iteration. A culture and a system that adapts to internal and external change rather than resisting it will perform much better.

- *DSM is an effective way to describe a product or process and to illustrate systemic complexity.* The matrix form allows for neatly organized information that can otherwise only be shown through difficult to read "spaghetti diagrams." The DSM neatly organizes tasks and shows the inter-task dependencies. Downstream flow of information, feedback loops, and process failure modes are easily identified and can even be easily quantitatively illustrated.

- *The organizational grouping in product development should depend on the company's priorities, strengths and weaknesses.* Functional grouping can have the effect of developing knowledgeable experts and facilitating advancement of functional tools and practices. Grouping based on the product architecture can have the effect of better functional integration in the development process, facilitating iteration. It is important that organizations find the optimal balance of these two types of grouping according to their priorities. An organization seeking performance improvement should consider minimizing functional grouping.

- *DSM-based simulation is useful from the insight that it offers, but the underlying data may not be accurate enough to offer detailed predictions of future development processes.* A manager who does not understand the dynamics of the development process will not be able to effectively manage it, and DSM-based simulation can help managers better understand the processes that they oversee. However, most task performers don't understand the role of their task in the development process, and they don't understand what portion of their work is due to changing requirements, internal or external, and iteration. The simulation tool developed for this thesis is only as accurate as the underlying data that is input to it, which at this point needs calibration against more sources and against historic development process performance before it could be used as an accurate planning tool. It is, however, more useful than tools that don't incorporate iteration.

- *Human resource flexibility can be extremely valuable in developing complex products.* Since the actual human resource demands of a process are difficult to predict and may come in waves, the ability to quickly add or remove people to and from the process can drastically reduce the final cost of development.

- *Disintegrated development efforts for an interdependent design are very inefficient.* The nature of systems-based grouping is to combine interdependent components so that they can be developed together. Grouping without considering interdependency intrinsically will require substantial inter-organization communication, which in formal processes results in excessively changing requirements that could have otherwise been avoided. This is particularly true of the sections architecture prevalent in the aircraft industry.

- *Long development processes should be divided into more manageable periods in order to better control the process and to provide more structure for the participants.* The development process for a complex product may take years. During that time substantial internal and external changes may occur which lead to iteration. By introducing phases upon on the development process, task participants can easily adapt their work in a manner consistent with each phase, and managers can more easily set priorities and more easily gauge the state of the development process. Less phasing results in a more de facto process, which is difficult to control.

- *Optimal staffing policy should incorporate the expected evolution of external requirements.* It does not make sense to pursue preliminary or immature requirements as though they are the final requirements. Such an approach would be analogous to sprinting to the finish line, when it is certain that the finish line will change locations. A more efficient approach would be to staff more lightly initially and more heavily as the maturity of requirements advances.

- *Capturing product design IP is a good way to reduce development costs.* Although each product developed by Spirit is individual and unique, there are commonalities that exist across similar products. The more intellectual property that Spirit can capture regarding product cross-commonality, the more it can eliminate redundancy from redeveloping these commonalities. Thus, capture of design IP can help improve the cost and schedule of future product development processes.

- *Organizational barriers can thwart any improvement or change initiative.* It is very important to consider the structure, politics, and culture of an organization when developing a tactical approach to implementing change.

11.2 Future Work

During the development of this thesis and during the author's internship at Spirit, several relevant opportunities for future research became apparent:

- Using Wiki's to facilitate communication in complex development processes.

- Applying integrated design and analysis software tools in developing aerostructures.

- Architecting a product development process that is robust to external change.

- Using a dynamic decision simulation to determine the optimal staffing policy.

- Developing requirements maturity ratings and evaluation of the historical requirements evolution rates of various products and industries.

- Studying the relation between design chain disintegration and changing requirements.

- Exploring the effects of developing aircraft structures by sections rather than systems.

11.3 Final Remarks

Superior product development is not only of importance for reducing costs, it can provide a strategic advantage for tier-one suppliers. In the age of outsourcing, new complexities in product development have arisen. The companies that can master these complexities and develop products efficiently will substantially reduce risk for the OEM's. This may provide opportunities for strategic partnerships in the best case, or will at least create guarantees for future business. However, the linear mentality of construction project management will have to end. In its place a deeper understanding of the nature of developing complex products is required. Organizations that can master the system dynamics perspective of product development will rise to the top. The static organizations that refuse to change will sink. This thesis provides a toolset for companies to better understand, model and improve their development processes. But more importantly, it provides a paradigm that if institutionalized can drastically improve the performance of product development in any industry.

APPENDIX A - MATLAB Code for Simulation Model

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DSM-Based Monte Carlo, Discrete Event Process Simulation
% by: Brad Rogers 2008
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Sequence
%-------------------------------------------------------------

  warning off

% Read input data

  control_data =      xlsread('main_data.xls','Control');
  group_data =        xlsread('main_data.xls','Groups');
  eco_data =          xlsread('main_data.xls','ECO');
  task_data =         xlsread('main_data.xls','Tasks');
  couplings_data =    xlsread('main_data.xls','Couplings');
  failures_data =     xlsread('main_data.xls','Failures');


% Create DSM Dependency Matrices
  dependency =        zeros(size(task_data,1));
  rework =            zeros(size(task_data,1));
  concurrency =       ones(size(task_data,1));

  for m = 1:size(couplings_data,1)
    dependency (couplings_data(m,1),couplings_data(m,3)) = 1;
    rework(couplings_data(m,1),couplings_data(m,3))      = couplings_data(m,5);
    concurrency (couplings_data(m,1),couplings_data(m,3)) = couplings_data(m,6);
  end

 % Establish Simulation Parameters
  max_trials =        control_data(4,1);
  hist_bins =         control_data(5,1);
  show_events =       control_data(6,1);

  task_group =        task_data(:,3);
  task_work_min =     task_data(:,5);
  task_work_med =     task_data(:,6);
  task_work_max =     task_data(:,7);
  task_lcf =          task_data(:,8);
  task_final_learning = task_data(:,9);
  task_staff_max =    task_data(:,10);
  task_convergence =  task_data(:,11);
  task_max_number =   length(task_lcf);

  group_staff_total = group_data(:,4);
  group_wage =        group_data(:,3);

  eco = eco_data;
  executed_events = [];
  staff_levels = [];


% Generate random numbers matrix for task completion times and process failures
  randnummat=sample_matricizer(task_work_min,task_work_max,task_work_med,max_trials);
  B = ones(size(failures_data),max_trials);
```

```
    for m=1:max_trials
       B(:,m)=B(:,m).*failures_data(:,7);
    end
    randnumfails = (rand(size(B))<=B);

% Allocate memory for results
    sim_result_time = zeros(max_trials,1);
    sim_result_cost = zeros(max_trials,1);
    sim_result_work = zeros(max_trials,1);
    U = 1000; %U should be set to the expected number of process events
    staff_levels = zeros(max_trials*U,3);
    events = 1;

% Discrete Event Initial State
%-----------------------------------------------------------------
for trial_count = (1:max_trials)
    disp(trial_count) %to track trials on screen

       % Reset calculated values and lists
          wip_list = [];
          clock = 0;
          clock_advance = 0;

          group_work_cum =       zeros(length(group_wage),1);
          group_cost_cum =       zeros(length(group_wage),1);
          group_staff_available = group_staff_total;
          cost_cum =             0;
          work_cum =             0;
          task_work_cum =        zeros(length(task_work_min),1);
          task_iteration =       zeros(length(task_work_min),1);
          task_staff =           zeros(length(task_work_min),1);
          task_converged =       zeros(length(task_work_min),1);
          task_input_change =    ones(length(task_work_min),1);

       % Generate initial wip list from tasks that don't require inputs
          for q = (1:1:task_max_number)
             if isempty(find(dependency(q,1:q-1)==1))
                wip_list = vertcat(wip_list,q);
             end
          end

       % Prioritize and Staff Tasks
          wip_list = sort(wip_list);
          for b = (1:length(wip_list))
             staff_lack = task_staff_max(wip_list(b)) - task_staff(wip_list(b));
             if group_staff_available(task_group(wip_list(b))) > staff_lack
                task_staff(wip_list(b)) = task_staff(wip_list(b)) + staff_lack;
                group_staff_available(task_group(wip_list(b))) = group_staff_available(task_group(wip_list(b))) -
staff_lack;
             else
                task_staff(wip_list(b)) = task_staff(wip_list(b)) + group_staff_available(task_group(wip_list(b))) ;
                group_staff_available(task_group(wip_list(b))) = 0;
             end
          end

          group_staff_used = group_staff_total - group_staff_available;
          hotlist = [0 0 0 0 0];

       % change status variables
          initiation_status = task_staff > 0 ;

       % initialize task work loads for this run
```

```
        task_work_initial = randnummat(:,trial_count);
        task_work_final = task_work_initial.*task_final_learning;
        task_work = max((task_work_initial .*((1-task_lcf).^task_iteration)),task_work_final);
        task_work_left = task_work;
        task_fraction_complete = (task_work - task_work_left) ./ task_work;
        task_time_left = task_work_left ./ task_staff;

      % initialize run failures into couplings matrix
        iter_limit = zeros(size(task_work_min,1));
        for m = 1:size(failures_data,1)
          if randnumfails(m,trial_count)==1
            dependency (couplings_data(failures_data(m,1)),couplings_data(failures_data(m,3))) = 1;
            rework(couplings_data(failures_data(m,1)),couplings_data(failures_data(m,3)))      = failures_data(m,5);
            concurrency (couplings_data(failures_data(m,1)),couplings_data(failures_data(m,3))) =
failures_data(m,6);
            iter_limit(couplings_data(failures_data(m,1)),couplings_data(failures_data(m,3)))  = 1;
          else
            dependency (couplings_data(failures_data(m,1)),couplings_data(failures_data(m,3))) = 0;
            rework(couplings_data(failures_data(m,1)),couplings_data(failures_data(m,3)))      = 0;
            concurrency (couplings_data(failures_data(m,1)),couplings_data(failures_data(m,3))) = 1;
          end
        end


% Main Discrete Event Loop
%-----------------------------------------------------------------
  while or(size(wip_list,1)>0, sum((eco(:,4)-task_convergence(eco(:,1))).*(eco(:,3)>0))>0)
  tic

      % Update ECO List with current time
        eco(:,3) = eco(:,7)-clock;

      % Create Completions List
        type = ones(length(wip_list));
        type(:,2:length(wip_list))=[];
        ones_list = 1 * type ;
        completions_list = horzcat(wip_list, type, task_time_left(wip_list), ones_list,ones_list);

      % Create Starts List
        starts_list = [];
        for c = (1:length(wip_list));
          task_pot_starts = find(dependency (:,wip_list(c)) == true);
          for d = (1:length(task_pot_starts))
            task_pot_start_inputs = find(dependency(task_pot_starts(d),:)==true);
            task_pot_start_inputs = find(task_pot_start_inputs <= d);
            task_pot_start_inputs = find(task_iteration(task_pot_start_inputs) > 0);
            if prod(task_pot_start_inputs) == 1
              if task_iteration(task_pot_starts(d))==1
                time_til_start =(1-concurrency(task_pot_starts(d),wip_list(c))-
(task_fraction_complete(wip_list(c))))*(task_work(wip_list(c))/(task_staff(wip_list(c))));
              else
                time_til_start =(1-
(task_fraction_complete(wip_list(c))))*(task_work(wip_list(c))/(task_staff(wip_list(c))));
              end
              if (hotlist(1,1)~=task_pot_starts(d) && hotlist(1,5)~=wip_list(c))
                starts_list = vertcat(starts_list,[task_pot_starts(d),3,time_til_start,
(task_input_change(wip_list(c)))*(rework(task_pot_starts(d),wip_list(c))),wip_list(c)]);
              end
            end
          end
        end
```

```
% Create Events List
   events_list = vertcat(completions_list,eco(:,1:5), starts_list);
   events_list = sortrows(events_list, 3);
   events_list(1:length(find(events_list(:,3)<= 0)),:) = [];

% Create Hotlist
   hotlist = events_list(1:size(find(events_list(:,3) < events_list(1,3)+0.001),1),:) ;
   hotlist = sortrows(hotlist,2);

% Remove process failures modes from DSM if initiated on hotlist
% (process failures may only occur once)
   for q = 1:size(hotlist,1);
      if iter_limit(hotlist(q,1),hotlist(q,5))==1
         rework(hotlist(q,1),hotlist(q,5))       = 0;
         concurrency(hotlist(q,1),hotlist(q,5))   = 1;
         dependency(hotlist(q,1),hotlist(q,5))    = 0;
         iter_limit(hotlist(q,1),hotlist(q,5))    = 0;
      end
   end

% Track run Staff Levels
   staff_levels(events,:) = [clock,sum(group_staff_used),sum(task_staff)];
   events = events+1;

% Calculate Clock Advance
   clock_advance = hotlist(1,3);
   clock = clock + clock_advance;

% Track first Events (may be used to calculate U on line 75
   if trial_count == 1
      k = ones(size(hotlist,1)).*clock;
      k = k(:,1);
      executed_events = vertcat(executed_events, horzcat(hotlist,k));
   end

% Update Calculated Variables
   group_work_cum = group_work_cum + (group_staff_used * clock_advance);
   group_cost_cum = group_cost_cum + (group_staff_used * clock_advance .* group_wage);
   cost_cum = sum(group_cost_cum);
   work_cum = sum(group_work_cum);
   task_work_cum = task_work_cum + (task_staff * clock_advance);
   task_initiation_status = task_staff > 0;
   task_work = max((task_work_initial .*((1-task_lcf).^task_iteration)),task_work_final);
   task_work_left = task_work_left - (task_staff * clock_advance);
   task_fraction_complete =(task_work - task_work_left) ./ task_work;
   task_time_left = task_work_left ./ task_staff;
   task_converged = 1 - (task_fraction_complete) <= task_convergence;

% Update WIP List for each event on Hotlist
   for f = [1:size(hotlist,1)]
      switch hotlist(f,2)
         case 1   % task complete
            group_staff_available(task_group(hotlist(f,1))) = group_staff_available(task_group(hotlist(f,1))) +
task_staff(hotlist(f,1));
            task_staff(hotlist(f,1)) = 0 ;
            task_iteration(hotlist(f,1)) = task_iteration(hotlist(f,1)) + 1;
            wip_list(wip_list(:,1)==hotlist(f,1),:) = [] ; % remove task from wip_list
            task_input_change(hotlist(f,1)) = 0;

         case 2   % requirements change;
            task_input_change(hotlist(f,1)) = 1-((1-task_input_change(hotlist(f,1)))*(1-(hotlist(f,4))));
            task_fraction_complete(hotlist(f,1)) = task_fraction_complete(hotlist(f,1)) * (1-(hotlist(f,4)));
```

```
            task_work_left(hotlist(f,1)) = (task_work(hotlist(f,1)))*(1-task_fraction_complete(hotlist(f,1)));
            if 1 - task_fraction_complete(hotlist(f,1)) <= task_convergence(hotlist(f,1))
               task_converged(hotlist(f,1)) = 1;
            else
               task_converged(hotlist(f,1)) = 0;
               wip_list = vertcat(wip_list,hotlist(f,1));
               wip_list = unique(wip_list);
            end

         case 3   % task initiated
            task_input_change(hotlist(f,1)) = 1-((1-task_input_change(hotlist(f,1)))*(1-(hotlist(f,4))));
            task_fraction_complete(hotlist(f,1)) = task_fraction_complete(hotlist(f,1)) * (1-(hotlist(f,4)));
            task_work_left(hotlist(f,1)) = (task_work(hotlist(f,1)))*(1-task_fraction_complete(hotlist(f,1)));
            if (1 - task_fraction_complete(hotlist(f,1))) <= task_convergence(hotlist(f,1));
               task_converged(hotlist(f,1)) = 1;
            else
               task_converged(hotlist(f,1)) = 0;
               wip_list = vertcat(wip_list,hotlist(f,1));
               wip_list = unique(wip_list);
            end
      end
   end

   % Sort wip_list by task number
      wip_list = sort(wip_list);

   % Staff Tasks
      for a = (1:length(wip_list))
         staff_lack = task_staff_max(wip_list(a)) - task_staff(wip_list(a));
         if group_staff_available(task_group(wip_list(a))) > staff_lack
            task_staff(wip_list(a)) = task_staff(wip_list(a)) + staff_lack;
            group_staff_available(task_group(wip_list(a))) = group_staff_available(task_group(wip_list(a))) -
staff_lack;
         else
            task_staff(wip_list(a)) = task_staff(wip_list(a)) + group_staff_available(task_group(wip_list(a))) ;
            group_staff_available(task_group(wip_list(a))) = 0;
         end
      end
      task_time_left = task_work_left ./ task_staff ;
      group_staff_used = group_staff_total - group_staff_available;
      if length(wip_list)==0

      end
   end % End discrete event run

% Record Trial results
   sim_result_time(trial_count)= clock;       % simulated trial results-time
   sim_result_cost(trial_count)= cost_cum;    % simulated trial results-cost
   sim_result_work(trial_count)= work_cum;    % simulated trial results-work

end % End Simulation Do Loop

% Format & Display Simulation Output
% ---------------------------------------------------------------------------

% Remove zeros from preallocated memory if not used (potentially inefficient)
   while staff_levels(length(staff_levels),1)==0
      staff_levels(length(staff_levels),:)= [];
   end

% Reformat task list into separate runs, swtich from event to time basis, calc stats
   expected_staff_levels=time_point_stats(time_align(trial_partition(staff_levels(:,1:2))));
```

```
% Display certain parameters on screen
   if show_events == 1
      disp('maximum executed events')
      disp(length(trial_partition(staff_levels(:,1:2))))
      disp('mean process duration')
      disp(mean(sim_result_time))
   end

% Create plots and histograms
   subplot(3,3,1)
   hist(sim_result_time,hist_bins)
   ylabel('Occurrences')
   xlabel('Business Hours')
   title('Distribution of PD Process Time','FontSize',16)

   subplot(3,3,2)
   hist(sim_result_cost,hist_bins)
   title('Distribution of PD Process Cost','FontSize',16)
   xlabel('Cost')

   subplot(3,3,3)
   hist(sim_result_work,hist_bins)
   title('Distribution of PD Process Workload','FontSize',16)
   xlabel('Man-Hours')

   subplot(3,3,4:6)
   scatter(staff_levels(:,1),staff_levels(:,2))
   xlabel('Time (bus-hrs)')
   ylabel('Group Staff Level (people)')

   subplot(3,3,7:9)
   scatter(expected_staff_levels(:,1),expected_staff_levels(:,2))
   xlabel('Time (bus-hrs)')
   ylabel('Expected Staff Level (people)')

% Output simulation data to an excel spreadsheet
   xlswrite('output_data.xls',expected_staff_levels,'staff');
   xlswrite('output_data.xls',sim_result_time,'schedule');
   xlswrite('output_data.xls',sim_result_work,'workload');

% The End

%-----------------------------------------------------------------------
function [output_mat] = sample_matricizer(a,b,c,N)

for i = 1:length(a)
   output_mat(i,:) =lhs_rng(tri_uni_divide(a(i),b(i),c(i),N),N);
end

%-----------------------------------------------------------------------
function [output_mat_1] = lhs_rng(input_mat,N)

used_bins = [];
bins = length(input_mat)-1;
minval = input_mat(1,1);
input_mat = input_mat(2:bins+1);
for j = 1:N
   bin = ceil(rand()*bins);
   while length(find(used_bins==bin))>0
      bin = ceil(rand()*bins);
   end
```

```
   used_bins(j)=bin;

   if bin == 1
      bottom = minval;
      top = input_mat(1);
   else
      bottom = input_mat(bin-1);
      top = input_mat(bin);
   end
   output_mat(j,1)= (rand()*(top-bottom))+bottom;
end
output_rank = zeros(N,1);
pick = ceil(rand()*N);
for i = 1:N
   while output_rank(pick)~=0
      pick = ceil(rand()*N);
   end
   output_rank(i) = pick;
end

for k =1:N
  output_mat_1(k) = output_mat(output_rank(k));
end

%-------------------------------------------------------------------------
function [output_mat]= trial_partition(input_mat)

%find longest series to generate initial matrix
last_time = 100000000;
max_trial_length = 0;
j=1;
trial=0;


last_time = 1000000000;
for i = 1:length(input_mat)
  if input_mat(i,1)< last_time
     trial_length = i-j;
     j=i;
     if trial_length > max_trial_length
        max_trial_length = trial_length;
     end
     trial=trial+1;
  end
  last_time = input_mat(i,1);
end



output_mat = zeros(max_trial_length,2*trial);

last_time = 100000000;
j = 1;
k = 1;
m = 0;
for i = 1:length(input_mat)
  if or(and(input_mat(i,1)< last_time,m==1),i==length(input_mat))

     if i ==length(input_mat)
        trial_length = i-j+1;
        output_mat(1:trial_length,k:k+1)= input_mat(j:i,1:2);
     else
```

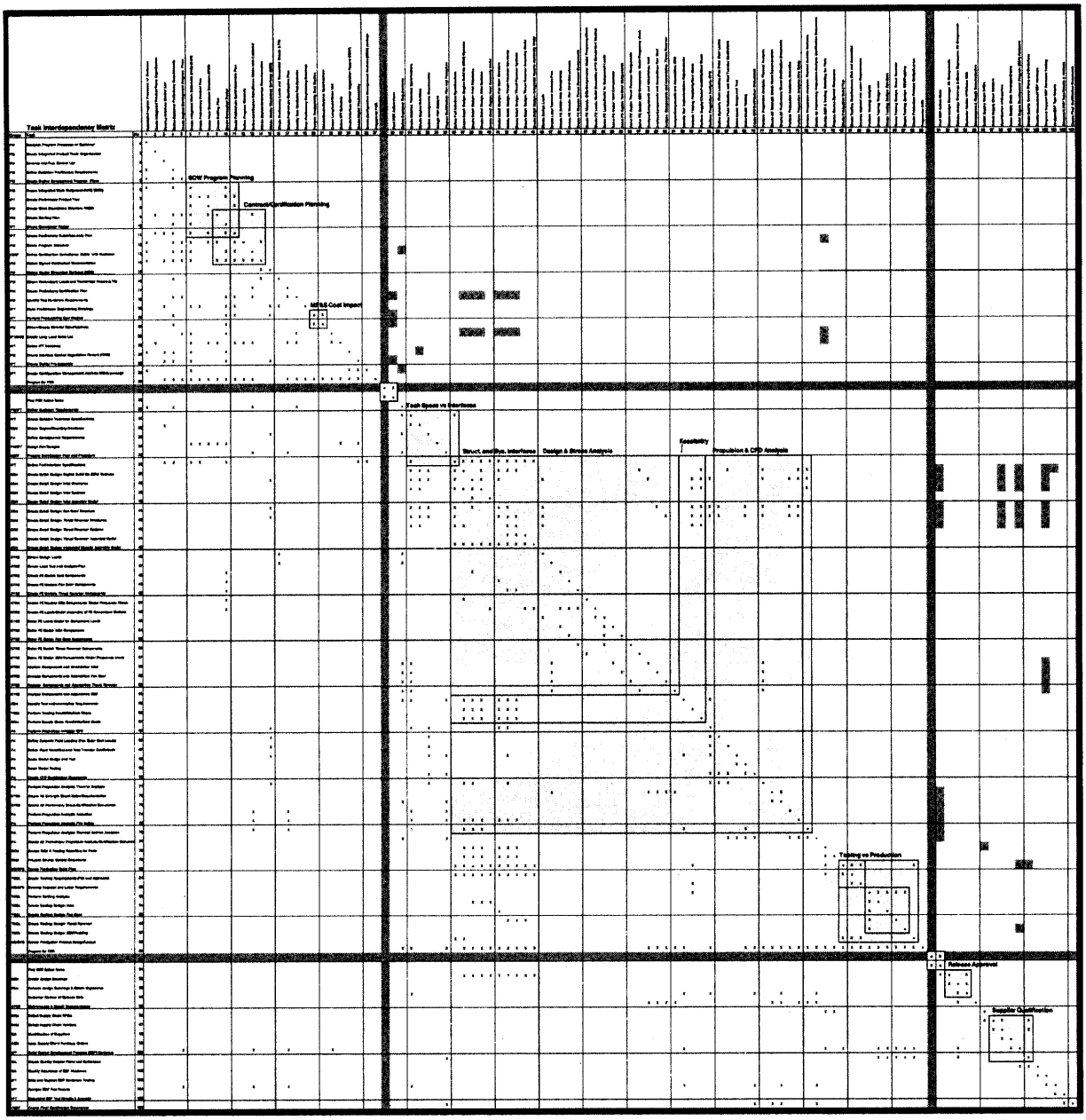
```
        trial_length = i-j;
        output_mat(1:trial_length,k:k+1)= input_mat(j:i-1,1:2);
      end
      j = i;
      k = k+2;
    end
    m=1;
    last_time = input_mat(i,1);
  end

%-------------------------------------------------------------------------
  function [output_mat] = time_point_stats(input_mat)

  output_mat(:,1)=input_mat(:,1);
  input_mat = input_mat(:,2:size(input_mat,2));

  trials = size(input_mat,2);

  output_mat(:,2)= mean(input_mat(:,1:trials),2);
  output_mat(:,3)= median(input_mat(:,1:trials),2);
  output_mat(:,4)= min(input_mat(:,1:trials),[],2);
  output_mat(:,5)= max(input_mat(:,1:trials),[],2);
  output_mat(:,6)= output_mat(:,2)-std(input_mat(:,1:trials),0,2);
  output_mat(:,7)= output_mat(:,2)+std(input_mat(:,1:trials),0,2);
  output_mat(:,8)= output_mat(:,2)+(2*std(input_mat(:,1:trials),0,2));

%-------------------------------------------------------------------------
 function [output_mat] = time_align(input_mat)

  %find max time
  time_max = 0;
  trials  = (size(input_mat,2)/2);
  for i = 1:trials
    trial_max = max(input_mat(:,((i*2)-1)));
    if trial_max > time_max
      time_max = trial_max;
    end
  end

  output_mat = zeros(time_max,trials+1);

  for i = 1:trials
    for j = 1:time_max+1
      time = j-1;
      output_mat(j,1)=time;
      input_time_col = ((i*2)-1);
      B = input_mat(:,input_time_col);
      C= B>time;
      if sum(C)~=0
        val=min(B(C));
      else
        val=max(B);
      end
      index=find(B==val);
      index = index(length(index));

      if time < max(input_mat(:,((i*2)-1)))
        output_mat(j,(i+1))=input_mat(index,i*2);
      else
        output_mat(j,(i+1))=0;
      end
    end
```

```
end

%-----------------------------------------------------------------------
function [output_mat]= trial_partition(input_mat)

 %find longest series to generate initial matrix
 last_time = 100000000;
 max_trial_length = 0;
 j=1;
 trial=0;

 last_time = 1000000000;
 for i = 1:length(input_mat)
   if input_mat(i,1)< last_time
      trial_length = i-j;
      j=i;
      if trial_length > max_trial_length
         max_trial_length = trial_length;
      end
      trial=trial+1;
   end
   last_time = input_mat(i,1);
 end

 output_mat = zeros(max_trial_length,2*trial);

 last_time = 100000000;
 j = 1;
 k = 1;
 m = 0;
 for i = 1:length(input_mat)
   if or(and(input_mat(i,1)< last_time,m==1),i==length(input_mat))

      if i ==length(input_mat)
         trial_length = i-j+1;
         output_mat(1:trial_length,k:k+1)= input_mat(j:i,1:2);
      else
         trial_length = i-j;
         output_mat(1:trial_length,k:k+1)= input_mat(j:i-1,1:2);
      end
      j = i;
      k = k+2;
   end
   m=1;
   last_time = input_mat(i,1);
 end
```

APPENDIX B - MATLAB Code for DSM Task "Network Distance" Measure

```
couplings_data =    xlsread('main_data.xls','Couplings');
dependency =        zeros(106);
for m = 1:size(couplings_data,1)
   dependency (couplings_data(m,1),couplings_data(m,3))  = 1;
end
mat = network_assess(dependency);
xlswrite('distance_output.xls',mat,'inoutdist');

%-------------------------------------------------------------------------
function [inoutdistance] = network_assess(inmat)

N = length(inmat);
dist1 = 1;
added = 1;
outmat = ones(N,N)*N;

%set diagonals equal to zero
for n = 1:N
   outmat(n,n)=0;
end

%set direct dependencies equal to one
for m = 1:N
   for n = 1:N
      if n~=m
         if inmat(m,n)==1;
            outmat(m,n)=1;
         end
      end
   end
end

%find other distances
while added > 0
   added = 0;
   for n = 1:N
      for m = 1:N
         if outmat(m,n)== dist1
            for i = 1:N
               if outmat(i,m)~=0
                  dist2=outmat(i,m);
                  tot = dist1 +dist2;
                  if tot <= outmat(i,n)
                     outmat(i,n)= tot;
                     added = added+1;
                  end
               end
            end
         end
      end
   end
   dist1 = dist1 + 1;
end

inoutdistance = zeros(N,3);
for j = 1:N
   inoutdistance(j,1)=j;
   inoutdistance(j,2)=sum(outmat(j,:))/(N*(N-1));
   inoutdistance(j,3)=sum(outmat(:,j))/(N*(N-1));
end
```

APPENDIX C - Binary DSM for Development Process of a Nacelle



BIBLIOGRAPHY

[1] Browning, Tyson R., "Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions" IEEE Transactions on Engineering Management, Vol. 48, No. 3, August 2001

[2] Browning, Tyson R., "Modeling and Analyzing Cost, Schedule, and Performance in Complex System Development" Massachusetts Institute of Technology, 1998

[3] Carroll, J.S., "Introduction to Organizational Analysis: The Three Lenses," Massachusetts Institute of Technology, June 2006

[4] Cassandras, C.G., S. Lafortune, "Introduction to Discrete Event Systems," Springer Science and Business Media, 2008

[5] Cho, Soo-Haeng, Steven D. Eppinger, "Product Development Process Modeling Using Advanced Simulation," DETC2001/DTM-21691 2001

[6] Eppinger, Steven D., Daniel E. Whitney, Robert P. Smith, and David A. Gebala, "A Model-Based Method for Organizing Tasks in Product Development," *Research in Engineering Design*, Vol. 6, No. 1, pp. 1-13, 1994.

[7] Lunsford, J. Lynn, "Jet Blues, Boeing Scrambles to Repair Problems with New Plane." Wall Street Journal, 7 December 2007, page A1.

[8] Naughton, A. B., "Aligning Tool Set Metrics for Operation in a Multi Technology High Mix Low Volume Manufacturing Environment", MIT 2005

[9] Pinkett, Randal, "Product Development Process Modeling and Analysis of Digital Wireless Telephones," Massachusetts Institute of Technology, 1998

[10] Robert, C., G. Casella, "Monte Carlo Statistical Methods", 2nd Edition, p.156, Springer 2004

[11] Smith, P., D. Reinertsen, *Developing Products in Half the Time*, 2nd Edition. Van Nostrand Reinhold, NY, 1995.

[12] Spirit AeroSystems, Incorporated, Annual Report, 2007

[13] Sosa, Manuel E.; Steven D. Eppinger; Craig M. Rowles, "A Network Approach to Define Modularity of Components in Complex Products," Transactions of the ASME, 1118 / Vol. 129, November 2007

[14] Steward, Donald V., "The Design Structure Matrix: A Method for Managing the Design of Complex Systems," *Institute of Electrical and Electronic Engineers (IEEE) Transactions on Engineering Management*, Vol. EM-28, No. 3, August 1981.

[15] Steward, Donald V., "*Systems Analysis and Management: Structure, Strategy and Design*," Petrocelli Books, Princeton, NJ, 1981.

[16] Tripathy, Anshuman; Steven D. Eppinger, "System Architecture Approach to Global Product Development" MIT 2007 Working Paper Number 4645-07

[17] Ulrich, Karl T., Steven D. Eppinger, "Product Design and Development," 4th Edition, McGraw-Hill, 2008

[18] Wayne, Leslie, Micheline Maynard, "New Boeing 787 Jetliner Faces Another Delay." New York Times, 5 December 2008.