PRODUCT DEVELOPMENT PROCESS MODELING AND ANALYSIS OF DIGITAL WIRELESS TELEPHONES

by

Randal D. Pinkett

B.S. Electrical Engineering, Rutgers University (1994)
M.S. Computer Science, University of Oxford (1996)

Submitted to the Department of Electrical Engineering and Computer Science and the Sloan School of Management
in partial fulfillment of the requirements for the degrees of

Master of Science in Electrical Engineering and Computer Science
and
Master of Business Administration

at the
Massachusetts Institute of Technology
June 1998

Signature of Author_____
Department of Electrical Engineering and Computer Science
Sloan School of Management
May 8, 1998

Certified by_____
Professor Steven D. Eppinger
Sloan School of Management
Thesis Supervisor

Certified by_____
Dr. Daniel E. Whitney
Department of Mechanical Engineering
Thesis Supervisor

Accepted by_____
Arthur C. Smith
Department of Electrical Engineering and Computer Science
Chairman, Committee on Graduate Studies

Accepted by_____
Lawrence S. Abeln
Sloan School of Management
Director, Master's Program

PRODUCT DEVELOPMENT PROCESS MODELING AND ANALYSIS OF DIGITAL WIRELESS TELEPHONES

by

Randal D. Pinkett

Submitted to the Department of Electrical Engineering and Computer Science and the
Sloan School of Management on May 8, 1998
in partial fulfillment of the requirements for the degrees of

Master of Science in Electrical Engineering and Computer Science
and
Master of Business Administration

ABSTRACT

When developing new products, there are a number of important considerations that must be examined closely in order to be successful, including the following: managing the interactions and interdependencies between functional teams, identifying the critical areas of engineering design, understanding the impact of iteration and rework on other design activities, and predicting the overall completion time. This thesis examines each of these areas in detail using a number of analytical tools to model the product development process of a digital wireless telephone, as a means toward analysis and improvement.

The Design Structure Matrix (DSM) is used to capture the structure of the product development process as a system. Using the DSM, Swapping is performed to represent the product development process from a variety of different perspectives, including by design task, by functional group, and by process flow. Coupling Analysis is used to analyze information flows and dependencies between design tasks, functional groups, and process flow phases, by quantifying their interrelationships. Partitioning is used to identify coupled sets of design activities. The Work Transformation Matrix (WTM) is used to model design iteration. Using the results of Partitioning and the WTM model, Controlling Features and Total Work Analysis (or Eigenstructure Analysis) is performed to identify closely related design activities that govern the rate and nature of convergence of the design effort. The Signal Flow Graph and the Reward Markov Chain are used to model hardware development, specifically to predict completion time of printed circuit boards (PCBs). The Aggregate-Code Model is used to model firmware development, specifically to predict completion time of large-scale projects. Software algorithms that implement many of these techniques are included in the form of MATLAB and Visual Basic code.

As a result of these analyses, a number of strategic recommendations are presented to the company that sponsored this research, as well as key learnings, and opportunities for future work.

Thesis Supervisors: David L. Tottle, Director
Steven D. Eppinger, Associate Professor
Daniel E. Whitney, Senior Research Scientist

ACKNOWLEDGEMENTS

"Rarely do we find people who willingly engage in hard, solid thinking.
There is an almost universal quest for easy answers and half-baked solutions.
Nothing pains some people more than having to think."

- Rev. Dr. Martin Luther King, Jr.

Like most research efforts, this thesis required hard, solid thinking, to find solutions to difficult problems. It would not have been possible without the assistance of numerous individuals and organizations.

I would like to thank my mother, Elizabeth Pinkett, my late father, Leslie Pinkett, my brother, Daniel Pinkett, and the rest of my family and friends for their continued love and support during my graduate education.

I would like to thank David Tottle, my supervisor at the sponsor company, for his able supervision of this research and editing of this thesis. Were it not for his persistence and diligence in arranging the assignment, this internship would not have been possible. Hopefully, my contributions to the sponsor company are in some way commensurate to my gratitude to him for agreeing to sponsor this project. I would like to thank Richard Albright, Ram Iyer, Anthony Dennis, and Sunil Lakhani, at the sponsor company, also without whom this internship would not have been possible. I would like to thank Steve Werden, at the sponsor company, for his able supervision of my efforts to model firmware, and Mark Van de Walle, at the sponsor company, for his generous assistance in my efforts to model hardware. I would like to thank Jack Gallagher, Ted Castellon, Peter Neary, Nick Iuzzolino, and all of the other managers and engineers at the sponsor company that helped in conducting this research. I would like to thank Dr. Peter Kroon, at the sponsor company, for his valuable advice and continued guidance of my endeavors. I would like to thank H.J. Carter and Selwyn Joseph, at the sponsor company, for their continued support.

I would like to thank Dr. Steven Eppinger and Dr. Daniel Whitney, my thesis supervisors at MIT, for their capable and insightful supervision of this research and editing of this thesis. Their research provided the foundation for this work, while their expertise proved invaluable toward setting the direction for my efforts. I would like to thank Dr. Robert Smith, at the University of Washington, for his generous assistance in my efforts to model design iteration.

I would like to thank Dr. Donald Rosenfield, the director of the Leaders for Manufacturing (LFM) Program, for helping to make this internship possible. I would like to thank Dr. Warren Seering, the director of the Center for Innovation in Product Development (CIPD), for his genuine concern for my best interests, and for enabling me to perform the assignment at the sponsor company. I would like to thank the staff of the LFM program for their assistance during my participation in the program. I would like to thank past LFM Fellows whose theses were extremely helpful in conducting this research. I would like to thank my colleagues at MIT in the LFM Class of 1998, School of Engineering Class of 1998, and Sloan School of Management Class of 1998, for their friendship and camaraderie during my participation in the program.

I would like to thank the National Science Foundation (NSF), the National Collegiate Athletic Association (NCAA), and the Cooperative Research Fellowship Program (CRFP), for their financial support of my course of study. It is truly appreciated.

I recognize that this formal learning experience is merely a supplement to the more valuable, informal learning experience known as life. I present this thesis as my own contribution to the engineering literature and the management literature, with the sincere hope that it will be of benefit to the learning experience of others.

"[I] certainly wasn't seeking any degree, the way a college confers a status symbol upon its students.
Not long ago an English writer telephoned me from London, asking questions.
One was, 'What's your alma mater?' I told him, 'Books.' "

- Malcolm X

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 Objective

With the rapid pace at which technology is evolving, high-tech firms are faced with incredible challenges when developing new products. To remain competitive, they must constantly monitor and stay abreast of changing market conditions, the competitive landscape, technological advancements, and the needs of their customers. Successful firms are differentiated from their competitors by their ability to respond to change quickly, while at the same time introduce high-quality products at the lowest possible cost. This growing need for companies to continuously improve their adaptability is further underscored by shrinking product life cycles and narrowing windows for time-to-market. Failure to meet an anticipated deadline can severely compromise a product's marketability and profitability, even causing some projects to be abandoned completely.

Given the significance of these issues, a number of companies are being forced to seriously examine all aspects of their product development processes including organization, division of labor, information transfer, technological capabilities, and allocation/collocation of resources. In doing so, it is their hope that they will identify ways to reduce development time while still achieving their targets for cost and quality. However, as is typically the case for large-scale projects within these industries, the product development process is extremely complex and often involves the contributions of hundreds of people over the course of several years.

To overcome some of these challenges, many companies utilize *concurrent engineering* [9], which involves multiple cross-functional teams working simultaneously on separate aspects of the overall development effort, particularly design and manufacturing. Concurrent engineering allows specialized groups to work in parallel and subsequently integrate their activities. Given the complex technical and organizational interactions between functional teams, integrating their efforts presents its own unique challenges.

To address the integration problem [18], several firms have adopted *rapid prototyping* [1], which is characterized by repeated "design-build-test" intervals, each resulting in a product prototype. Increasing levels of functionality are introduced with each successive iteration, leading up to the product's general availability. Rapid prototyping regularly synchronizes the efforts of the product development teams, while allowing cross-functional problems to be identified quickly and resolved throughout the development cycle.

Naturally, these two approaches lead to a number of important considerations that must be examined closely in order to be successful, including the following:

- *How do I manage the interactions and interdependencies between functional teams?*
- *What are the critical areas of engineering design?*
- *What is the impact of iteration and rework on other design activities?*
- *How can I predict the overall completion time?*

The objective of this thesis is address to each of these areas in detail using a number of analytical tools such as the Design Structure Matrix (DSM), Swapping, Coupling Analysis, Partitioning, the Work Transformation Matrix (WTM), the Signal Flow Graph, the Reward Markov Chain, the Aggregate-Code Model, and more, to model the product development process of a digital wireless telephone, as means toward analysis and improvement.

1.2 Background

This research was performed during a seven-month internship as part of collaborative work between a high-tech manufacturer of consumer electronics (henceforth referred to as "the sponsor company"), the MIT Leaders for Manufacturing (LFM) Program, and the MIT Center for Innovation in Product Development (CIPD). LFM is a partnership between the MIT School of Engineering, the MIT Sloan School of Management, and U.S. manufacturing firms, to discover and translate into teaching and practice the principles that produce world-class manufacturing and manufacturing leaders. CIPD is also a joint effort of the MIT School of Engineering, the MIT Sloan School of Management, and industrial partners, for research and education in product development. The research facility was dedicated to the design and manufacture of a specific product platform, digital wireless telephones (henceforth referred to as "the handset"), which was the focus of this research. The assignment was characterized by three unique attributes.

First, the internship started during the latter stages of development for the handset, and was completed just before the product's launch. This timing of the research provided an excellent opportunity to investigate product development in progress.

Second, all of the functional areas associated with the product were collocated within the research facility. This included marketing and sales, project management, research and development, and manufacturing. This approach to the allocation of resources made the facility an ideal candidate for examining product development across a number of different areas, because each functional group was easily accessible.

Third, the product development process was characterized by concurrent engineering in a rapid prototyping environment. A number of functional groups worked in parallel on segmented areas of design and manufacturing, while regularly scheduled builds and releases served as an integrating force across the various groups. New functions and new features were introduced with each prototype iteration throughout the product's evolution, leading up to its eventual completion. These techniques for design and manufacturing provided useful insights from both a technical and an organizational perspective.

In summary, these characteristics, coupled with the competitive nature of the industry, the powerful nature of the technology, and the complex nature of the handset, made the research environment extremely fertile ground for product development process modeling and analysis. In the following sections, we present a brief overview of the technologies used in wireless communications, an overview of the wireless industry, and an overview of the handset itself - a digital wireless telephone.

1.3 Technology Overview

The wireless industry embodies two major themes: Wireless Network Infrastructure and Wireless Communications Technology.

1.3.1 Wireless Network Infrastructure

Wireless networks provide a number of multimedia services to their end-users using radio technology, primary among them telephony - the ability to send and receive telephone calls. A simplified wireless network infrastructure consists of the following main elements [12]:

- **Mobile Station (MS)** - The MS is typically a stand-alone device (e.g. telephone or handset), but it can also be connected to other devices (e.g. computer). The MS provides the end-user with access to the network's services via the radio path.
- **Base Station (BS)** - The BS's are geographically dispersed sites that communicate with the MS's via the radio path. The BS's are stationary and provide coverage of specific geographic regions, while the MS's are free to move throughout the service area.
- **Mobile Switching Center (MSC)** - The MSC communicates with the BS's and exchanges messages with the wireline Public Switched Telephone Network (PSTN) and other wireless networks.

Figure 1.1 shows a pictorial representation of a simplified wireless network infrastructure.

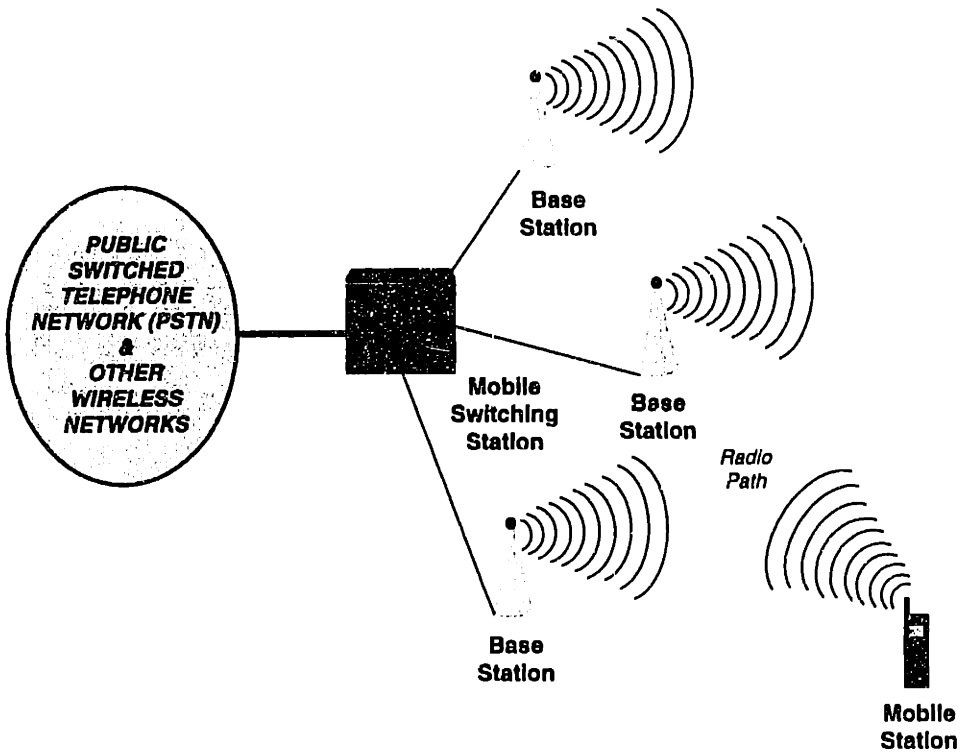


Figure 1.1: Simplified Wireless Network Infrastructure

1.3.2 Wireless Communications Technology

Over the last 15 to 20 years, wireless communications technology has experienced tremendous evolution. In the 1980's, the first generation of wireless networks were deployed at cellular frequencies (800 MHz). These networks used analog transmission modes, such as AMPS (Advanced Mobile Phone System), the U.S. standard. Toward the latter part of the 1980's many of these analog networks reached their capacities. To accommodate growing demand, the second generation of wireless networks were proposed using digital transmission modes. These networks were based on FDMA (Frequency-Division Multiple Access), where each user is assigned to a different frequency band, and TDMA (Time-Division Multiple Access), where each user is assigned to a different time slot.

In the early 1990's, TDMA emerged as the dominant digital transmission standard, and a number of TDMA networks were implemented as an adjunct the existing analog networks. The U.S. standard was based on TDMA technology, and the European standard, GSM (Global System of Mobile Communications), was also based on TDMA technology. Digital technologies offered a number of benefits over their analog counterparts including improved spectral efficiency, encryption, enhanced services (e.g. facsimile), better robustness, lower operations costs, reduced power consumption, and smaller/lighter handsets. However, looking toward the 21st century and beyond, there is still growing demand for even greater services than are presently available using TDMA. The wireless networks of the future will deliver voice, video, data, facsimile, and more, seamlessly to the end user, regardless of their location in the world. These services are commonly referred to as Personal Communications Services (PCS).

Toward this end, in 1993, wireless communications reached another milestone when the U.S. Congress allocated new spectrum at what were designated PCS frequencies (1.8 to 2.0 GHz). Within the same timeframe, a new transmission mode was proposed based on CDMA (Code-Division Multiple Access) technology, which offered additional benefits not afforded by TDMA. Traditional TDMA systems are narrowband systems, and therefore, dimension limited. TDMA systems cannot accommodate additional users once all of the time slots have been assigned. CDMA systems are based on a spread spectrum

convention whereby the number of users is only limited by the bandwidth and the amount of interference. Multiple conversations can be spread across a wide segment of broadcast spectrum by assigning one of 4.4 trillion unique codes to distinguish it from the other calls being transmitted simultaneously. CDMA results in increased capacity, higher voice quality, fewer dropped calls, better security and privacy, lower power consumption, reduced operating costs, and enhanced services. CDMA and PCS have ushered in the latest generation of wireless networks.

From a technological perspective, there are two main considerations facing the wireless industry - 1) What *frequency band* will you use for signal transmission? *(Cellular or PCS)* and 2) What *transmission mode (or standard)* will you use for spectral allocation? *(CDMA, TDMA, or GSM)*. Any combination of frequency bands and transmission modes is viable.

Table 1.1 shows retail sales of digital wireless telephones in 1997, and projected sales in 1998, for each permutation of frequency bands and transmission modes (Source: *Radio Communications Report*). Clearly, the highest levels of growth are projected for the PCS frequency band and the CDMA transmission mode. There are also a number of dual-band/dual-mode and dual-band/tri-mode wireless telephones currently being introduced or undergoing development. As standards continue to be established for these platforms, the market can expect to see a dramatic increase in the number of cross-technology handsets.

	Cellular Telephone Retail Sales *(Handsets)*			PCS Telephone Retail Sales *(Handsets)*		
Technology	1997	1998	Growth	1997	1998	Growth
CDMA	3M	6.9M	130%	0.7M	1.5M	114%
TDMA	1M	1.2M	20%	0.2M	0.5M	150%
GSM	36.4M	48.3M	32%	1.7M	2.3M	102%
TOTAL	**40.4M**	**56.4M**	**40%**	**2.6M**	**4.3M**	**65%**

Table 1.1: World Digital Wireless Telephone Retail Sales

1.4 Industry Overview

The wireless industry exists to provide communications services to end-users such as voice, video, data, and facsimile. Demographically, the typical customer is 40 years of age, with an average income above $60,000. In 1996, nearly 60% of wireless customers were female. The primary reason for purchasing a wireless telephone among customers is for safety purposes (45%), while the primary location for use is in the automobile (90%). Figure 1.2 shows the worldwide growth in wireless subscribers between 1992 and 1996, which experienced increases as high as 45% in the U.S. (Source: *U.S. Department of Commerce*). During this period, the average monthly service bill to the customer was $58.65, against 89 minutes of airtime. The number of wireless subscribers worldwide is expected to reach 200 million by the year 2000 (Source: *U.S. Department of Commerce*).

The wireless industry is divided into three main segments: 1) *Equipment Manufacturers* including mobile switching centers and base stations, 2) *Handset Manufacturers* including analog and digital wireless telephones, and 3) *Service Providers* that maintain wireless networks and purchase goods from both equipment manufacturers and handset manufacturers [13].

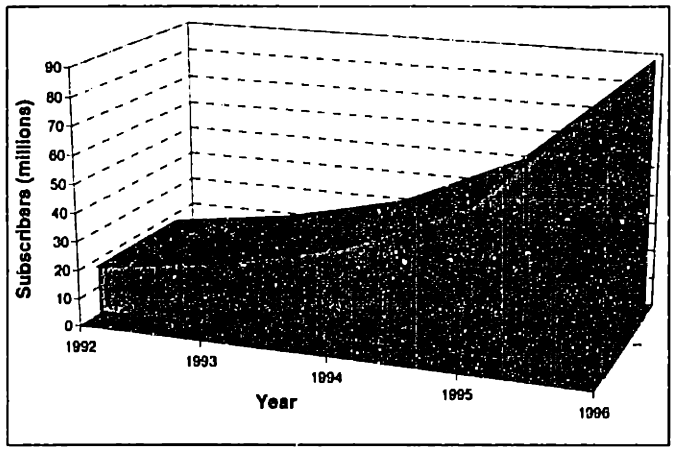


Figure 1.2: World Wireless Subscriber Growth

1.4.1 Equipment Manufacturers

The wireless equipment manufacturing industry is extremely concentrated in a few firms that essentially dominate the market. These firms include (market share in parenthesis): Lucent (37.2%), Ericsson (23.2), Northern Telecom or Nortel (19.2%), and Motorola (14.5%). These four companies alone represent more than 94% of the equipment manufacturing market. Figure 1.3 shows 1996 data of the world market share for cellular and PCS equipment manufacturers based on revenue (Source: *Dataquest*).

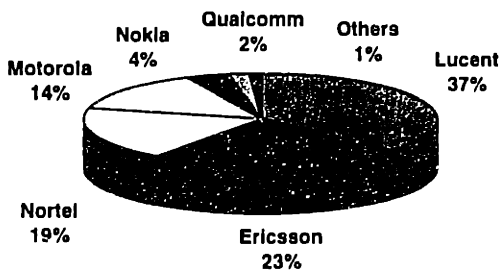


Figure 1.3: 1996 World Cellular and PCS Equipment Manufacturers Market Share

In 1996, the equipment manufacturing market grew 63% from $3.3 billion to $5.3 billion. This growth was largely fueled by the sales of PCS equipment which experienced a 428% increase from $530 million to $2.8 billion. Cellular equipment sales experienced negative growth during that same period. The typical capital investment for telecommunications network infrastructure equipment is $285,000 for a cell cite, which includes a radio tower, antennas, a building, radio channels, system controllers, and a backup power supply, and between $450,000 and $1.1M for a radio channel between cell sites. For TDMA, CDMA, and GSM, a single site can support 3,060, 9,600, and 4,800 subscribers respectively.

1.4.2 Handset Manufacturers

The wireless handset manufacturing industry is highly competitive in the digital wireless business, with more than thirteen players currently competing. This is due to the huge demand from service providers seeking to benefit from the cost and performance benefits of digital technologies. The primary players are (market share in parenthesis): Nokia (18%), Ericsson (18%), and Motorola (17%). Motorola and Nokia presently dominate the more mature, analog wireless business as well with 39% and 22% of the market respectively. Figure 1.4 shows 1997 data of the world market share for cellular and PCS handset manufacturers based on revenue (Source: *Dataquest*).

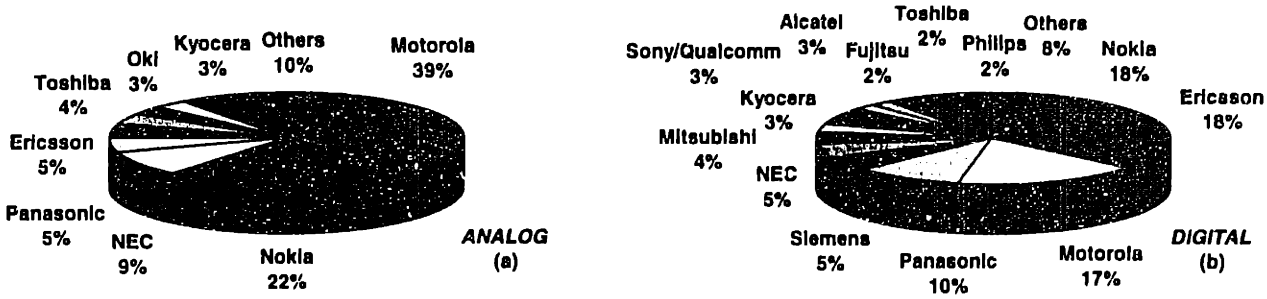


Figure 1.4: 1997 World Cellular and PCS Handset Manufacturers Market Share

The average wholesale price of terminals in 1997 was $431 (CDMA) and $315 (TDMA). These numbers are expected to drop to $162 and $87 respectively by the year 2000. Shipments of digital handsets was estimated at 1,653,000 in 1996. This is projected to reach 29,655,000 by the year 2000 - a compound annual growth rate of 105.8% (Source: *Dataquest*).

1.4.3 Service Providers

The wireless service provider industry is fairly consolidated at both cellular and PCS frequencies. This is partly due to the huge capital investment associated with obtaining frequency spectra and installing a wireless network. However, once a network is established there is extreme pressure to minimize the activation cost to the customer given the low marginal cost associated with adding new users to the system. In the U.S., the top five service providers at cellular frequencies using CDMA technology, own 90% of the market. At the same frequency, the top seven service providers using TDMA technology own 95% of the market. GSM has not been deployed in the U.S. at cellular frequencies. Figure 1.5 shows 1997 data of the U.S. market share for cellular service providers based on points-of-presence (POPS) (Source: *Mobile Phone News*).

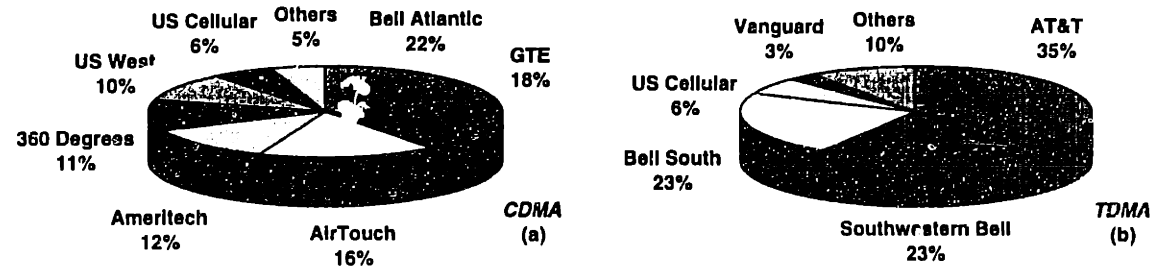


Figure 1.5: 1997 U.S. Cellular Service Providers Market Share

The average end-user price for a digital handset (CDMA and TDMA) in 1997 was $500. This is expected to drop to $333 by the year 2000. Presently, the cost of most handsets is subsidized by the service providers. The breakdown of cellular and PCS subscribers was estimated at 96% (cellular) and 4% (PCS) in 1996. This is projected at 65% (cellular) and 35% (PCS) by the year 2000 (Source: *Dataquest*).

Similarly, at PCS frequencies the top six (CDMA), three (TDMA), and eight (GSM) service providers own 93%, 99%, and 92% of their markets respectively. Figure 1.6 shows 1997 data of the U.S. market share for PCS service providers based on points-of-presence (POPS) (Source: *PCS Week*).

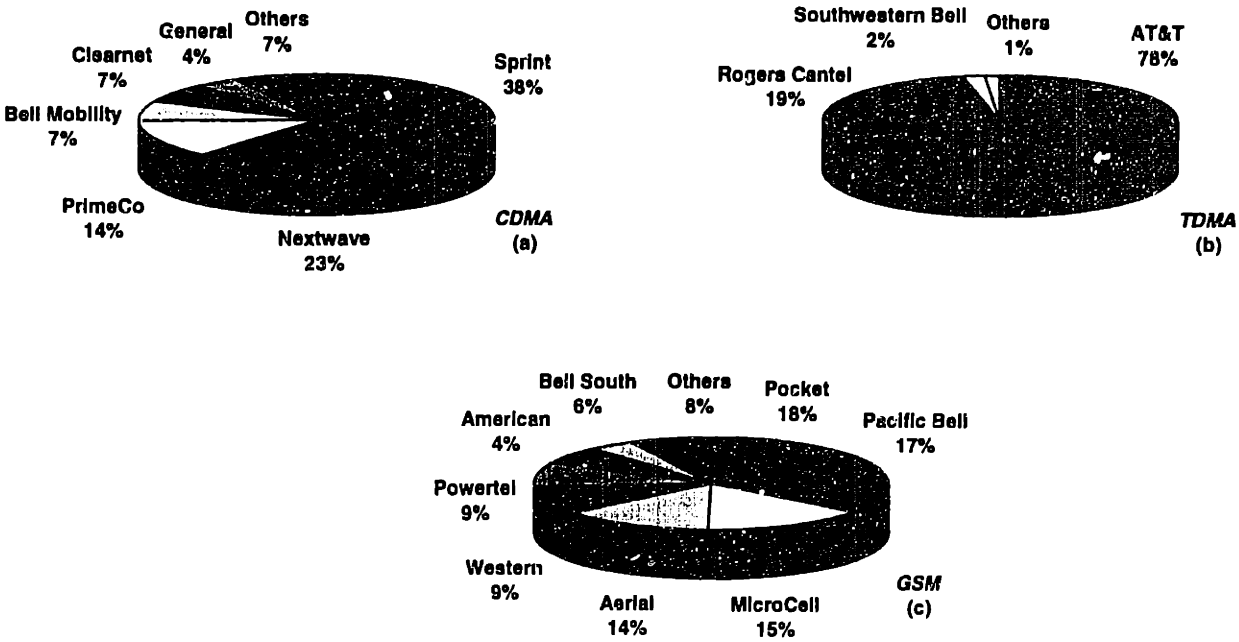


Figure 1.6: 1997 U.S. PCS Service Providers Market Share

1.5 Product Overview

Figure 1.7 shows a front-view of a digital wireless telephone. At a conceptual level, the elements of a digital wireless telephone can be divided into three primary categories: 1) *Hardware*, 2) *Firmware*, and 3) *Physical Components*. Figure 1.8 shows a simplified block diagram of a digital wireless telephone.

1.5.1 Hardware

Hardware denotes a printed circuit board (PCB) resident in the handset that includes analog circuitry and digital circuitry. The analog circuity, also referred to as Radio Frequency circuitry, includes a *Receiver* and a *Transmitter* that communicate directly with the base stations via an *Antennae*. The digital circuitry, also referred to as the Baseband circuitry, includes a *Microcontroller* with its associated *Memory* devices, a *Digital Signal Processor (DSP)*, a *Modem* (modulator/demodulator) that interfaces between the analog circuitry and the *DSP* by performing analog-to-digital and digital-to-analog conversion, and circuitry that interfaces between the *Microcontroller* and the *Display* and *Keypad/Buttons*, and the *DSP* and the *Audio/Acoustics* system.

1.5.2 Firmware

Firmware denotes binary/hexadecimal versions of software as read-only-memory (ROM) files resident in the handset that program the *Microcontroller* and the *Digital Signal Processor (DSP)*. The *Microcontroller* firmware controls the *User Interface* (*Display* and *Keypad/Buttons*), and establishes the handset's current state (e.g. initialization, synchronization, etc.) by controlling the *DSP*. The *DSP* firmware communicates directly with the analog circuitry by transmitting and receiving bits from the *Modem*, performs framing, error detection, and retransmission, and interfaces with the *Audio/Acoustics* system by processing the speech signal using a vocoder (voice coder/decoder).

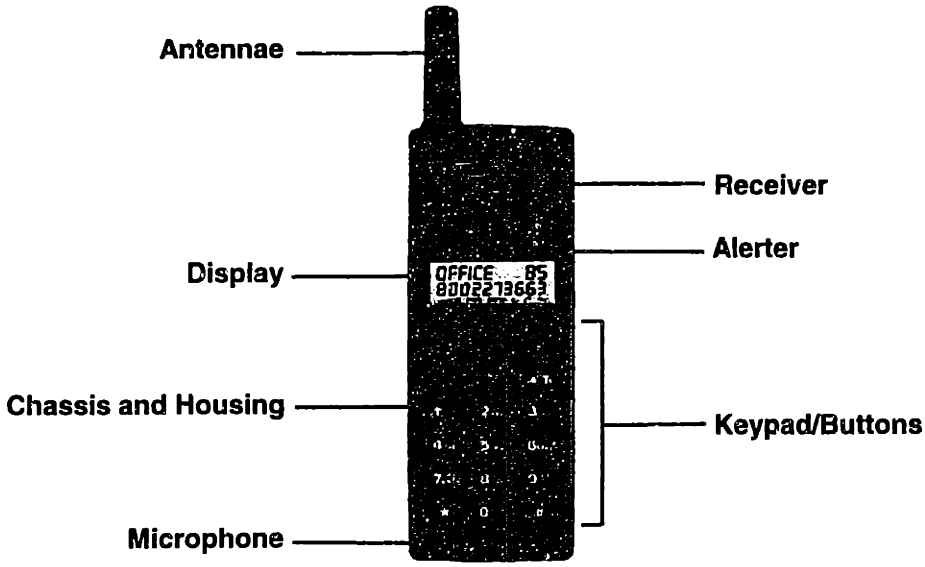


Figure 1.7: Front-View of a Digital Wireless Telephone

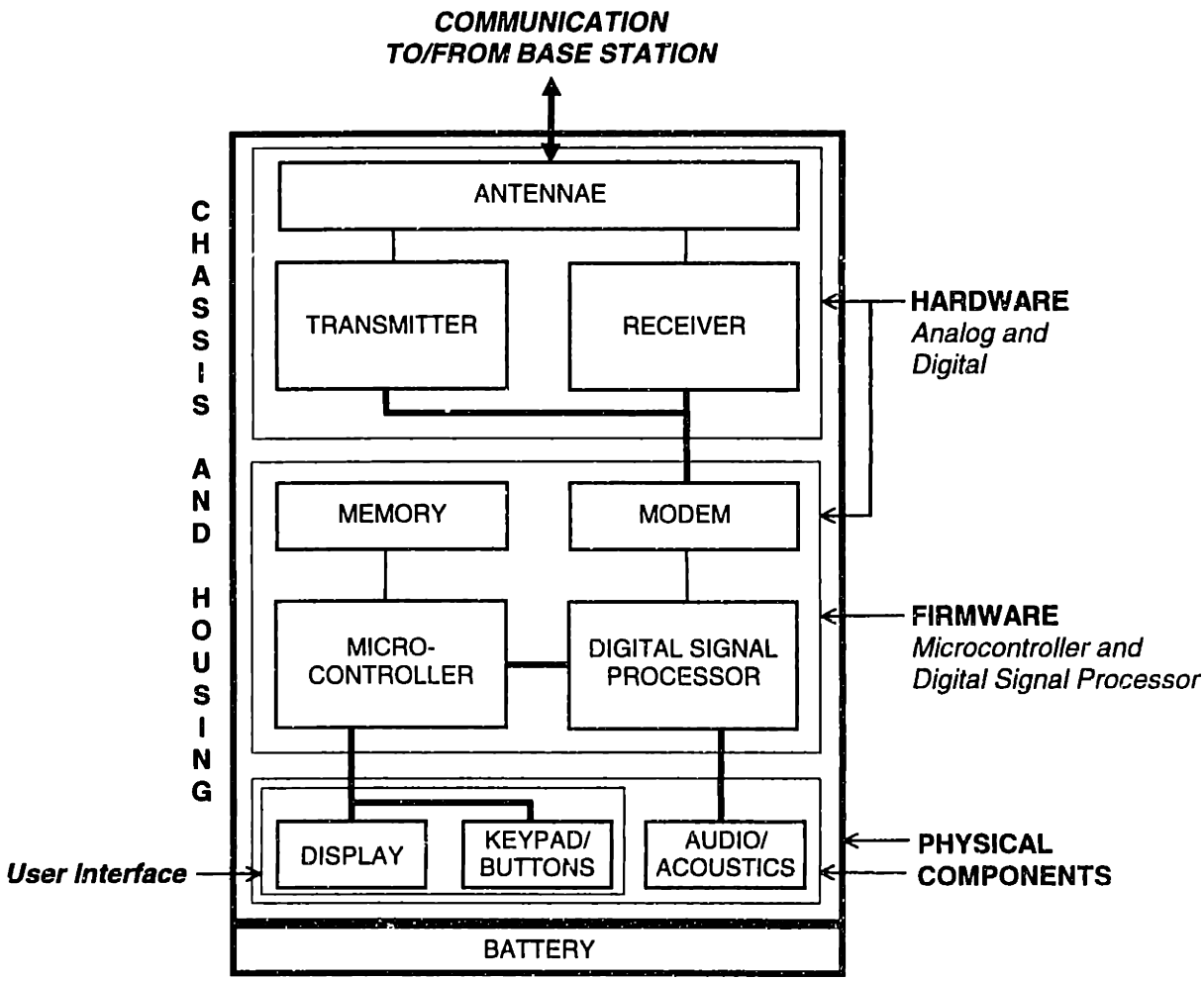


Figure 1.8: Simplified Block Diagram of a Digital Wireless Telephone

1.5.3 Physical Components

Physical Components denote the man-to-machine interface of the handset that includes the *Chassis and Housing*, *Display* monitor (typically a Liquid Crystal Display (LCD)), *Keypad/Buttons*, and the *Audio/Acoustics* system (microphone, receiver, and alerter). The *User Interface* components (*Display* and *Keypad/Buttons*) interface directly with the *Microcontroller*. The *Display* receives information (e.g. menu options) from the *Microcontroller* and the *Keypad/Buttons* send information (e.g. dialing selections) to the *Microcontroller*. The *Audio/Acoustics* components interface directly with the *DSP*. The microphone inputs an audio signal (the talker's or the listener's) to the *DSP*, the receiver outputs an audio signal (the listener's or the talker's) from the *DSP*, and the alerter outputs various tones (e.g. ringing) generated by the *DSP*.

1.6 Thesis Overview

This thesis is organized into six main chapters, each focusing on a particular aspect of the product development process at the sponsor company.

Chapter 2 provides an overview of the product development process at the design task level, functional group level, and the process flow level. This chapter includes an introduction to the *Design Structure Matrix (DSM)*, a modeling tool that represents the relationships among development activities in a matrix form, and presents the product development process within a DSM framework.

Chapter 3 analyzes information flow and dependencies between design tasks, between functional groups, and between the various phases of process flow. This chapter includes an introduction to *Swapping*, which uses the DSM to manipulate these interactions, and *Coupling Analysis*, which uses the DSM to quantify these interactions.

Chapter 4 analyzes the controlling features of engineering design iteration and overall total work. *Controlling Features and Total Work Analysis (or Eigenstructure Analysis)* predicts sets of closely related design activities that govern the rate and nature of convergence of the design effort. This chapter includes an introduction to *Partitioning*, which uses the DSM to identify the most tightly coupled subsets of design tasks, and the *Work Transformation Matrix (WTM)* model, which extends the DSM's capabilities as means toward understanding the impact of iteration and rework on other design activities.

Chapter 5 models hardware completion time for printed circuit boards (PCBs). This chapter includes an introduction to the *Signal Flow Graph*, a tool typically used in circuit and systems analysis in electrical engineering for modeling discrete event systems, and the *Reward Markov Chain*, a tool typically used in operations management, manufacturing, reliability, and telecommunications network and computer modeling to conduct queuing and system analysis for performance evaluation. Both tools are used to predict the completion time of a hardware development process that is modeled as sequential iteration. Transition probabilities determine feedback and iteration. Both models also offer the benefit of a sensitivity analysis with respect to the model parameters.

Chapter 6 models firmware completion time for large-scale software projects. This chapter includes an introduction to the *Aggregate-Code Model* that was developed specifically for the sponsor company, using aggregated code-related metrics such as the productivity, fault density, and mean time-to-repair faults as inputs. The model predicts the completion time of a firmware development process that is modeled by segmenting the code into various categories at an appropriate level of abstraction. This model also offers the benefit of a sensitivity analysis with respect to the model parameters.

Chapter 7 presents a set of strategic recommendations to the sponsor company, key learnings that have been drawn as a result of this research, and possibilities for future work.

1.6.1 Notes

Names of the design tasks and functional groups have been disguised at the request of the sponsor company. Data have also been disguised at the request of the sponsor company. The term "standard" will generically refer to any industry standard for digital wireless telephones (e.g. standards issued by the American National Standards Institute (ANSI), Electronic Industry Association (EIA), Telecommunications Industry Association (TIA), International Standard Organization (ISO), or Alliance for Telecommunications Industry Solutions (ATIS)).

Throughout this document, many of the analytical and mathematical techniques will be presented using simple introductory examples. For subsequent uses of the same technique, only the final results (e.g. matrices, expressions, etc.) will be presented. Software tools that implement the algorithms used to perform these examples are included in the Appendix in the form of MATLAB 4.0 code [16, 17] and Microsoft Excel 5.0 Visual Basic code [19, 20]. Not included are additional software tools with expanded functionality that were developed using MATLAB and Visual Basic and served as a graphical front-end and user interface.

CHAPTER 2: PRODUCT DEVELOPMENT PROCESS OVERVIEW

2.1 Systems and Structure

A system is a collection of parts and relations between parts such that the behavior of the whole is a function not only of the behavior of the parts, but also the relations among them. The structure of the system shows where the parts are connected, or related [39]. Figure 2.1 shows the structure of a system that is comprised of six parts (letters A through F).

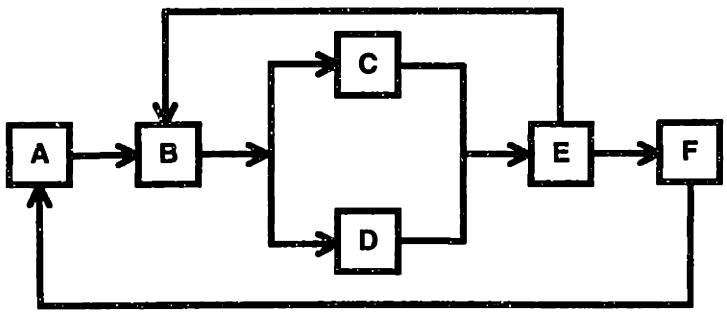


Figure 2.1: Structure of a System

This diagram could represent any number of systems such as a gathering of individuals that are exchanging information, a collection of routes that are offered by an airline, or a network of suppliers that comprise a value chain.

2.2 Product Development Process Modeling and Analysis

Products are systems. When developing new products, one must understand the structure of the product as a system. For example, a computer mouse has various parts such as buttons, the housing, and a track ball. A design team must be able to look at each part individually as well as within the context of the entire system. It is only by recognizing how the size of the buttons can affect the housing, and how the housing can affect the location of the track ball, that one can create a mouse for optimal performance.

Similarly, product development processes are systems. When developing new products, one must also understand the structure of the product development process as a system. Referring to the previous example, the human and technical resources (the parts) necessary to design and manufacture the mouse must all work together individually and collectively (the system). It is only by recognizing how the industrial designer must interface with the mechanical engineer, and how the mechanical engineer must interface with the manufacturing engineering, that one can create a mouse for optimal performance.

Typically, when we cannot break a real system into parts, we construct a model of the system that we can break into parts. A model is an approximation of the real system, such that an understanding of the model can be translated into an understanding of the real system. Models can be particularly useful when examining complex systems. Just as models can be created for products, models can also be created for product development processes [42]. By modeling and analyzing the product development process we hope to achieve greater insight that can hopefully be leveraged for subsequent improvement.

In this chapter, we present an overview of the product development process for the digital wireless telephone at the sponsor company.

2.3 Design Structure Matrix (DSM)

The Design Structure Matrix (DSM) is a product development process modeling tool that represents the relationships among development activities in a matrix form [38]. The DSM was first introduced by Steward and captures coupling and dependence between the design tasks of a project [39]. 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. As an example, Figure 2.2 shows a 6 x 6 binary DSM representation of the system shown in Figure 2.1. Each part (letters A through F) represents a task that is performed during the product development process.

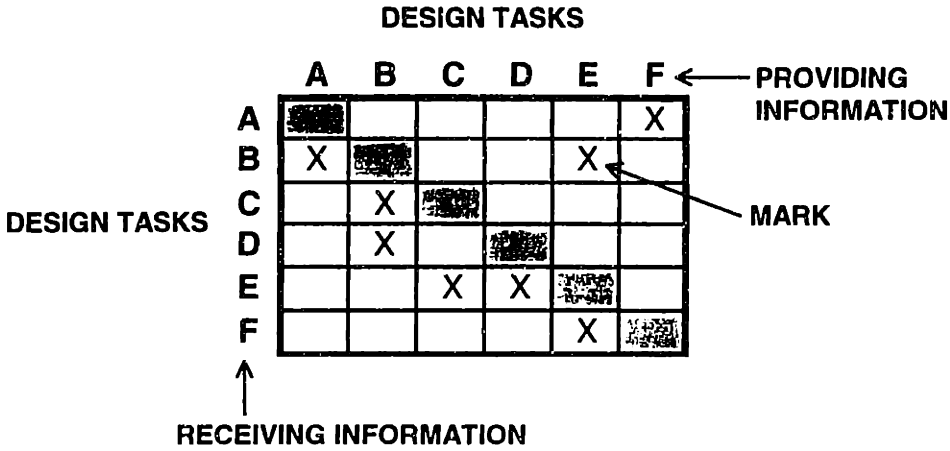


Figure 2.2: Binary Design Structure Matrix

The mark located in row three, column two, 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 are dependent on it. For example, looking down column five we see that Tasks B and F are dependent on Task E to be completed. Looking across a task's row, one can easily determine those tasks that it is dependent on. For example, looking across row five we see that Task E is dependent on Tasks C and 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 feedforward (information transferred from earlier tasks to later tasks). In the following chapter we will expand on this concept in the context of *Partitioning*, a method which attempts to find a lower triangular ordering for the tasks in the DSM. Such an ordering represents an uncoupled design problem (no feedback) where each task receives all of its required information from its predecessors.

An extension to Steward's work was introduced by Eppinger, Whitney, Smith, and Gebala [10] 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 marks measure task duration (or some other metric that characterizes an intra-task relationship). This *Numerical Design Structure Matrix* captures task interrelationships at a much deeper level than its binary counterpart, in addition to capturing completion time. As an example, Figure 2.3 shows a 6 x 6 numerical DSM representation of the binary DSM shown in Figure 2.2.

The number 2.1 located in row two, column four, denotes the relative strength of dependence by Task B on Task D. The number 7 located in row three, column three, denotes a duration of 7 days for Task C to be completed.

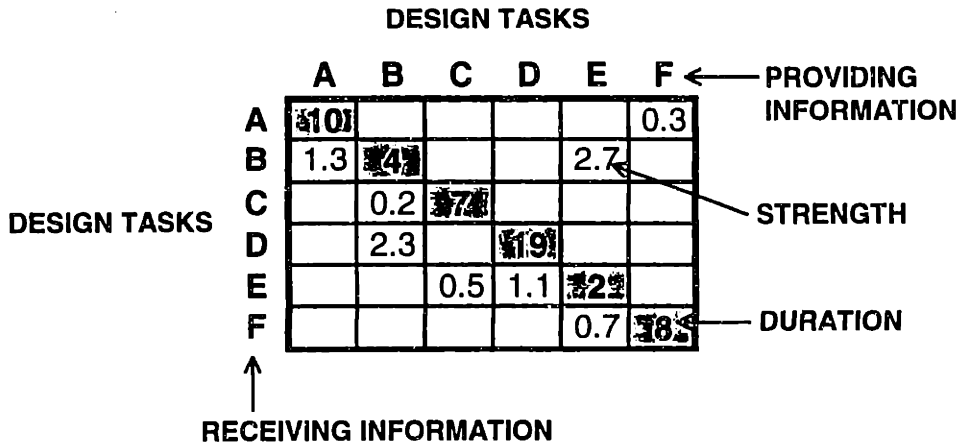


Figure 2.3: Numerical Design Structure Matrix

As evidenced from these simple examples, the DSM is an effective tools for capturing the structure of a system both concisely and compactly.

2.4 Constructing the DSM

The four steps to constructing the DSM are as follows:

- Select the areas of the product development process to be examined.
- Identify development activities at a sufficient level of abstraction, and define development activities according to an appropriate rule of classification *(Task-Level DSM, Parametric-Level DSM, or Hybrid DSM)*.
- Select an inter-task metric for the off-diagonal positions *(Sequential Iteration Model* or *Parallel Iteration Model)*, and select an intra-task metric for the on-diagonal positions.
- Select interviewees and collect data.

To construct a DSM, the areas of the product development process that are going to be examined must be selected. This is the first and perhaps most important step in constructing a DSM because it affects all subsequent steps. Examining an area that is too broad or too limited in scope can severely inhibit the ability of the DSM to provide valuable insight.

Second, once these areas have been selected, development activities must be: 1) identified at a sufficient level of abstraction, and 2) defined according to an appropriate rule of classification. Selecting an appropriate level of abstraction is important because one that is too low can prove cumbersome during construction and collection of data. Similarly, a level of abstraction that is too high can prove worthless when performing analysis and seeking real insight. Selecting an appropriate rule of classification is equally important, and refers to the way in which development activities are defined. Typically, they are categorized as either design tasks *(Task-Level DSM)* or design parameters *(Parametric-Level DSM)*. A DSM can also combine both design tasks and design parameters *(Hybrid DSM)* [10]. Design tasks are best described as discrete physical activities performed by people. Examples of entries in a task-level DSM include documenting customer requirements and ordering parts for a product prototype. Design parameters are best described as product specifications or attributes. Examples of entries in a parametric-level DSM include the frequency response of an acoustical system or the clearance of the housing for a component. Henceforth for simplicity, we will refer to both design tasks and design parameters as design tasks, unless explicitly stated otherwise. Once identified and defined, design tasks are placed along the left column and the top row of the matrix. As mentioned earlier, along the left column these tasks denote design activities that receive information. Along the top row, these tasks denote design activities that provide information.

Third, an inter-task metric is selected (e.g. task precedence or coupling strength) for the off-diagonal numerical values, and an intra-task metric can be selected (e.g. duration or cost) for the on-diagonal numerical values [8, 18, 25]. If the intent is to build a DSM model that captures iteration, one must distinguish between the kind of iteration they seek to capture - sequential iteration or parallel iteration. In the *Sequential Iteration Model* (which uses the Reward Markov Chain and is described in greater detail in Chapter 5), the off-diagonal numerical values indicate the probability that one additional iteration will be necessary if the interdependent tasks are performed in the specified order [10, 36]. In the *Parallel Iteration Model* (which uses the Work Transformation Matrix and is described in greater detail in Chapter 3), the off-diagonal values measure what portion of information produced during the first iteration would need to be changed during the second iteration [10, 35]. Using these models, a number of analytical methods can be applied to predict iteration time or identify strongly coupled sets of tasks. A clear understanding of the intended use for the data can help guide in the decision regarding which iteration model is best suited for a given purpose. Such an assessment will also govern how the questionnaires are worded.

Fourth, interviewees are identified and data are collected via questionnaires. Typically, the interviewees are participants in the product development process (e.g. engineers or managers). Because the inter-task metrics are usually subjective (e.g. rework amounts, iteration probabilities), it is important to identify individuals that are deemed capable of providing data that accurately depicts reality. We should note that previous research efforts have also stressed the confusion that can be easily encountered between the "currently followed", "should-be followed", and the "will-be followed" product development processes when gathering DSM data [34]. This must be clearly distinguished and articulated at the onset.

Finally, once the data set is collected, intra-task metrics are placed in the on-diagonal positions of the matrix and inter-task metrics are placed in the off-diagonal positions of the matrix. In the following sections we will demonstrate how the DSM was used to model the product development process at the sponsor company. This was achieved by first identifying the various functional groupings and process flow phases that governed design activity, and then proceeding to construct the DSM and collect data.

2.5 Functional Groups

Products have an inherent technical structure that in some way should be matched with respect to the organization of resources. In practice, without a detailed understanding of these interactions in advance, it is difficult to know what represents the optimum. Firms typically organize design activities into a number of functional areas. Each functional area is then focused on a particular product "chunk". This is particularly useful for large-scale projects, complex products, and design efforts that involve individuals or teams that are geographically dispersed. One of the objectives of such an approach is to minimize the interactions across functional groups, and maximize the interactions within functional groups, resulting in better coordination and reduced time-to-market.

Product development at the research company was organized by the fifteen functional groups listed in Table 2.1 (abbreviations in parenthesis).

Functional Group	Functional Group
Product Management (PM)	Digital Signal Processor Design (DSP)
System Engineering (SE)	Microcontroller Design (MC)
System Architecture (SA)	System Integration (SI)
Human Factors (HF)	System Test (ST)
Supply Chain Management (SCM)	Reliability (REL)
Industrial Design (ID)	Field Test (FT)
Analog Design (AD)	Manufacturing (MFG)
Digital Design (DD)	

Table 2.1: List of Functional Groups

Each of these functional groups worked in parallel (concurrent engineering) and was responsible for a specific area related to the product, ranging from defining requirements to final manufacturing. Note that

we identified the design task "Field Test" as a functional group, although it was not a formal functional group. The following is a brief description of each functional group (for a detailed understanding of the product's technical structure, the reader is referred to Section 1.5):

Product Management - Manages the day-to-day profitability of the business including marketing, sales, and service, and generates the Customer Requirements (CR) document for product/handset platforms.

System Engineering - Translates customer/market needs (from the CR) and operational requirements (from industry standard documents) into engineering requirements for the handset by generating the Technical Requirements (TR) document. System Engineering also monitors and interprets industry standard documents.

System Architecture - Translates engineering requirements (from the TR) into requirements for the design groups and evaluates proposed techniques for feasibility.

Human Factors - Designs and tests the product's user interface, ergonomics, and other functional and operational aspects such as tones, call flow, menu structure, error messages, button actions, button labels, display characteristics, etc.

Supply Chain Management - Manages additions to/deletions from the qualified parts list, validates/updates the qualified suppliers list, and works with procurement to obtain parts from vendors for product prototypes and volume production units. Supply Chain Management works closely with the Analog Design, Digital Design, and Manufacturing groups, to obtain parts for these builds.

Industrial Design - Defines the physical dimensions of the handset and ensures that the industrial design of the product meets or exceeds customer needs, both functionally and aesthetically. Industrial Design works closely with the Analog Design, Digital Design, and Manufacturing groups to provide the dimensional specifications of the handset.

Analog Design - Designs the analog printed circuit board (PCB) including the antennae, receiver, transmitter, transfer oscillator, and power control. Analog Design works closely with the Industrial Design group to receive the dimensions of the handset, the Supply Chain Management group to secure parts, and the Manufacturing group to build the PCB.

Digital Design - Designs the digital printed circuit board (PCB) including the microcontroller, digital signal processor (DSP), keypad, memory facilities for both processors, and modem. Digital Design works closely with the Industrial Design group to receive the dimensions of the handset, the Supply Chain Management group to secure parts, and the Manufacturing group to build the PCB.

Digital Signal Processor Design - Designs and tests the DSP firmware including the vocoder, layer 1 (physical layer) processes (transmit/receive bits), and layer 2 (data link layer) processes (perform framing, error detection, and retransmission). Digital Signal Processor Design works closely with the Microcontroller Design group, particularly given the fact that the microcontroller controls the function of the DSP.

Microcontroller Design - Designs and tests the microcontroller firmware including the Type 1 software (communication with the base stations), user interface (display and keypad/buttons), and machine-to-machine interface (data I/O port that can connect to a PC). The Microcontroller Design group works closely with the Digital Signal Processor Design group, particularly given the fact that the microcontroller controls the function of the DSP.

System Integration - Integrates the hardware and software components from the development organizations and/or outside vendors, and tests for operability and performance. System Integration works closely with the hardware and firmware design groups (Analog Design, Digital Design, Digital Signal Processor Design, and Microcontroller Design) to test essential features and ensure the overall quality of the product prior to delivering the handset to the System Test group.

System Test - Evaluates and documents the handset's overall performance and compliance with industry standards at the system level. System Test receives units from the System Integration group. System-level testing is considered the final phase of each prototype iteration. System Test works closely with the System Integration group, who performs testing of the handset at a level of abstraction below that of System Test. System Test essentially performs testing of the handset as a "black-box" (using external interfaces) focusing solely on inputs and outputs.

Reliability - Provides a means to plan for, design in, assess, predict, monitor, control and improve handset reliability throughout its life cycle. Reliability develops product hardware and firmware reliability requirements, certifies vendors/components for reliability qualification, and qualifies product prototypes and volume production units. Reliability works closely with the Industrial Design and Supply Chain Management groups to evaluate the handset at the component level and the unit level.

Field Test - Performs evaluation of the handset in the field. Field Test receives units from the System Test group. Field testing is performed in a real-world environment.

Manufacturing - Conducts Design for Manufacturing/Assembly/Testing (DFX) reviews, manufacturers the analog/digital printed circuit board (PCB), and performs final assembly of the unit. Manufacturing also provides a method for implementing new products into an existing manufacturing environment that supports rapid prototyping and high volume manufacturing. Manufacturing works closely with the hardware design groups (Analog Design and Digital Design), and the Industrial Design and Supply Chain Management groups to perform manufacturing and assembly.

2.6 Process Flow

The product overview helped us to understand the product as a system, and its underlying structure, from a technical perspective. Functional groupings helped us to understand the product development process as a system, and its underlying structure, from an organizational perspective. Finally, the process flow helped us to understand the product development process as a system, and its underlying structure, from a temporal perspective. The process flow diagram was also extremely useful in clearly identifying the relevant areas of the product development process for investigation.

The process flow for the product development process at the sponsor company could be segmented into four distinct phases: 1) *Requirements*, 2) *Design*, 3) *Integration*, and 4) *Test*, the latter three phases constituting an iterative loop (rapid prototyping iterations). This research focused specifically on these three phases, because they represented a significant portion of the product development process over time. Figure 2.4 shows the process flow diagram of the product development process.

The following is a brief description of each process flow phase:

Requirements - During the Requirements phase, customer requirements are identified by Product Management and translated to technical specifications by System Engineering. These technical specifications are translated to design specifications by System Architecture, which are finally translated to detailed design specifications by each of the design groups (Analog Design, Digital Design, Digital Signal Processor Design, and Microcontroller Design).

Design - During the Design phase, functional groups related to hardware (Analog Design, Digital Design, Supply Chain Management, Industrial Design, Reliability, and Manufacturing) and firmware (Digital Signal Processor Design and Microcontroller Design) work in parallel within their respective areas to complete regularly and separately scheduled hardware builds and firmware releases, respectively. Increasing functionality is introduced with each successive prototype iteration, leading up to the product's eventual completion. System Integration is more closely aligned with the firmware design groups because many of the functions and features are realized in firmware. Also conducted during this phase are the procurement of parts by Supply Chain Management, sub-unit level tests by Industrial Design and Reliability (e.g. a specific component), and the development of test cases by System Integration. Note that as a strategic resolve, the sponsor company intentionally decoupled hardware and firmware activities to minimize the interactions between the hardware and firmware design groups. It was believed that such interdependencies could severely hinder the speed of development. Hardware builds and firmware releases were therefore scheduled according to different timetables. Consequently, hardware and firmware design could be considered relatively independent activities (to the extent that this strategy was successful), as indicated in Figure 2.4.

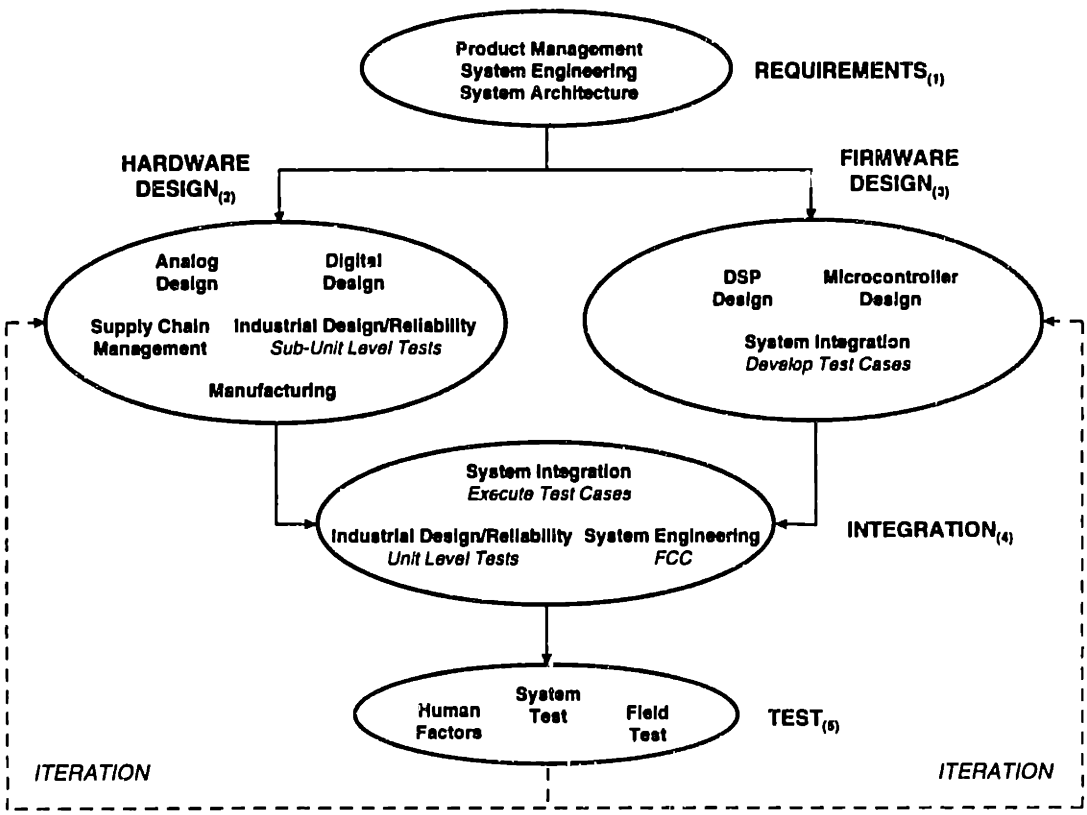


Figure 2.4: Process Flow Diagram of the Product Development Process

Integration - During the Integration phase, System Integration integrates the hardware and firmware by executing the test cases that were developed during the Design phase. These test cases evaluate the handset at a level of abstraction below that of System Test and deal primarily with features that cut across functional areas. Also conducted during this phase are unit level tests by Industrial Design and Reliability (e.g. an environmental test of the handset), and regulatory testing by System Engineering. Once System Integration is satisfied with the performance of the handset, it is handed over to System Test for final evaluation.

Test - During the Test phase, System Test performs "black-box" testing of the handset according to established internal (e.g. requirements documents) and external (e.g. industry standards) specifications. Also conducted during this phase are various man-machine interface related tests by Human Factors, and testing in the field by Field Test. By the time this phase has completed, the next iteration of the Design phase has already begun. In fact, when looking at a level of abstraction below the large "Design-Integration-Test" loop, there are a number of smaller loops conducted within each respective functional area. The larger loop is the result of the introduction of new functionality associated with rapid prototyping. The smaller loops are the result of either their own local processes and tests associated with concurrent engineering, or more global feedback received from System Integration or System Test.

2.7 Design Tasks

Design tasks represent the basic elements of the DSM. With the functional groups identified and the process flow clearly understood, interviews were conducted with the technical managers of each functional group to generate a list of design tasks that were directly relevant to the handset. A particular emphasis was placed on activities conducted between each successive hardware build and firmware release (rapid prototyping iterations - the Design, Integration, and Test phases). Brief follow-up interviews were then conducted with the same technical managers to confirm the list's accuracy and completeness. The final list of 114 design tasks is shown in Table 2.2 (each task is preceded by its abbreviated functional group, referenced in Table 2.1).

As mentioned earlier, this research focused specifically on the Design, Integration, and Test phases. In limiting the study to these activities solely, a number of development activities were either taken for granted, combined into aggregate tasks, or completely ignored. For example, prior to the requirements phase, a number steps must be performed to gain project approval. These tasks were taken for granted. During the requirements phase, a number of steps must be performed leading up to the release of the Customer Requirements (CR) document. All of these tasks were aggregated into the single task "Customer Requirements". Once the product is ready for general availability, there are a number of associated tasks including field support and customer support. These tasks were ignored.

Lastly, although each of these tasks are associated with a specific functional group, many can be considered cross-functional activities ranging from documenting the customer requirements to performing testing in the field. Functional group associations are primarily indicative of ownership. The following is a brief description of each design task (categorized by functional group):

Product Management

Customer Requirements - Document that describes the product's functionality as required by the customer.

System Engineering

Technical Requirements - Document that describes the product's functionality at a technical level and identifies the optimal approach for development.

Handset Requirements - Document that describes detailed, product-specific requirements for hardware/firmware developers, testers, and Product Management.

Standard Features - External industry standards that must be specified and govern interoperability with the base station.

Standard Type 1 Messages - Specifications of which Type 1 messages will be supported on the handset from an industry standard.

Standard Type 2 Messages - Specifications of which Type 2 messages will be supported on the handset from an industry standard.

Standard Type 3 Messages - Specifications of which Type 3 messages will be supported on the handset from an industry standard.

Regulatory Emission Test - Regulatory compliance emissions test.

Regulatory Scan - Regulatory compliance scan.

Regulatory Compliance Certification - Regulatory compliance certification.

No.	Design Task	No.	Design Task
1	PM: Customer Requirements	57	MC: Standard Type 1 Changes
2	SE: Technical Requirements	58	MC: NAM Programming Changes
3	SE: Handset Requirements	59	MC: Standard System Determination
4	SE: Standard Features	60	MC: Type 3 Service Provisioning
5	SE: Standard Type 1 Messages	61	MC: Type 2 Messages
6	SE: Standard Type 2 Messages	62	MC: User Interface
7	SE: Standard Type 3 Messages	63	MC: CNIP
8	SE: Regulatory Emission Test	64	MC: Authentication
9	SE: Regulatory Scan	65	MC: Roaming
10	SE: Regulatory Compliance Certification	66	MC: Voice Mail Notification
11	SA: Architecture Requirements	67	MC: Programming Lock
12	HF: Voice Quality Test	68	MC: Type 3 Roaming List
13	HF: User Interface Test	69	MC: Integrity
14	HF: Audibles Test	70	MC: Sleep
15	HF: Ergonomic Test	71	MC: Type 4 Protocol
16	HF: Field Trial Test	72	MC: Boot Block Support
17	SCM: Order/Secure Standard Materials	73	MC: MC Firmware Release
18	SCM: Order/Secure Custom Parts	74	SI: Develop Accessories Test Cases
19	ID: Tool Parts Available	75	SI: Develop Type 1A Test Cases
20	ID: Component Plastics Test	76	SI: Develop Type 1B Test Cases
21	ID: Component LCD Test	77	SI: Develop Type 4 Test Cases
22	ID: Component Chassis Test	78	SI: Develop Integrity Test Cases
23	ID: Component Acoustics Test	79	SI: Develop Type 3 Test Cases
24	ID: Unit Drop Test	80	SI: Develop Support Tools Test Cases
25	ID: Unit Reliability/Environmental Test	81	SI: Develop User Interface Test Cases
26	ID: Unit Acoustics Test	82	SI: Integrate System Components
27	ID: Industrial Design Release	83	SI: Execute Accessories Test Cases
28	AD: Draw/Update Schematic	84	SI: Execute Type 1A Test Cases
29	AD: Enter Schematic (ECAD)	85	SI: Execute Type 1B Test Cases
30	AD: Component Placement	86	SI: Execute Type 4 Test Cases
31	AD: Routing/Via/Ground/Thermal Relief	87	SI: Execute Integrity Test Cases
32	AD: Design Rule Check	88	SI: Execute Type 3 Test Cases
33	AD: Gen Art/Prep Docs/Format ECAD	89	SI: Execute Support Tools Test Cases
34	AD: Solder Stencil (send/receive supplier)	90	SI: Execute User Interface Test Cases
35	AD: Circuit Board (send/receive supplier)	91	SI: Demo Prep
36	AD: Parts (send/receive BOM to SCM)	92	SI: Field Test Support
37	AD: Fabricate PCB	93	SI: System Integration Release
38	AD: Unit Test	94	ST: Layer 1 Test
39	AD: AD Engineering Sample	95	ST: Type 1 Test
40	DD: Draw/Update Schematic	96	ST: Interoperability Test
41	DD: Enter Schematic (ECAD)	97	ST: User Interface Test
42	DD: Component Placement	98	ST: Audio/Acoustics Test
43	DD: Routing/Via/Ground/Thermal Relief	99	ST: System Test Release
44	DD: Design Rule Check	100	REL: Component Test
45	DD: Gen Art/Prep Docs/Format ECAD	101	REL: Board Test
46	DD: Solder Stencil (send/receive supplier)	102	REL: Reliability Test
47	DD: Circuit Board (send/receive supplier)	103	REL: Environmental Test
48	DD: Parts (send/receive BOM to SCM)	104	REL: Verification Test
49	DD: Fabricate PCB	105	REL: Reliability Release
50	DD: Unit Test	106	FT: Field Test
51	DD: DD Engineering Sample	107	MFG: DFM Review
52	DSP: Layer 1 Processes	108	MFG: DFT Review
53	DSP: Layer 2 Processes	109	MFG: DFA Review
54	DSP: Speech Processes	110	MFG: Build Preliminary Prototypes
55	DSP: DSP States	111	MFG: Prototype Testing
56	DSP: DSP Firmware Release	112	MFG: Build Engineering Samples
		113	MFG: Engineering Sample Testing
		114	MFG: Post-Build Analysis

Table 2.2: List of Design Tasks

System Architecture

Architecture Requirements - Document that identifies detailed specifications for the design groups (Analog Design, Digital Design, Digital Signal Processor Design, and Microcontroller Design).

Human Factors

Voice Quality Test - Evaluates performance of the handset's audio voice quality in the field by a group of expert or trained listeners (e.g. subjective Mean Opinion Scores (MOS)).

User Interface Test - Evaluates performance of the handset's user interface to ensure and improve its usability and friendliness.

Audibles Test - Evaluates performance of the handset's ring patterns and tones to verify that they are distinguishable from one another and at the appropriate volume level.

Ergonomic Test - Evaluates performance of the handset's plastics to ensure a comfortable form factor, appropriate weight, absence of sharp edges, etc.

Field Trial Test - Evaluates performance of the handset in the field - prototypes are supplied to end-users that report data in the all of the aforementioned test areas.

Supply Chain Management

Order/Secure Standard Materials - Order and secure standard/generic materials (typically short lead times) from the Bill of Materials (BOM) for engineering samples (regularly delivered to Manufacturing).

Order/Secure Custom Parts - Order and secure custom/specialized materials (typically long lead times) from the Bill of Materials (BOM) for engineering samples (regularly delivered to Manufacturing).

Industrial Design

Tool Parts Available - Chassis and housing tools parts available for engineering samples.

Component Plastics Test - Evaluates performance of the handset's plastic components (e.g. reliability).

Component LCD Test - Evaluates performance of the handset's Liquid Crystal Display (LCD) (e.g. pixel cross-talk, contrast).

Component Chassis Test - Evaluates performance of the handset's chassis (e.g. dimensional compliance).

Component Acoustics Test - Evaluates performance of the handset's acoustic components (e.g. microphone).

Unit Drop Test - Evaluates performance of the handset to being dropped in a variety of configurations.

Unit Reliability/Environmental Test - Evaluates reliability performance of the handset to environmental conditions.

Unit Acoustics Test - Evaluates performance of the handset's acoustic system (e.g. frequency response, attenuation)

Industrial Design Release - Mechanical Computer Aided Design (MCAD) release (periodically released to Analog Design, Digital Design, and Manufacturing).

Analog Design

Draw/Update Schematic - Changes to the analog circuitry schematic drawings.

Enter Schematic (ECAD) - Enter the analog circuitry schematic drawings into the Electrical Computer Aided Design (ECAD) package.

Component Placement - Place the components on the printed circuit board in ECAD.

Routing/Via/Ground/Thermal Relief - Route the connections (via and ground) between components and perform thermal relief analysis.

Design Rule Check - Perform verification of design rules such as component spacing, line width, and other geometric attributes.

Gen Art/Prep Docs/Format ECAD - Generate artwork (for solder stencil supplier), prepare documents (e.g. Bill of Materials), and format the ECAD data (for circuit board supplier).

Solder Stencil (send/receive supplier) - Send artwork to supplier and receive solder stencil from supplier.

Circuit Board (send/receive supplier) - Send ECAD data to supplier and receive circuit board from supplier.

Parts (send/receive BOM to SCM) - Send Bill of Materials (BOM) to Supply Chain Management (SCM) and receive parts from Supply Chain Management.

Fabricate PCB - Manufacture the printed circuit board (PCB).
Unit Test - Evaluates performance of the analog subsystem.
AD Engineering Sample - Analog subsystem engineering sample (periodically released and delivered to System Integration).

Digital Design
Draw/Update Schematic - Changes to the digital circuitry schematic drawings.
Enter Schematic (ECAD) - Enter the digital circuitry schematic drawings into the Electrical Computer Aided Design (ECAD) package.
Component Placement - Place the components on the printed circuit board in ECAD.
Routing/Via/Ground/Thermal Relief - Route the connections (via and ground) between components and perform thermal relief analysis.
Design Rule Check - Perform verification of design rules such as component spacing, line width, and other geometric attributes.
Gen Art/Prep Docs/Format ECAD - Generate artwork (for solder stencil supplier), prepare documents (e.g. Bill of Materials), and format the ECAD data (for circuit board supplier).
Solder Stencil (send/receive supplier) - Send artwork to supplier and receive solder stencil from supplier.
Circuit Board (send/receive supplier) - Send ECAD data to supplier and receive circuit board from supplier.
Parts (send/receive BOM to SCM) - Send Bill of Materials (BOM) to Supply Chain Management (SCM) and receive parts from Supply Chain Management.
Fabricate PCB - Manufacture the printed circuit board (PCB).
Unit Test - Evaluates performance of the digital subsystem.
DD Engineering Sample - Digital subsystem engineering sample (periodically released and delivered to System Integration).

Digital Signal Processor Design
Layer 1 Processes - Processes running at the lowest possible layer (the DSP's low-level interface with the analog circuitry) that transmit/receive bits and perform framing. Also called Physical Layer processes.
Layer 2 Processes - Processes running at the layer above Layer 1 processes, that perform error detection and retransmission. Also called Data Link Layer processes.
Speech Processes - Processes running at the layer above the Layer 2 processes, that perform speech compression and decompression.
DSP States - Various states that the DSP assumes to perform initialization, synchronization, etc.
DSP Firmware Release - Compiled DSP source code into a binary object file and a hexadecimal file for use in integration testing, system integration, and manufacturing (periodically released and delivered to System Integration).

Microcontroller Design
Standard Type 1 Changes - Programming changes to code that adhered an old standard, necessary for the code to adhere to a new standard.
NAM Programming Changes - Changes to the Number Assignment Module (NAM), non-volatile memory that stored unit specific parameters (e.g. the telephone number, preferences).
Standard System Determination - Algorithm used to determine which service provider's system (commercial network of base stations) to get service from, based on a standard recommendation from an industry development group.
Type 3 Service Provisioning - Process by which the NAM parameters can be loaded using Type 3 service provisioning, also based on a standard recommendation from an industry standard group.
Type 2 Messages - Type 2 messages delivered from the base station to the handset, similar to paging services.
User Interface - Handset's interface with the user including the display and keypad/buttons.
CNIP - Caller Party Number Identification and Presentation (caller ID).
Authentication - Encryption of the handset's electronic serial number (ESN) to avoid fraud.

Roaming - Accelerated search procedure for frequency space (by utilizing a roaming list database) in areas where service is not offered by the service provider, built to a recommendation by a handset/equipment manufacturer.
Voice Mail Notification - Enables the base station to communicate notification and the number of voice mail messages to the handset.
Programming Lock - Ensures that the handset only works with the service provider.
Type 3 Roaming List - Allows the roaming list database to be transmitted using Type 3 service provisioning for access by the handset.
Integrity - Implements scenario using specialized mathematical techniques to test the integrity of the communications infrastructure between the base station and the handset.
Sleep - Extends the life of the battery (when the handset is not in its time slot, it is powered down).
Type 4 Protocol - Type 4 protocol that facilitates the management of service negotiation between two base stations at the time service is transferred from one base station to another.
Boot Block Support - Enacts a protected section of the flash memory such that the handset can recover from a prematurely terminated update cycle.
MC Firmware Release - Compiled microcontroller source code into a binary object file and a hexadecimal file for use in integration testing, system integration, and manufacturing (periodically released and delivered to System Integration).

System Integration
Develop Accessories Test Cases - Development of test cases for the handset accessories (e.g. hands-free kit).
Develop Type 1A Test Cases - Development of test cases for Type 1A functionality by simulating various transmission environments.
Develop Type 1B Test Cases - Development of test cases for Type 1B functionality such as originations and terminations.
Develop Type 4 Test Cases - Development of test cases for Type 4 protocol that facilitates base station-to-base station service negotiation.
Develop Integrity Test Cases - Development of test cases for the specialized mathematical scenarios that test the integrity of the communications infrastructure between the base station and the handset.
Develop Type 3 Test Cases - Development of test cases for Type 3 service provisioning that loads the handset parameters.
Develop Support Tools Test Cases - Development of test cases for the handset support tools (e.g. mobile diagnostic system monitor).
Develop User Interface Test Cases - Development of test cases for the user interface (e.g. display and keypad/buttons).
Integrate System Components - Integration of the system components (hardware and firmware).
Execute Accessories Test Cases - Execution of test cases for the handset accessories (e.g. hands-free kit).
Execute Type 1A Test Cases - Execution of test cases for Type 1A functionality by simulating various transmission environments.
Execute Type 1B Test Cases - Execution of test cases for Type 1B functionality such as originations and terminations.
Execute Type 4 Test Cases - Execution of test cases for Type 4 protocol that facilitates base station-to-base station service negotiation.
Execute Integrity Test Cases - Execution of test cases for the specialized mathematical scenarios that test the integrity of the communications infrastructure between the base station and the handset.
Execute Type 3 Test Cases - Execution of test cases for Type 3 service provisioning that loads the handset parameters.
Execute Support Tools Test Cases - Execution of test cases for the handset support tools (e.g. mobile diagnostic system monitor).
Execute User Interface Test Cases - Execution of test cases for the user interface (e.g. display and keypad/buttons).
Demo Prep - Activities to support preparation for demonstrations.
Field Test Support - Activities to support field testing.

System Integration Release - Integrated system components (hardware and firmware, periodically released and delivered to System Test).

System Test
Layer 1 Test - Evaluates performance of Layer 1, the low-level interface between the analog circuitry and the DSP that transmits/receives bits and performs framing.
Type 1 Test - Evaluates performance of Type 1 functionality.
Interoperability Test - Evaluates performance of Layer 2, the layer above Layer 1, that performs error detection and retransmission.
User Interface Test - Evaluates performance of the user interface (e.g. display and keypad/buttons).
Audio/Acoustics Test - Evaluates performance of the audio and acoustics system (e.g. frequency response, attenuation, and subjective Mean Opinion Scores (MOS)).
System Test Release - System Test results (periodically released and fedback to the functional groups).

Reliability
Component Test - Evaluates reliability/environmental performance of components.
Board Test - Evaluates reliability/environmental performance of printed circuit boards (PCBs).
Reliability Test - Evaluates performance of the handset to a sequence of environmental stresses that are designed to identify problem areas and verify product reliability (e.g. vibration, heat, drop, electro-static discharge, and temperature).
Environmental Test - Evaluates performance of the handset to a sequence of environmental stresses (similar to the Reliability Test, but based on the environment that the prototype units will be subjected to in normal usage).
Verification Test - Evaluates performance of the handset to a sequence of environmental stresses (similar to the Environmental Test, but for tool made samples and production units).
Reliability Release - Reliability test results (periodically released and fedback to functional groups).

Field Test
Field Test - Evaluates the overall performance of the handset (e.g. ability to originate and terminate calls) in a real-world environment.

Manufacturing
DFM Review - Design for Manufacturing (DFM) review prior to manufacturing engineering samples.
DFT Review - Design for Testing (DFT) review prior to manufacturing engineering samples.
DFA Review - Design for Assembly (DFA) review prior to manufacturing engineering samples.
Build Preliminary Prototypes - Build of preliminary prototypes of engineering samples.
Prototype Testing - Testing of preliminary prototypes of engineering samples.
Build Engineering Samples - Build of engineering samples.
Engineering Sample Testing - Testing of engineering samples.
Post-Build Analysis - Post-build analysis of manufacturing process.

2.8 Data Collection

The design activities identified in this research for constructing the DSM constituted a task-level DSM. Activities defined in the Requirements phase were primarily task-based - e.g. specification documents such as the Customer Requirements. Activities defined in the Design phase were primarily task-based - e.g. entering a schematic or writing the code for voice mail notification. Activities defined in the Integration and Test phases were primarily task-based - e.g. unit level testing, regulatory testing, and field testing. Also, this DSM captured the "currently followed" product development process.

For the analyses to be performed, coupling strength was selected as the inter-task metric for the off-diagonal positions. Selection of an intra-task metric for the on-diagonal positions was unnecessary because here, the purpose of the DSM was to analyze process structure and task interrelationships as opposed to task duration or cost. We will revisit the selection of an intra-task metric in Chapter 5.

Each technical manager recommended engineers that they felt were qualified to provide data on coupling strengths. Questionnaires were then distributed to approximately 68 engineers, and they were asked to evaluate their assigned task(s) relative to the other design tasks. In other words, if an engineer was recommended to evaluate the task "Perform Test X", they then had to evaluate the strength of the coupling between this task and the remaining 113 tasks from the exhaustive list. In a memo that accompanied the questionnaire, coupling strengths were rated on a High (H), Medium (M), and Low (L) scale, according the definitions shown in Table 2.3.

Symbol	Description	Possible Examples
H	*High Level of Coupling* - Task A requires significant interaction with Task B.	• Task A is frequently dependent on Task B for information, perhaps daily. • Task A cannot begin until Task B is completed.
M	*Medium Level of Coupling* - Task A requires moderate interaction with Task B.	• Task A is generally dependent on Task B for information, perhaps weekly or bi-weekly. • Task A can begin but cannot be completed without input from Task B.
L	*Low Level of Coupling* - Task A requires very little interaction with Task B.	• Task A is infrequently dependent on Task B for information, yet sometimes. • Task A can be completed but would prefer input from Task B.
EMPTY	*No Coupling* - Task A requires no interaction with Task B.	• Task A is not dependent on Task B for information. • Task A can be completed without input from Task B.

Table 2.3: Definitions of the Coupling Strengths for DSM Data Collection

This questionnaire captured neither pure sequential iteration nor pure parallel iteration. Tremendous difficulty was experienced in defining the coupling strengths because different functional groups had different interpretations of what it meant to be coupled. A purely sequential definition (e.g. Task A cannot begin until Task B is completed) did not appear to be globally applicable to all of the functional groups. A purely parallel iteration definition (e.g. Task A is frequently dependent on Task B for information, perhaps daily) did not appear to be globally applicable to all of the functional groups either. In retrospect, it appears that the sequential iteration definition was more appropriate when dealing with tasks that occurred during *different* process flow phases. For example, a task in the Integration phase could not begin until a task in the Design phase was completed. The parallel iteration definition was more appropriate when dealing with tasks that occurred during the *same* process flow phase. For example, a task in the Design phase would be frequently dependent on another task in the Design phase for information, perhaps daily. Therefore, this questionnaire essentially captured both sequential and parallel iteration - a form of combined iteration. The only exception was in the hardware design groups (Analog Design and Digital Design), where although the tasks in these groups occurred within the same process flow phase, iteration in these groups was essentially performed in sequence.

In Chapter 4 we will use these observations to segment the tasks into groups of parallel activities (the process flow phases) as a means to identify subsets of strongly coupled tasks, using the parallel iteration model and the Work Transformation Matrix (WTM). In Chapter 5, we will also build on these observations to predict completion time in the hardware design groups, using the sequential iteration model and the Reward Markov Chain (as well as the Signal Flow Graph).

Engineers were recommended to provide data for as low as a single task and as high as twelve tasks. This caused a significant disparity in the responses from the questionnaires. Some engineers responded that they were coupled to very few tasks, almost none, while others responded that they were coupled to

almost every task. To resolve these discrepancies and normalize the data, technical managers were asked to review the responses from the engineers in their functional group. Once the technical managers made their modifications, the data was considered valid and representative of the working environment. The manager's modifications were an extremely important step in normalizing the responses from the engineers.

This resulted in a 114 x 114 DSM that modeled the product development process. In the following two chapters we will present this DSM from different perspectives, and we will apply a number of analytical tools to perform analysis.

2.9 Discussion

In this chapter, we provided an overview of the product development process at the sponsor company. We also provided an introduction to the Design Structure Matrix (DSM), a modeling tool that captures the interrelationships between design activities in a compact form. In constructing the DSM, there were a few important observations that must be restated.

First, the importance of properly identifying the areas of the product development process to be modeled cannot be overemphasized. As mentioned earlier in the chapter, this is the first and perhaps most important step in constructing the DSM because it affects all subsequent steps.

Second, defining the design tasks at a sufficient level of abstraction and according to an appropriate rule of classification is also of critical importance. The choices here include a task-level DSM, parametric-level DSM, and hybrid DSM.

Third, differentiation between the "currently followed", "should-be followed", and the "will-be followed" product development processes must be clearly understood and articulated to interviewees when gathering DSM data.

Fourth, if the intent is to build a DSM that captures iteration, one must choose between the parallel iteration model and the sequential iteration model, and formulate design questionnaires accordingly. Lastly, once the data set is collected, if it has been gathered from a large number of people (whose opinions are likely to differ according to what constitutes a strong coupling or a weak coupling) then it is recommended that a much smaller number of people (e.g. managers) review the data before it is considered representative of the process. This serves to normalize the data across multiple responses.

Overall, the DSM gave us an extremely versatile framework to capture the product development process at the sponsor company. In fact, the DSM underlies much of the work described in the following two chapters, as well as this thesis.

CHAPTER 3: ANALYZING INFORMATION FLOW AND DEPENDENCIES

3.1 Modeling Information Transfer

The DSM can be used to capture information flow and dependencies between design activities. By substituting the marks in the matrix with numerical values we can also begin to quantify the strength of these couplings. Therefore, the DSM can serve two simultaneous purposes - as a visual aid and as a quantitative tool to examine the transfer of information throughout the product development process.

By rearranging and aggregating the marks in the matrix, we can represent the DSM from a number of different perspectives such as by design task or by groups of design tasks. These DSM representations can be useful in that they provide a visual perspective on information flow and dependencies during the product development process. Using the same perspectives, we can also perform coupling analysis as a means to quantify the interrelationships between design activities.

In this chapter, we first present an introduction to the basics underlying DSM representations and coupling analysis. Finally, we present the DSM representations and perform coupling analysis on the product development process for the digital wireless telephone at the sponsor company.

3.2 DSM Representations

The DSM is a useful tool for analyzing information flows and information dependencies. One of the reasons why the DSM is so attractive in this context is its flexibility. By exchanging rows/columns and rearranging the data accordingly, a method called Swapping [39], we can manipulate the DSM matrix and represent the same process from different perspectives. A visual inspection of the DSM from these different perspectives can provide general insight to the product development process.

As mentioned in Chapter 2, the tasks listed along the left column of the DSM represent design activities that receive information, while the tasks listed along the top row represent design activities that provide information. A relative measure of information received by a design task, or information that a design task is dependent upon from other tasks (inputs), can be evaluated by looking across its row. A relative measure of information provided by a design task, or information that other design tasks are dependent upon from a task (outputs), can be evaluated by looking down its column. We will build on this concept in the following section. As an example, Figure 3.1a shows the DSM representation of the same binary DSM previously shown in Figure 2.2.

	A	B	C	D	E	F
A						X
B	X				X	
C		X				
D		X				
E			X	X		
F					X	

(a)

	A	B	C	D	E	F
A						0.3
B	1.3				2.7	
C		0.2				
D		2.3				
E			0.5	1.1		
F					0.7	

(b)

Figure 3.1: DSM Representation and NDSM Representation

Performing a simple visual inspection of the DSM representation, we can see that Tasks B and E receive a relatively large amount of information (looking across their rows, the sum of the marks for both tasks is equal to 2), and Tasks B and E provide a relatively large amount of information (looking down their

columns, the sum of the marks for both tasks is equal to 2). While such a visual perspective of the DSM is instructive, for much larger matrices a more quantified and detailed perspective can be achieved by performing Coupling Analysis.

3.3 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.

At the individual task level, we can examine how information is transferred between individual design tasks, using only a few basic metrics. For example, we can identify design tasks that have the greatest potential to delay other design tasks or to be delayed by other design tasks. At the group task level, we can examine how information is transferred between groups of design tasks, such as functional groups or process flow phases. For example, we can identify functional groups or process flow phases that have the greatest potential to delay other functional groups or process flow phases or to be delayed by other functional groups or process flow phases, using only a few basic group metrics. This is done by assigning each design task to a particular grouping (e.g. a functional grouping such as "manufacturing" or a process flow phase grouping such as "integration"), and aggregating the design task data to a new, and hopefully more useful level of abstraction. Coupling metrics at the individual task level and the group task level can also be decomposed in terms of their constituent elements. For example, a functional group that is heavily dependent on another functional group for information can identify the tasks within that functional group that contribute to the dependency most. In the following sections, we describe in detail how to calculate and interpret the coupling metrics at the individual task level and the group task level.

3.3.1 Individual Task Level

If we replace the off-diagonal marks in the DSM with numerical equivalents (coupling strengths), we create a *Numerical Design Structure Matrix (NDSM)*. As an example, Figure 3.1b shows the numerical DSM representation of the same numerical DSM previously shown in Figure 2.3.

Using the NDSM, we can calculate metrics that characterize information flow and dependencies at the individual task level. Naturally, the coupling strengths themselves are useful metrics and give us immediate insight to tightly coupled pairs of tasks (for an $n \times n$ DSM, there are n^2 such pairs of tasks). In fact, they serve as the basis for calculating the other coupling metrics at the task level, which are as follows: input measure, output measure, and volume measure. We can also calculate corresponding measures as a percent of the total coupling.

As mentioned earlier, the tasks listed along the left column of the DSM represent design activities that receive information, while the tasks listed along the top row represent design activities that provide information. We can therefore calculate a relative measure of the amount of information required by a design task (input) by summing the values in its row, as shown in Equation 3.1.

$$input\_measure_j = \sum_{i=1}^{n} NDSM_{ji} \qquad \text{where} \quad \begin{array}{l} j = \text{task number} \\ n = \text{dimension of the NDSM} \end{array}$$

Equation 3.1

This simple metric can be useful in identifying tasks that may be overloaded or heavily dependent on other design activities in order to be completed. Similarly, we can calculate a relative measure of the amount of information provided by a task (output) by summing the values in its column, as shown in Equation 3.2.

$$output\_measure_j = \sum_{i=1}^{n} NDSM_{ij} \qquad \text{where} \quad \begin{array}{l} j = \text{task number} \\ n = \text{dimension of the NDSM} \end{array}$$

Equation 3.2

This metric can be useful in identifying tasks that may be overloading others or that others are heavily dependent upon in order to be completed. Similarly, we can calculate a relative measure of the amount of information received and provided by a task (volume) by summing its input and output measure, as shown in Equation 3.3.

$$volume\_measure_j = input\_measure_j + output\_measure_j \qquad \text{where} \quad j = \text{task number}$$

Equation 3.3

This metric can be useful in identifying tasks that are both heavily dependent on other tasks in order to be completed, and other tasks are heavily dependent upon in order to be completed. Finally, the coupling metrics at the task level (coupling strength, input measure, output measure, and volume measure*) can also be expressed as a percent of the overall coupling (upper bound of 100), by dividing each metric by the sum of all the coupling strengths, as shown in Equation 3.4.

$$metric\_percent_k = \frac{metric_k}{\sum_{i=1}^{n}\sum_{j=1}^{n} NDSM_{ij}} \qquad \text{where} \quad \begin{array}{l} k = \text{task number} \\ n = \text{dimension of the NDSM} \end{array}$$

*Note that *volume_percent* must be divided by 2 to avoid double counting.

Equation 3.4

As an example, Figure 3.2a shows coupling strengths, input measures, and output measures for the numerical DSM previously shown in Figure 2.3. Figure 3.2b shows coupling percentages, input percentages, and output percentages for the same numerical DSM.

	A	B	C	D	E	F	*Input Measure*
A						0.3	0.3
B	1.3				2.7		4.0
C		0.2					0.2
D		2.3					2.3
E			0.5	1.1			1.6
F					0.7		0.7
Output Measure	1.3	2.5	0.5	1.1	3.4	0.3	9.1

(a)

	A	B	C	D	E	F	*Input Percent*
A						3%	3%
B	14%				30%		44%
C		2%					2%
D		25%					25%
E			5%	12%			18%
F					8%		8%
Output Percent	14%	27%	5%	12%	37%	3%	100%

(b)

Figure 3.2: Coupling Analysis of a Numerical Design Structure Matrix

The cells in the matrix reveal that Tasks E and B represent the pairwise combination of tasks that transfer the largest amount of information (coupling strength = 2.7, coupling percent = 30%). The input measure and input percent reveal that Task B receives a relatively large amount of information (input measure = 4.0, input percent = 44%). Similarly, the output measure and output percent reveal that Task E provides a relatively large amount of information (output measure = 3.4, output percent = 37%). Calculating volume measures and volume percentages reveals that Task B yields that largest relative transfer of information by volume (volume measure = 6.5, volume percent = 36%).

3.3.2 Group Task Level

By aggregating the coupling strengths at the individual task level we can generate similar metrics for information flow and dependencies at the group task level. We can use these metrics to perform the same analysis at higher level of abstractions (e.g. by functional group or by process flow). As an example, Figure 3.3 shows a 3 x 3 *Aggregate Design Structure Matrix (ADSM)*, derived from the 6 x 6 numerical DSM previously shown in Figure 2.3. Here, we assume that each of the six tasks (letters A through F) is associated with one of three groups (numbers 1 through 3).

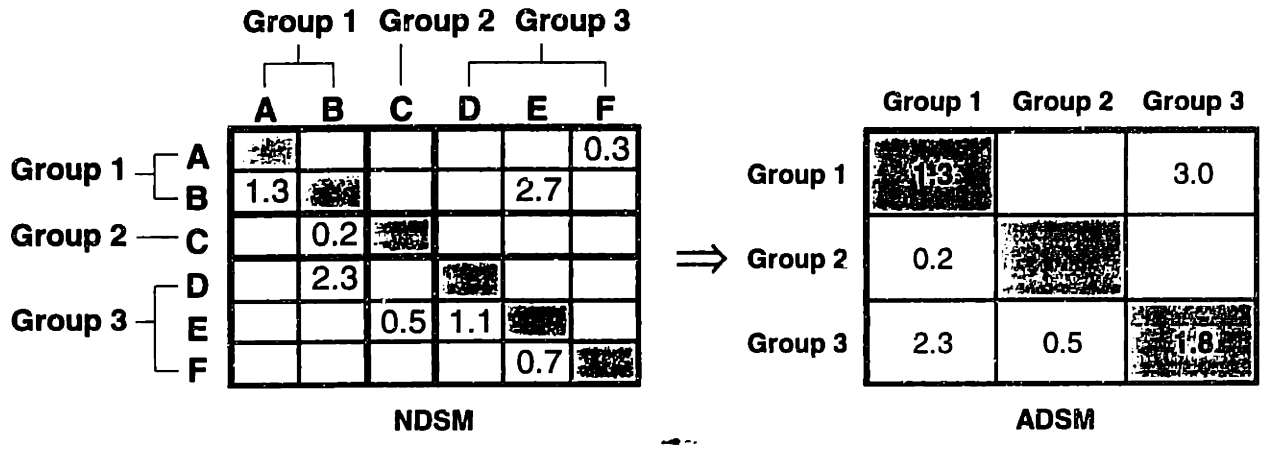


Figure 3.3: Aggregate Design Structure Matrix

The coupling strengths in the aggregate DSM serve as the basis for calculating the coupling metrics at the group level, which are as follows: relative coupling percent, relative input percent, relative output percent, and relative volume percent. All of these metrics are a function of the total available coupling for each metric.

We cannot calculate coupling metrics at the group level directly from the strengths in the aggregate DSM, because they are skewed in favor of large groupings. For example, a group made up of fifty tasks is much more likely to have a high coupling strength than a group made up of five tasks, simply by virtue of the potential for more couplings. We therefore modify the previous metrics, beginning with the coupling strengths themselves and generate a *Group Design Structure Matrix (GDSM)* with more appropriate relative coupling percentages. These percentages take into account the sizes of the various groupings. They are calculated as a function of the coupling strengths in the aggregate DSM and the maximum availble coupling strength for each cell in the group DSM. To determine these metrics, we must first define the maximum available coupling strength for a cell in the group DSM.

We will assume that the maximum coupling strength in the numerical DSM is the upper bound for the entire numerical DSM. Each cell in the aggregate DSM represents either the intersection of two groups with each other (off-diagonal positions) or a single group with itself (on-diagonal positions). The coupling strengths in these cells are derived directly from the coupling strengths in the numerical DSM (they are the sum of the coupling strengths for each respective grouping). Therefore, the maximum available coupling strength for any cell in the aggregate DSM is the product of the maximum coupling strength in the numerical DSM and the dimensions of the submatrix in the numerical DSM that constitute the two groupings or the single grouping. For example, the maximum coupling strength for the cell located in row 1/column 3 or Group #1/Group #3 of the aggregate DSM shown in Figure 3.3b is calculated as follows: 2.7 · 2 · 3 = 16.2, where 2.7 is the maximum coupling strength in the numerical DSM shown in Figure 3.3a, and 2 x 3 is the dimension of the submatrix in the numerical DSM shown in Figure 3.3a that constitute the intersection of Group #1 and Group #3.

If we divide each cell in the aggregate DSM by this metric, we generate a coupling measure that factors in the relative size of the grouping, and is expressed as a percent of the maximum available coupling strength for that cell. These relative coupling percentages represent the cells of the group DSM, and are calculated as shown in Equation 3.5.

$$GDSM_{ij} = \frac{ADSM_{ij}}{\delta \cdot m_i \cdot m_j}$$

where δ = max. coupling strength in *NDSM*
m_i = no. of tasks in group i*
m_j = no. of tasks in group j*

*Note that *i* and *j* correspond to row/column numbers *and* group numbers in the Group DSM/Aggregate DSM.

Equation 3.5

The relative coupling percentages are useful metrics and give us immediate insight to tightly coupled pairs of groups (for an $n \times n$ group DSM, there are n^2 such pairs of groups). As an example, Figure 3.4 shows the group DSM calculated from the coupling strengths of the aggregate DSM shown in Figure 3.3 (e.g. the relative coupling percent for row 1/column 3 or Group #1/Group #3 is calculated using Equation 3.5 as follows: 3.0 / (2.7 · 2 · 3) = 19%).

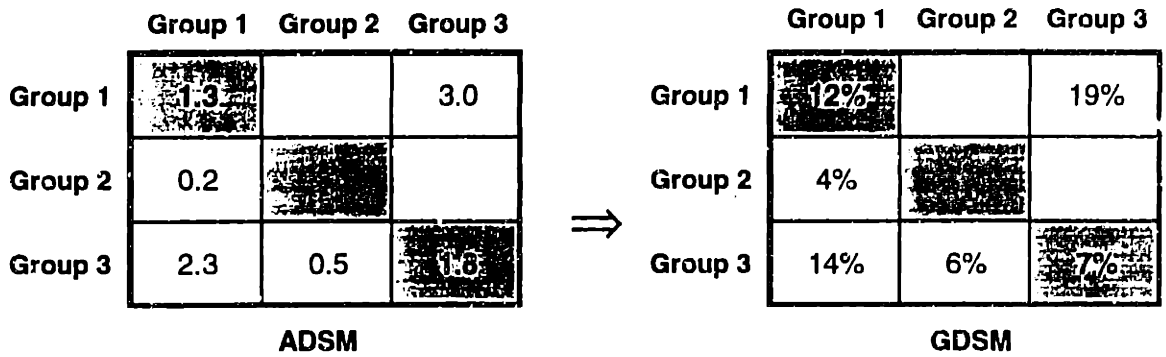


Figure 3.4: Group Design Structure Matrix

The remaining relative coupling metrics at the group task level are determined in two steps. First, we calculate an input measure, output measure, and volume measure for each group. This is an intermediate step. It is done using Equation 3.1, Equation 3.2, and Equation 3.3. However, we use the relative coupling strengths in the *aggregate* DSM instead of the coupling strengths in the *numerical* DSM to perform these calculations. This gives us corresponding group metrics (group input measure, group output measure, and group volume measure). Second, we divide these group metrics by the maximum available coupling for each metric (upper bound of 100). As opposed to the metric percentages at the individual task level, this gives us relative metric percentages at the group task level (relative input percent, relative output percent, and relative volume percent) as shown in Equation 3.6.

$$relative\_metric\_percent_k = \frac{group\_metric_k}{\rho}$$

where k = group number
ρ = max. available coupling

Equation 3.6a

$$\text{If } group\_metric_k = \begin{cases} group\_input\_measure, & \rho = \delta \cdot (n \cdot m_k) \\ group\_output\_measure, & \rho = \delta \cdot (n \cdot m_k) \\ group\_volume\_measure^*, & \rho = \delta \cdot (2 \cdot n \cdot m_k - m_k^2) \end{cases}$$

where
δ = max. coupling strength in *NDSM*
n = dimension of *NDSM*
m_k = no. of tasks in group k

*Note that $group\_volume\_measure_k = group\_input\_measure_k + group\_output\_measure_k - ADSM_{kk}$ to avoid double counting.

Equation 3.6b

To perform analysis, we use the relative metric percentages only. This information is useful in identifying groups of activities that must communicate regularly or work closely together, or in quantifying the amount of information transferred across groups or the level of dependence between groups. This would include intra-group information transfer as well as inter-group information transfer. As an example, Figure 3.5 shows relative coupling percentages (repeated), relative input percentages, and relative output percentages for the group DSM shown in Figure 3.4 (e.g. the relative input percent for Group #1 is calculated using Equation 3.6 as follows: (1.3 + 3.0) / (2.7 · 6 · 2) = 13%).

	Group 1	Group 2	Group 3	*Relative Input Percent*
Group 1	12%		19%	**13%**
Group 2	4%			**1%**
Group 3	14%	6%	7%	**9%**
Relative Output Percent	**12%**	**3%**	**10%**	

Figure 3.5: Coupling Analysis of a Group Design Structure Matrix

The cells in the matrix reveal that Groups #3 and #1 represent the pairwise combination of groups that transfer the largest amount of information (relative coupling percent = 19%). Notice that using the coupling strengths in the aggregate DSM shown in Figure 3.4a, one would have erroneously concluded that information transferred within Group #3 is the third ranked pair. However, when we take into account the relative size of Group #3 (it is the largest of all of the groups with three tasks) and its potential for greater coupling in the group DSM, we see that it is actually information transferred within Group #1 that represents the third ranked pair.

The relative input percent reveals that Group #1 receives a large amount of information (relative input percent = 13%). The relative output percent reveals that Group #1 also provides a large amount of information (relative output percent = 12%). Calculating relative volume percentages reveals, not surprisingly, that Group #1 yields that largest relative transfer of information by volume (relative volume percent = 13%).

An additional benefit of coupling analysis on a group DSM is the ability to quantify interactions at the higher levels of abstraction, in terms of the lower levels of abstraction. Large relative percentages at the group level can be further examined to identify the underlying individual tasks that contribute most to the value. For example, earlier we recognized information transferred from Group #3 to Group #1 as the maximum information transferred between groups by pair. Further investigation shows that this coupling is largely governed by information transferred from Task E to B (2.7 / 3.0 = 90% of the total coupling).

Appendix A shows MATLAB code that implements the coupling analyses performed in these examples.

3.4 Research Methodology

The data used to generate the DSM representations at the sponsor company were based on the results of questionnaires distributed to engineers and managers (see Section 2.8 for a description of the data collection process). They were asked to evaluate coupling strengths on a High (H), Medium (M), and Low (L) rating scale (see Table 2.3 for the definitions of these coupling strengths). To create a numerical DSM from this data, that could then be used for coupling and subsequent analysis, the three ratings were replaced with corresponding numerical values of 0.50, 0.25, and 0.05. These values were identical to the numbers used by Smith and Eppinger in a similar research project [33]. Once this steps was completed, we were prepared to generate the DSM representations and perform coupling analysis.

In the following sections, we present the DSM representations at the sponsor company organized by design task, by functional group, and by process flow. We also examine information flow and dependencies using coupling analysis at the individual task level (by design task) and the group task level (by functional group and by process flow).

3.5 Design Task Analysis

A DSM representation was generated and coupling analysis was performed at the design task level using the 114 design tasks previously identified in the Chapter 2 (see Table 2.2 for the complete listing).

3.5.1 DSM Representation

The DSM representation of the product development process by design task is shown in Figure 3.6. This representation was the most detailed representation. It gave a visual perspective on information flow and dependencies between design tasks. For example, by looking the DSM it was clear that the Handset Requirements (Task #3) required a significant amount of information from tasks that were associated with the Microcontroller Design group (notice the clustering of coupling marks near the center of its row). Despite these and other broad based observations, we found this representation to be too granular for any detailed conclusions. The reader is advised to refer to this diagram as we perform coupling analysis on the DSM representation by design task.

3.5.2 Coupling Analysis

Table 3.1 lists the top ranked design tasks by input measure and input percent. Table 3.2 lists the top ranked design tasks by output measure and output percent. Table 3.3 lists the top ranked design tasks by volume measure and volume percent. While hard recommendations for improvement could not necessarily be drawn from this data, the exhaustive lists were extremely useful as reference material. The input data was valuable in assessing the breadth and depth of tasks that had the greatest potential to delay a task by way of information or deliverables. The output data was valuable in identifying a task that had the greatest potential to delay other tasks by way of information or deliverables, and assessing the breadth and depth of the impact. The volume data was valuable in identifying tasks that experienced large inflows and outflows of information or deliverables, and assessing the breadth and depth of these movements.

Looking at the highest ranked tasks from the input data, it was clear that the owners of the Reliability board test and component test were heavily dependent on a large number of other tasks in order to perform these activities. Further investigation revealed that these dependencies were heavily concentrated in System Integration (37% and 31% of the total coupling respectively) and Manufacturing (9% and 14% of the total coupling respectively). Also, it was clear that System Integration's development of the user interface test and execution of the user interface test required significant input both within the group itself (26% and 33% of the total coupling respectively) and from Microcontroller Design (29% and 25% of the total coupling respectively). In the event that these or other tasks were experiencing delays, it was easy to reference this data and identify tasks that were likely to be the cause of the delay.

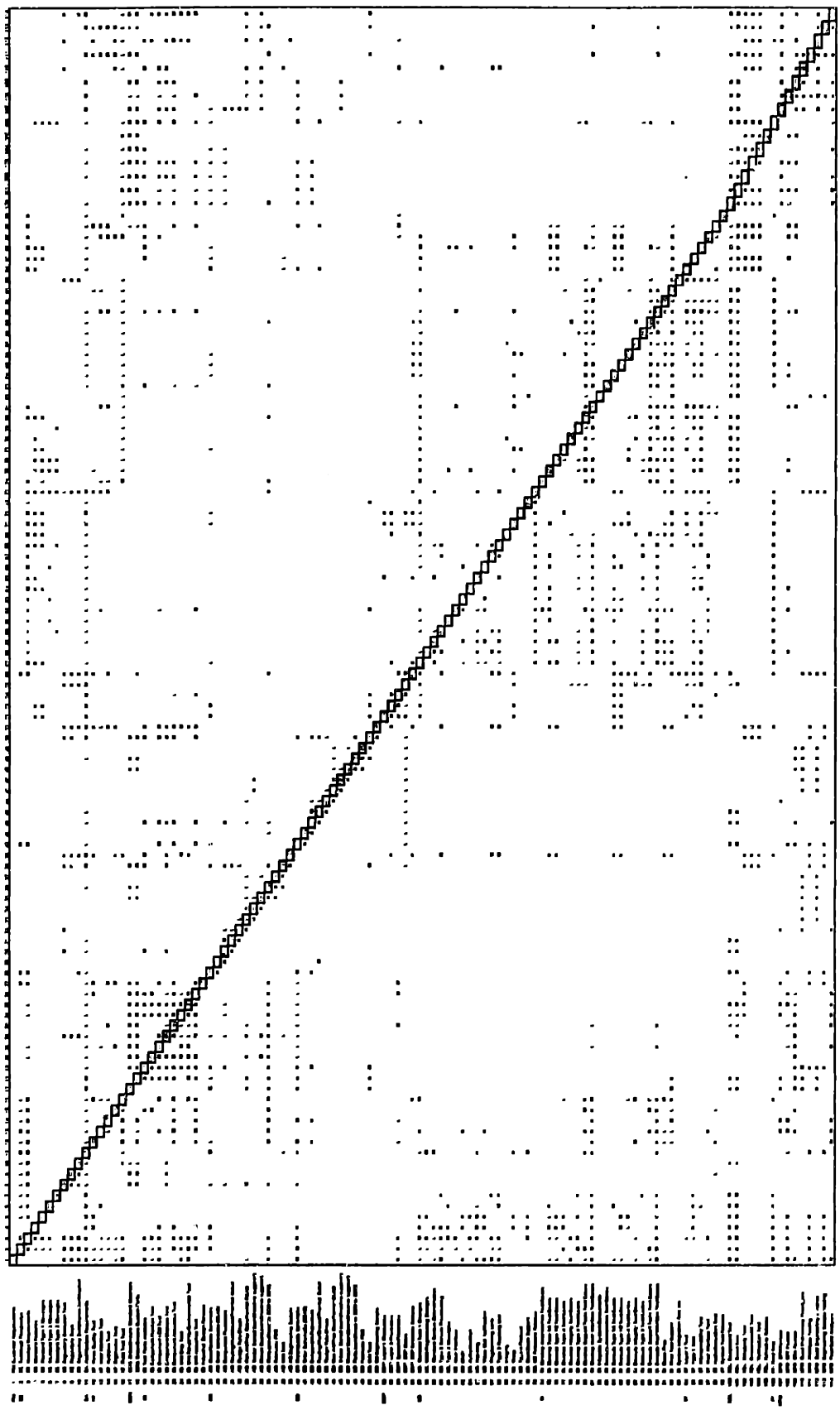


Figure 3.6: DSM Representation by Design Task

Rank	Design Task	Input Measure	Input Percent
1	REL: Board Test	20.10	3.11%
2	SI: Execute User Interface Test Cases	18.45	2.85%
3	SI: Field Test Support	17.80	2.75%
4	SE: Handset Requirements	16.25	2.51%
5	SI: Develop User Interface Test Cases	15.85	2.45%
6	REL: Component Test	15.45	2.39%
7	MC: Standard Type 1 Changes	14.95	2.31%
8	AD: Parts (send/receive BOM to SCM)	13.60	2.10%
9	FT: Field Test	13.00	2.01%
10	ST: Interoperability Test	12.15	1.88%

Table 3.1: Design Tasks Ranked by Input Measure and Input Percent

Rank	Design Task	Output Measure	Output Percent
1	SE: Handset Requirements	17.45	2.70%
2	DSP: DSP Firmware Release	14.50	2.24%
3	MC: MC Firmware Release	13.75	2.13%
4	DD: Engineering Sample	13.05	2.02%
5	AD: Engineering Sample	12.50	1.93%
6	PM: Customer Requirements	12.00	1.86%
7	SE: Standard Features	11.75	1.82%
8	SE: Standard Type 1 Messages	10.10	1.56%
9	SE: Technical Requirements	10.00	1.55%
10	MC: User Interface	9.40	1.45%

Table 3.2: Design Tasks Ranked by Output Measure and Output Percent

Rank	Design Task	Volume Measure	Volume Percent
1	SE: Handset Requirements	33.70	5.21%
2	SI: Execute User Interface Test Cases	27.10	4.19%
3	REL: Board Test	25.85	4.00%
4	MC: Standard Type 1 Changes	22.55	3.49%
5	SI: Develop User Interface Test Cases	22.45	3.47%
6	REL: Component Test	21.90	3.39%
7	SE: Standard Features	21.85	3.38%
8	MC: MC Firmware Release	21.80	3.37%
9	SI: Field Test Support	21.00	3.25%
10	DSP: DSP Firmware Release	21.00	3.25%

Table 3.3: Design Tasks Ranked by Volume Measure and Volume Percent

Many of the tasks that ranked high in the output data made intuitive sense - requirements documents, specification documents, and engineering samples/firmware releases, that countless other tasks were dependent upon. However, the data also suggested that Microcontroller Design's coding and testing of the user interface provided a significant amount of information to System Integration (51% of the total coupling), and Industrial Design's ability to make the chassis and housing tool parts available (not shown, ranked eleventh) was of considerable (and intuitive) importance to Industrial Design testing (38% of the total coupling) and tasks performed by Manufacturing (24% of the total coupling). In the event that these or other tasks were experiencing delays, it was easy to reference this data and identify tasks that were likely to feel the effects of the delay. The majority of the tasks that ranked high in the volume data were

also among the tasks that ranked high in either the input data or the output data. More valuable conclusions can be drawn from the functional group and process flow analyses.

3.6 Functional Group Analysis

A DSM representation was generated and coupling analysis was performed at the functional group level using the fifteen functional groups identified in Chapter 2 (see Table 2.1 for the complete list with abbreviations). The number of tasks associated with each functional group were as follows: *Product Management* (1), *System Engineering* (2), *System Architecture* (1), *Human Factors* (5), *Supply Chain Management* (2), *Industrial Design* (9), *Analog Design* (12), *Digital Design* (12), *Digital Signal Processor Design* (12), *Microcontroller Design* (17), *System Integration* (20), *System Test* (6), *Reliability* (6), *Field Test* (1), and *Manufacturing* (8).

3.6.1 DSM Representation

The DSM representation of the product development process by functional group is shown in Figure 3.8. This representation can be considered an organizational representation. It gave a visual perspective on information flow and dependencies between functional groups as well as insight to the organization of resources. For example, by looking at the DSM we noticed the sequential nature of design tasks in the Analog Design and Digital Design groups (notice the marks located below the diagonal), which would suggest a strong precedence relationship among these tasks. We could also see that Microcontroller Design provided significant outputs to System Integration (notice the clustering of marks located below the Microcontroller Design group and to the left of the System Integration group), which suggested the need for strong communication between the two groups. Again, the reader is advised to refer to this diagram as we perform coupling analysis on the DSM representation by functional group.

3.6.2 Coupling Analysis

Figure 3.7 shows the group DSM by functional group.

	PM	SE	SA	HF	SCM	ID	AD	DD	DSP	MC	SI	ST	REL	FT	MFG	***Relative Input Measure***
PM																**0%**
SE	33%	20%	11%	9%		4%	7%	5%	24%	24%	11%	17%		44%	6%	**12%**
SA	100%	53%		10%	50%	10%	12%	12%	50%	16%	4%	10%	7%	10%	38%	**19%**
HF	16%	10%	2%	10%		7%	5%	4%	6%	4%	6%	15%	6%	2%	21%	**7%**
SCM		12%		12%		61%	21%	21%	10%			3%	49%	50%	39%	**17%**
ID	40%	19%		27%	36%	40%	5%	11%	3%		2%	11%	27%	28%	26%	**13%**
AD	10%	5%	5%	4%	20%	14%	13%	2%	3%	1%	2%	1%	2%	1%	21%	**6%**
DD	4%	3%	6%	5%	17%	3%	2%	13%	4%	1%		6%	11%	8%	9%	**5%**
DSP	20%	4%		5%		4%	4%	5%	19%	15%	5%	2%		2%	2%	**6%**
MC	31%	20%	21%	4%				1%	11%	14%	8%	8%		6%	1%	**7%**
SI	16%	14%	15%	18%	4%	1%	2%	2%	17%	20%	19%	15%		16%	2%	**11%**
ST	2%	9%		7%					31%	23%	29%	13%				**11%**
REL	27%	17%			33%	11%	15%	15%	4%		21%	35%	49%	42%	46%	**18%**
FT	50%	56%		20%			8%	8%	20%	54%	28%	17%			13%	**23%**
MFG	31%	13%	15%	14%	44%	22%	21%	25%	10%	4%	6%	5%	24%	1%	44%	**16%**
Relative Output Measure	**21%**	**13%**	**9%**	**10%**	**13%**	**9%**	**6%**	**7%**	**12%**	**11%**	**9%**	**10%**	**9%**	**14%**	**14%**	

Figure 3.7: Group DSM by Functional Group

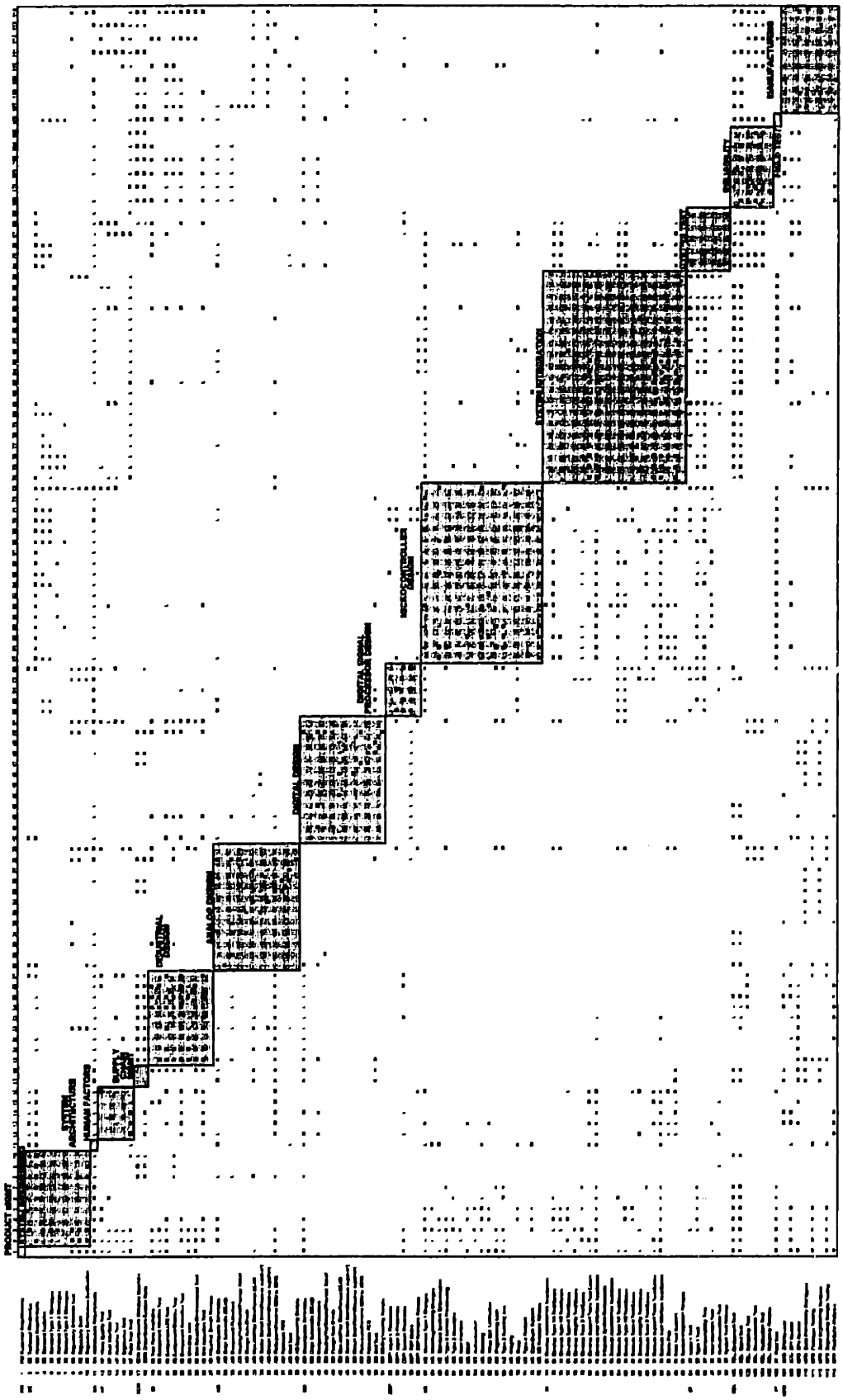


Figure 3.8: DSM Representation by Functional Group

In analyzing the DSM from these perspectives, we ignored the Product Management grouping, the System Architecture grouping, and the Field Test grouping, because each was comprised of a single task. In this sense, they did not constitute real groupings and tended to skew the rankings. From this data we were able to assess the effectiveness of the functional groupings, concurrent engineering, and strategic decoupling.

One of the goals of segmenting product development activities into functional groups is to minimize the interactions across groups and maximize the interactions within groups. An effective functional segmentation would be characterized by higher relative coupling percentages in the on-diagonal positions (intra-group information transfer) and lower relative coupling percentages in the off-diagonal positions (inter-group information transfer) of the group DSM. The data shows that except for System Test (relative coupling percent of 3%), whose activities were performed in relative isolation (within the group itself), this was generally the case as the remaining functional groups exhibited double-digit relative coupling percentages.

Consistent with concurrent engineering we should also expect relatively high degrees of coupling between the hardware design groups (Analog Design and Digital Design) and Manufacturing. The data shows relative coupling percentages of 21% (Analog Design to Manufacturing), 21% (Manufacturing to Analog Design), 25% (Digital Design to Manufacturing), and 9% (Manufacturing to Digital Design). This would support our expectations, with the exception that there appears to be a somewhat stronger reciprocal relationship between Analog Design and Manufacturing, as opposed to Digital Design and Manufacturing. In Chapter 7, we will present recommendations that are closely related to this observation.

Lastly, the sponsor company made an explicit effort to decouple hardware and firmware design activities at the beginning of the product development process by establishing detailed specifications on the hardware/firmware interface. This was a strategic choice in some sense, given the strong inherent coupling between the hardware (analog and digital) and firmware (DSP and microcontroller) on the handset, resulting from their close technical relationship. By minimizing this potentially time consuming coupling, it was their intent to reduce time-to-market. The group DSM provided a way to assess the effectiveness of this strategy. If it was effective then we would expect lower coupling values across the hardware and firmware groups, and higher coupling values within these groups. Looking at the group DSM, we see that the highest relative coupling values between the Analog Design, Digital Design, Digital Signal Processor Design, and Microcontroller Design groups are found in the on-diagonal values (within groups). These percentages lie between 13% and 19%. The remaining off-diagonal values (across groups) lie between 1% and 5%. The only off-diagonal values that are beyond this range represent information transferred from Microcontroller Design to Digital Signal Processor Design (14%), and vice-versa (15%), which is expected given the fact that both of these groups are firmware groups. The conclusion is therefore that the strategy was successful. Presumably, the specifications were established well enough at the beginning of the process to minimize the potential coupling between hardware and firmware. Another valuable perspective is gained by ranking the relative coupling metrics.

Table 3.4 lists the top ranked functional groups by relative coupling measure - a pairwise measure of information transfer between functional groups (information that is transferred within a functional group is highlighted in bold).

From this data it was clear that there was a strong relationship between the four functional groups that appeared consistently in the rankings: Industrial Design, Supply Chain Management, Reliability, and Manufacturing. The area of product development that these functional groups shared was the acquisition of parts and the qualification of suppliers. Industrial Design was responsible for selecting and testing the display and audio/acoustics parts. Supply Chain Management was directly responsible for the qualified suppliers and qualified parts list, and the procurement of parts for prototypes and production runs. Reliability was responsible for environmental and other testing of parts. Manufacturing was responsible for assessing the ability to manufacture, test, and assemble parts. As a result, one of the recommendations to the sponsor company was to establish a cross-functional team of representatives from each of these groups, charged with the joint responsibility of acquiring parts and qualifying suppliers. Recommendations are summarized in Chapter 7.

Rank	Functional Groups		Relative Coupling Percent
	From	To	
1	Industrial Design	Supply Chain Management	61.11%
2	Reliability	Supply Chain Management	49.17%
3	**Reliability**	**Reliability**	**49.17%**
4	Manufacturing	Reliability	45.83%
5	Supply Chain Management	Manufacturing	43.59%
6	**Manufacturing**	**Manufacturing**	**43.75%**
7	**Industrial Design**	**Industrial Design**	**39.75%**
8	Manufacturing	Supply Chain Management	38.75%
9	Supply Chain Management	Industrial Design	36.11%
10	System Test	Reliability	35.28%

Table 3.4: Functional Groups Ranked by Relative Coupling Percent

Table 3.5 lists the top ranked functional groups by relative input percent. Table 3.6 lists the top ranked functional groups by relative output percent. Table 3.7 lists the top ranked functional groups by relative volume percent.

Rank	Functional Group	Relative Input Percent
1	Reliability	17.89%
2	Supply Chain Management	17.02%
3	Manufacturing	16.03%
4	Industrial Design	13.12%
5	System Engineering	12.35%
6	System Test	11.10%
7	System Integration	10.98%
8	Human Factors	7.42%
9	Microcontroller Design	6.69%
10	Digital Signal Processor Design	6.14%
11	Analog Design	5.99%
12	Digital Design	4.55%

Table 3.5: Functional Groups Ranked by Relative Input Percent

Rank	Functional Group	Relative Output Percent
1	Manufacturing	13.66%
2	System Engineering	13.35%
3	Supply Chain Management	12.63%
4	Digital Signal Processor Design	11.89%
5	Microcontroller Design	10.66%
6	System Test	10.20%
7	Human Factors	10.05%
8	System Integration	9.47%
9	Industrial Design	9.12%
10	Reliability	9.04%
11	Digital Design	6.98%
12	Analog Design	6.17%

Table 3.6: Functional Groups Ranked by Relative Output Percent

Rank	Functional Group	Relative Volume Percent
1	Manufacturing	15.32%
2	Supply Chain Management	14.96%
3	Reliability	13.83%
4	System Engineering	13.33%
5	System Integration	11.58%
6	Industrial Design	11.14%
7	System Test	10.94%
8	Microcontroller Design	9.28%
9	Digital Signal Processor Design	9.22%
10	Human Factors	8.92%
11	Analog Design	6.40%
12	Digital Design	6.05%

Table 3.7: Functional Groups Ranked by Relative Volume Percent

The input data also supported our earlier observation that the four functional groups associated with parts and suppliers had a strong relationship to one another. Each of these groups appeared at the top of these rankings as well. This suggested that these groups also shared some commonality in either the nature or the amount of information that they required. Based on our previous conclusion, we surmised that such commonality was found in the former, with the nature of the information again being related to parts and suppliers.

The input data also suggested that the four design groups (Analog Design, Digital Design, Digital Signal Processor Design, and Microcontroller Design) were not heavily dependent on other functional groups for information, as they all appeared at the bottom of the rankings. In contrast, the two firmware groups ranked much higher (fourth and fifth respectively) by relative output percent. The fact that the two hardware groups did not rank high by relative output percent said more about the way that the groups' design tasks were first identified than it did about the importance of their deliverables. The tasks in the hardware groups were essentially performed in sequence, so there was a high dependence on the final tasks in these groups only (34% and 38% of these group's total coupling with other groups). On the other hand, the tasks in the firmware groups were largely performed in parallel, so there was a higher dependence on the intermediate tasks in these groups (55% and 85% of these group's total coupling with other groups) as well as the final tasks. Referring back to the design task rankings by input measure and input percent, we saw that the deliverables from the four design groups ranked second, third, fourth, and fifth. As a general observation, this underscored the importance of the design groups within a rapid prototyping paradigm. This data also supported the equal importance of Manufacturing's role, given its position in the relative input, output, and volume percent rankings (third, first, and first respectively).

Conceivably, the input, output and volume data could have also been used to identify functional groups that were the best candidates to be relocated in the event such an action was necessary. For example, the data suggested that Human Factors was neither heavily coupled by input, by output, or by volume, and would have been a likely candidate. Additional insight was drawn from the process flow analysis.

3.7 Process Flow Analysis

A DSM representation was generated and coupling analysis was performed at the process flow level using the four process flow phases identified in Chapter 2 (see Figure 2.4 for the process flow diagram). The number of tasks associated with each process flow phase were as follows: *Requirements* (8), *Hardware Design* (42), *Firmware Design* (31), *Integration* (21), and *Test* (12).

3.7.1 DSM Representation

The DSM representation by process flow is shown in Figure 3.10. This representation can be considered a temporal representation. As you move down a column and/or across a row, the tasks are ordered quasi-sequentially by virtue of their association with each of the four phases. This gave a visual perspective on information flow and dependencies between Requirements, Design (both Hardware and Firmware), Integration, and Test. For example, all of the tasks in the Hardware and Firmware phases occured before the tasks in the Integration phase. Also note that marks located above the diagonal represent feedback, and marks located below the diagonal represent feedforward (here, the diagonal is defined as the five large boxes within the DSM). For example, the marks located directly to the right of the Integration phase box and above the Test phase box represented feedback from the Test phase to the Integration phase. Again, the reader is advised to refer to this diagram as we perform coupling analysis on the DSM representation by process flow.

3.7.2 Coupling Analysis

Figure 3.9 shows the group DSM by process flow.

	REQ	HW	FW	INT	TEST	*Relative Input Measure*
Requirements	23%	3%	25%	2%	19%	**12%**
Hardware Design	12%	16%	3%	10%	9%	**10%**
Firmware Design	25%	1%	11%	5%	8%	**7%**
Integration	14%	10%	16%	11%	15%	**12%**
Test	15%	4%	18%	12%	8%	**11%**
Relative Output Measure	**17%**	**9%**	**11%**	**8%**	**10%**	

Figure 3.9: Group DSM by Process Flow

From this data we were able to assess the effectiveness of the process flow groupings. One of the goals of rapid prototyping with its short and iterative cycles (here, the Design, Integration, and Test phases) is to serve as an integrative force throughout the product development process. By integrating the activities of the various functional groups it is the hope that cross-functional problems will be uncovered downstream and fedback upstream. A more traditional product development process would entail longer and less frequent design, integration, and test cycles. In the traditional approach (if performed well), one would expect very high coupling values for feedforward, and very low coupling values for feedback. In the rapid prototyping approach (if performed well) one would expect more balance between feedforward and fedback. The group DSM provided a way to assess the effectiveness of this approach. Looking at the group DSM, we saw that balance was generally achieved (a 61% / 50% split between the sum of feedforward and feedback - Design, Integration and Test phases only, calculated by summing the relative coupling strengths above/below the diagonal, excluding the relative coupling strengths related to the Requirements phase). Therefore, our conclusion was that the strategy was successful - from the perspective of information flow. It appeared that information was indeed transferred from Hardware Design and Firmware Design to Integration and Test, and vice-versa.

Looking at the group DSM, we also observed that the lowest coupling values were shared by the Hardware Design phase and the Firmware Design phase (relative coupling percents of 1% and 3%). This supported our earlier conclusion that the sponsor company was successful in decoupling these two activities.

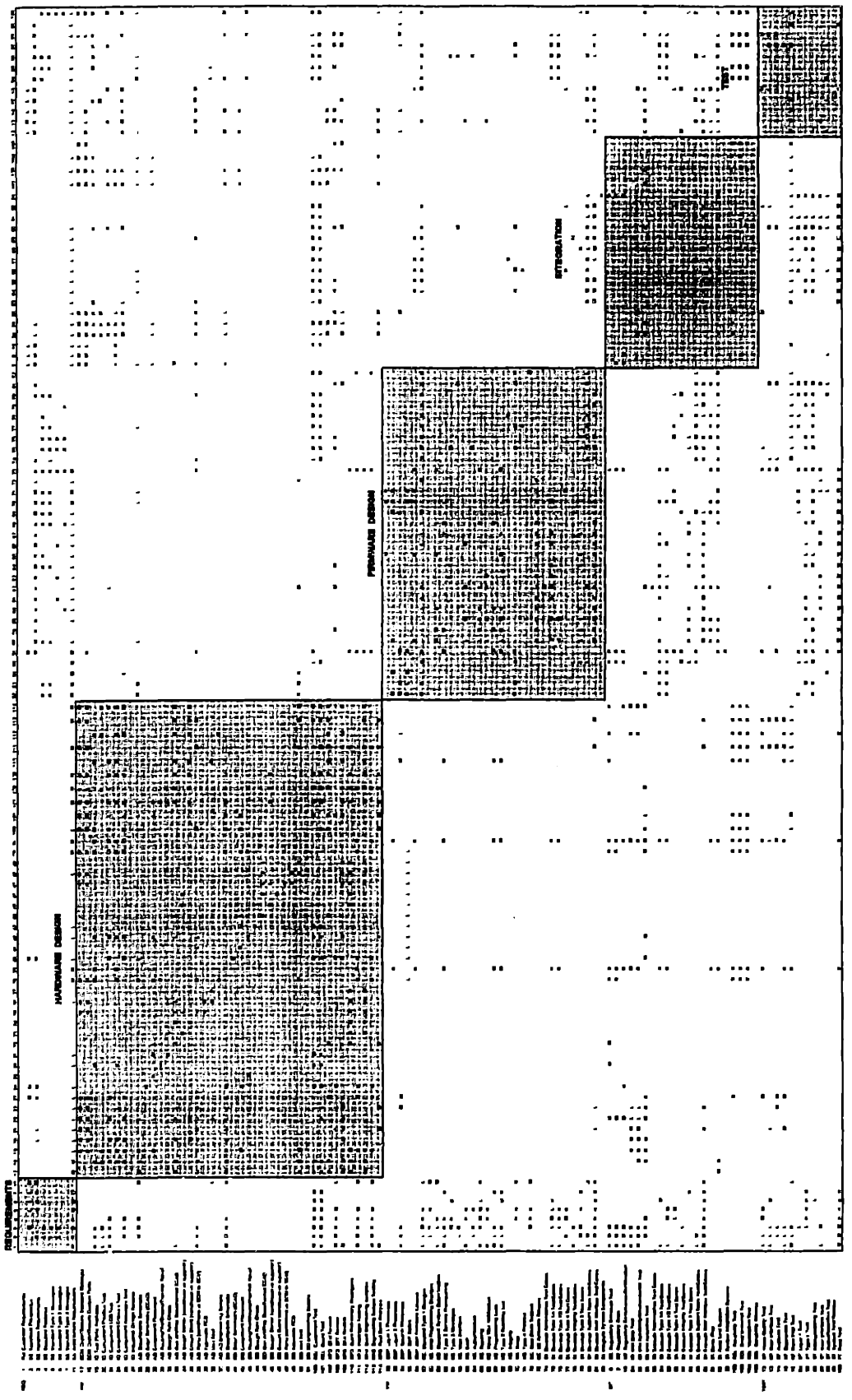


Figure 3.10: DSM Representation by Process Flow

Another valuable perspective was gained by ranking the relative coupling metrics. Table 3.8 lists the top ranked process flow phases by relative coupling measure - a pairwise measure of information transfer between process flow phases. Notice that the highest coupling values were shared by the Requirements phase and Firmware phase (relative coupling percent of 25% for feedforward and feedback). This suggested that the requirements established for firmware experienced considerable revision or modification. Consequently, one of the recommendations to the sponsor company was to invest more time at the beginning of the product development process specifying the requirements for firmware, particularly Microcontroller firmware (62% of the total feedforward coupling and 61% of the total feedback coupling), to avoid significant changes later on. Recommendations are summarized in Chapter 7.

We also noticed a high degree of coupling between the Integration and Test phases (relative coupling percent of 12% for feedforward and 15% for feedback). This suggested a strong temporal relationship between these phases that we will expand upon in Chapter 4.

Table 3.9 lists the top ranked functional groups by relative input percent. Table 3.10 lists the top ranked functional groups by relative output percent. Table 3.11 lists the top ranked functional groups by relative volume percent. The only observation that was derived from this data was the noticeably high rankings for the Requirements phase (second by input, first by output, and first by volume). It was concluded that the reason for this phenomenon stemmed from the fact that many of the requirements documents were based on industry standards, that were constantly being monitored and changing. Many of these standards described specifications that were related to micrccontroller firmware (e.g. Type 1, Type 2 messages, and Type 3 service provisioning). This would partially explain the aforementioned strong relationship between the Requirements phase and the Firmware phase. However it was also believed, as mentioned earlier, that part of this relationship was due to poorly defined internal requirements, independent of the industry standards.

Rank	Process Flow Phases		*Relative Coupling Strength*
	From	To	
1	Firmware	Requirements	25.20%
2	Requirements	Firmware	25.12%
3	Test	Requirements	19.38%
4	Firmware	Test	17.66%
5	Firmware	Integration	15.84%

Table 3.8: Process Flow Phases Ranked Pairwise

Rank	Process Flow Phase	*Relative Input Percent*
1	Integration	12.41%
2	Requirements	12.13%
3	Test	10.51%
4	Hardware Design	10.29%
5	Firmware Design	7.05%

Table 3.9: Process Flow Phases Ranked by Relative Input Measure

Rank	Process Flow Phase	Relative Output Percent
1	Requirements	17.01%
2	Firmware Design	10.63%
3	Test	10.43%
4	Hardware Design	8.78%
5	Integration	8.33%

Table 3.10: Process Flow Phases Ranked by Relative Output Measure

Rank	Process Flow Phase	Relative Volume Percent
1	Requirements	14.27%
2	Hardware Design	11.18%
3	Integration	10.87%
4	Test	10.49%
5	Firmware Design	9.21%

Table 3.11: Process Flow Phases Ranked by Relative Volume Measure

3.8 Discussion

In this chapter, we used DSM Representations and Coupling Analysis to analyze information flow and dependencies within the product development process at the sponsor company. DSM representations use a method called Swapping, which exchanges rows/columns of the DSM matrix, to represent the same process from different perspectives. DSM representations provided an interesting visual perspective on information inflows and outflows, but were too granular at times for any detailed conclusions. They were particularly useful as a supplement to the more formal coupling analysis. Coupling analysis is a simple tool that uses the information captured by the DSM to quantify the interrelationships between design tasks, functional groups, and process flow phases. Coupling analysis was shown to be a valuable tool in extracting a few worthwhile observations and general conclusions.

We performed coupling analysis at the individual task level (by design task) and at the group task level (by functional group and by process flow) using the product development process at the sponsor company. At the individual task level we calculated coupling metrics for information transfer by input, output, and volume. At the group level we calculated similar coupling metrics for relative information transfer by input, output, and volume, by adjusting for the sizes of the groupings. Furthermore, coupling metrics at both of these levels were decomposed in terms of their constituent elements.

The coupling metrics allowed us to assess the sponsor company's performance in a few areas such as the effective use of strategic decoupling, concurrent engineering, and rapid prototyping. Potentially, coupling analysis could have also been used to identify functional groups that would have had the least affect on the product development process were they to be relocated. While coupling analysis was useful to us in providing certain insights, it does present a number of challenges and difficulties.

First, it is too detailed at the design task level to derive any strong recommendations for improvement. At this level it is potentially useful as a reference when investigating potential delays to design tasks, and design tasks that can potentially delay other tasks.

Second, it ignores the relative weight or effort associated with a design task. Essentially, each design task contributes an equal weight to the coupling metrics regardless of the time or resources needed to accomplish the task. A task that requires fifty people, that has the same coupling strength as a task that requires a single person, contributes the same amount to the coupling metrics. This reinforces a point made in Chapter 2 regarding the importance of segmenting the product development process intelligently

when using the DSM. This also reinforces a similar point from Chapter 2 regarding the importance of normalizing the data when constructing the DSM.

Third, at the group task level coupling analysis can be slightly skewed for small groupings (on the order of one to two tasks). In such instances it is recommended that particular attention be paid to these groups. For purposes of this research these groupings were removed from the final rankings.

Fourth, the data that serves as the basis for coupling analysis is largely subjective. Task and group rankings by *input* are based on the survey responses of those requiring information, as opposed to task and group rankings by *output* that are based on the survey responses of those providing information. In other words, if a task or group is ranked high by input, it is because the owner of *that* task or the representatives from *that* group believed it to be true, and have reflected it in their survey responses. On the other hand, if a task or group is ranked high by output, it is because the owners of *other* tasks or the representatives from *other* functional groups believed it to be true, and have reflected it in their survey responses. Again, this reinforces a point made in Chapter 2 regarding the careful and intelligent selection of interviewees when gathering data for the DSM, and the importance of normalizing the data once it has been collected.

Lastly, coupling analysis requires a basic understanding of the product development process and its subtle dynamics to generate worthwhile recommendations. It is arguable whether or not it is simply a reflection of that which is already known.

Despite their shortcomings, both the DSM representation and coupling analysis, combined with a moderate familiarity with the product development process at the sponsor company, proved to be valuable tools in analyzing information flow and dependencies.

CHAPTER 4: ANALYZING CONTROLLING FEATURES AND TOTAL WORK

4.1 Modeling Design Iteration

The DSM can also be used as a means toward identifying coupled sets of design activities (Partitioning), coupled features of the design problem that will require many iterations (Controlling Features), and the total number of times a task is attempted during the iterative stages of the design process (Total Work). This can be extremely useful in predicting sets of closely related design activities that govern the rate and nature of convergence of the design effort.

Partitioning is performed directly on the DSM and is a precursor to controlling features and total work analysis. These analyses use a DSM-based model called the Work Transformation Matrix (WTM) model [35].

In this chapter, we describe partitioning first. This is followed by a description of the WTM model, its underlying assumptions, and the mathematical basis for the model itself. Next, we present the methodologies associated with controlling features and total work analysis (Eigenstructure Analysis). Finally, we apply these tools to the product development process of the digital wireless telephone at the sponsor company.

4.2 Partitioning

Partitioning is a process whereby the tasks in the DSM are rearranged by exchanging rows/columns (swapping) in an attempt to find a lower triangular solution [39]. As mentioned in Chapter 2, a lower triangular ordering represents an uncoupled design problem (no feedback) where each task receives all of its required information from its predecessors. It is rarely the case that a lower triangular solution is obtained, but rather a block lower triangular solution. Partitioning is also a useful tool in identifying the optimal sequence for a given set of tasks, or in identifying coupled blocks of tasks within the DSM. Here, we use partitioning to perform the latter, and as a precursor to controlling features and total work analysis. As an example, Figure 4.1a shows an unpartitioned binary DSM (the same binary DSM previously shown in Figure 2.2 with the coupling between Tasks E and F removed for illustrative purposes). Figure 4.1b shows the corresponding partitioned binary DSM.

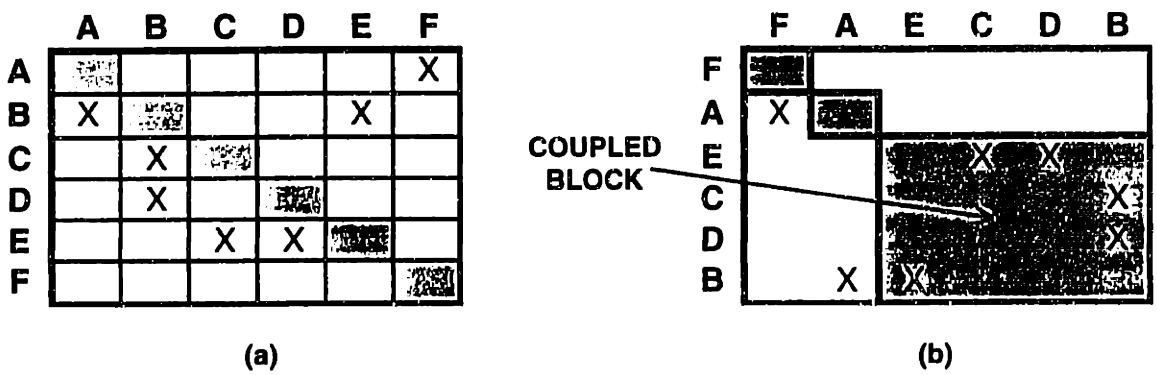


Figure 4.1: Unpartitioned and Partitioned Binary Design Structure Matrices

The solution represented by this ordering (F, A. E, C, D, B) is block lower triangular. The shaded blocks located on the diagonal represent individual tasks (Tasks F and A) and coupled sets of tasks due to feedback (Tasks E, C, D and B), while the marks below the diagonal represent feedforward. Later in this chapter, controlling features and total work analysis will be performed on the 4 x 4 coupled block (Tasks E, C, D and E) identified here via partitioning.

There are a number of different algorithms and approaches that can be used to perform partitioning. There is a publicly available software tool from NASA called DeMAID/GA (Design Manager's Aid for Intelligent Decomposition with a Genetic Algorithm) [29, 30, 31, 33] that automates partitioning, and was used in performing this research (described later in this chapter). DeMAID uses its own language for describing the structure of the DSM. Coupling strengths are captured using a seven-level scale: Extremely Weak (EW), Very Weak (VW), Weak (W), Neutral (N), Strong (S), Very Strong (VS), and Extremely Strong (ES). Code describing the structure of the DSM is input to DeMAID, partitioning is performed using a knowledge base coupled with a genetic algorithm, and the software outputs the optimal lower triangular or block lower triangular ordering [28, 32].

As an example, Figure 4.2a shows an unpartitioned numerical DSM (the same numerical DSM previously shown in Figure 2.3 with the coupling between Tasks E and F removed for illustrative purposes). Note that we have replaced the numerical values with a three-level coupling strength arrangement: Low (L), for values from 0.0 to 0.4, Medium (M), for values from 0.5 to 1.2, and High (H), for values from 1.3 to 2.7. We have also placed arbitrary task times in the on-diagonal positions for illustrative purposes (numbers 11, 37, 25, 18, 29, and 32). Figure 4.2b shows the corresponding partitioned numerical DSM.

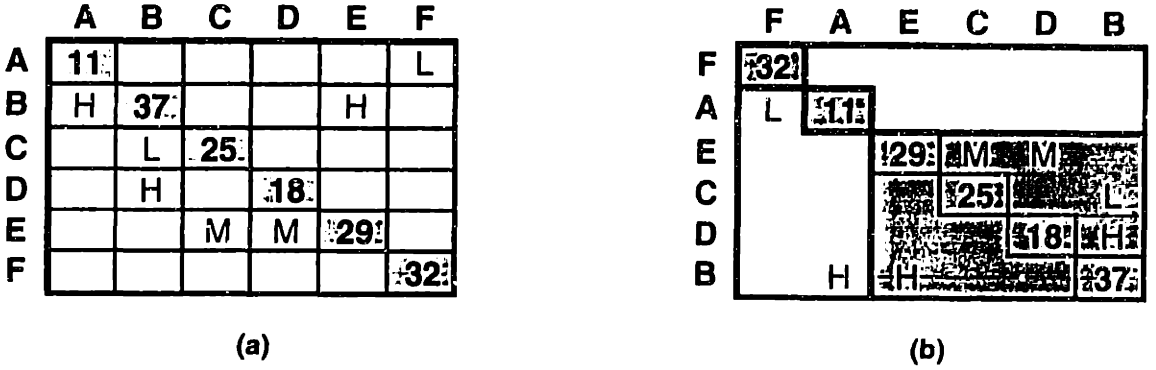


Figure 4.2: Unpartitioned and Partitioned Numerical Design Structure Matrix

Figure 4.3a shows input code to DeMAID corresponding to the DSM shown in Figure 4.2a. Note that we have replaced the three-level coupling strengths with corresponding DeMAID coupling strengths as follows: High (H) = Extremely Strong (ES), Medium (M) = Neutral (N), and Low (L) = Extremely Weak (EW). Figure 4.3b shows the DeMAID output corresponding to the partitioned DSM shown in Figure 4.2b, resulting from its own partitioning algorithm (note the following: 1) DeMAID allows a cost to be associated with each task - here we have assumed that the cost of each task is 0 as noted by the "0" in the fourth position of each "module" line of code, 2) Task #7, labeled "GOAL", is an artificial task sometimes generated by DeMAID to reach a solution, and 3) the input-output convention is reversed in DeMAID).

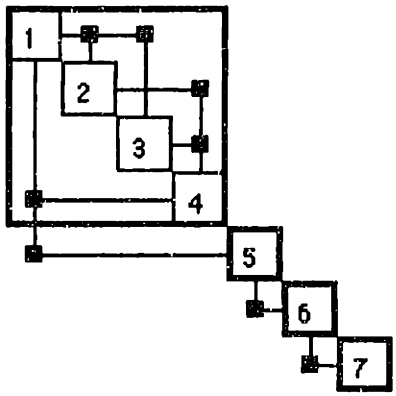
Clearly, DeMAID's solution is the same block lower triangular ordering (F, A, E, C, D, B) previously identified in Figure 4.2. Again, later in this chapter, controlling features and total work analysis will be performed on the 4 x 4 coupled block (Tasks E, C, D and E) identified here via partitioning.

Appendix B shows instructions and Visual Basic code that generates the DeMAID code in this example by converting a DSM stored in a Microsoft Excel spreadsheet to a format suitable for input to DeMAID.

```
(title dsm)
(maximum 7)
(module 1 A 0 11 A uk B)
(module 2 B 0 37 B uk C D)
(module 3 C 0 25 C uk E)
(module 4 D 0 18 D uk E)
(module 5 E 0 29 E uk B)
(module 6 F 0 32 F uk A)
(module 7 GOAL 0 0 goal uk F)
(strength uk es A B)
(strength uk ew B C)
(strength uk es B D)
(strength uk n C E)
(strength uk n D E)
(strength uk es E B)
(strength uk ew F A)
(strength uk n goal F)
```

(a)

###	Label	Time
1	B	37
2	D	18
3	C	25
4	E	29
5	A	11
6	F	32
7	GOAL	0



(b)

Figure 4.3: DeMAID Code and Output

4.3 Work Transformation Matrix (WTM)

Controlling features and total work analysis are based on the Work Transformation Matrix (WTM) model. The assumptions underlying the WTM model are as follows [35]:

- Every task is performed during each iteration stage.
- Each task creates a deterministic amount of rework for other tasks.
- Rework performed in the current iteration stage is a function of work performed in the previous iteration stage.

The WTM models design iteration by analyzing a coupled set of design tasks from a numerical DSM. The coupled set of design tasks is identified via partitioning, on which controlling features and total work analysis are then performed.

Recall that the on-diagonal elements of the DSM represent an intra-task metric. Here, this metric is the amount of time it takes to complete a task during its first iteration. The off-diagonal elements of the DSM represent an inter-task metric. Here, this metric is the strength of dependence between tasks, which causes rework. Rework is defined as "the required repetition of a task because it was originally attempted with imperfect information (assumptions)" [35]. For example, a coupling strength of 0.5 between Task A and B, would be interpreted to say that if Task A is performed, it causes 50% of Task B to be reworked. Rework gives rise to iterations. As information is acquired, the amount of rework diminishes, and the design problem converges.

The WTM model begins with a work vector u_t, an n-tuple, where n is the number of coupled design tasks to be completed, as well as the dimension of the coupled block identified via partitioning. The elements of u_t represent the amount of work to be performed on each design task after iteration t. During each iteration, all work is performed on the design tasks and each design task creates rework for other design tasks. The WTM matrix contains amounts of rework in the off-diagonal positions, and task times in the on-diagonal positions. The elements of the $n \times n$ matrix A are the off-diagonal elements of the WTM, while the elements of the $n \times n$ matrix W are the on-diagonal elements of the WTM. For example, if all of the design tasks in the DSM are coupled, then A is the off-diagonal elements of the entire DSM, while W is the on-diagonal elements of the entire DSM.

The work vector u_t changes during each iteration stage, according to the relation shown in Equation 4.1.

$$u_{t+1} = Au_t$$

Equation 4.1

The n elements of the initial work vector u_0 are all ones, which is interpreted to mean that work has yet to be performed on all of the design tasks. The work vector u_t can therefore be expressed as shown in Equation 4.2.

$$u_t = A^t u_0$$

Equation 4.2

The total work vector U, and n-tuple, is the sum of all of the work vectors, and is expressed in units of the initial amount of work to be performed. This vector represents the total number of times each of the design tasks is attempted during the M iterations that occur between iterations, as shown in Equation 4.3.

$$U = \sum_{t=0}^{M} u_t = \sum_{t=0}^{M} A^t u_0 = \left(\sum_{t=0}^{M} A^t \right) u_0$$

Equation 4.3

If the $n \times n$ matrix A is diagonalizable (has n linearly independent eigenvectors), it can be decomposed by performing *diagonalization*. Diagonalization factors A into a product of the form shown in Equation 4.4 .

$$A = S\Lambda S^{-1} \quad \text{where} \quad \begin{array}{l} S = \text{diagonalizing eigenvector matrix of } A \\ \Lambda = \text{diagonal eigenvalue matrix of } A \end{array}$$

Equation 4.4

If A is diagonalizable, the column vectors of the $n \times n$ diagonalizing matrix S are the eigenvectors or characteristic vectors of A, and the elements of the $n \times n$ diagonal matrix Λ are the corresponding eigenvalues or characteristics values of A. A^t can therefore be expressed as shown in Equation 4.5.

$$A^t = S\Lambda^t S^{-1}$$

Equation 4.5

Substituting Equation 4.5 into Equation 4.3, the total work vector U can be expressed as shown in Equation 4.6.

$$U = S\left(\sum_{t=0}^{M} \Lambda^t \right) S^{-1} u_0$$

Equation 4.6

As M approaches infinity, the total work vector is bounded as long as the system is stable (the maximum eigenvalue is less than one). A sufficient, but not necessary, condition for stability is that the entries in every row or in every column of the matrix A sum to less than one [35]. Taking the limit as M increases toward infinity, we use the relation shown in Equation 4.7.

$$\lim_{M \to \infty} \sum_{t=0}^{M} \Lambda^t = (I - \Lambda)^{-1} \quad \text{where } I = \text{identity matrix (diagonal of ones)}$$

Equation 4.7

Finally, substituting Equation 4.7 into Equation 4.6 we can express the total work vector U as shown in Equation 4.8.

$$U = S(I - \Lambda)^{-1} S^{-1} u_0$$

Equation 4.8

Given the task durations in the diagonal positions of the matrix *W*, we can calculate the total amount of time required by each design task in the vector *T*, as shown in Equation 4.9.

$$T = WU$$

Equation 4.9

Given this mathematical framework, we can now describe the specific steps for using the WTM model to perform controlling features and total work analysis.

4.4 Controlling Features and Total Work Analysis

Once a coupled set of design tasks has been identified via partitioning, the next step in using the WTM as an analytical tool is to examine the controlling features of design iteration. Controlling features analysis identifies design modes. Design modes are defined as "groups of design tasks that are very closely related such that working on any one of them creates significant work, directly or indirectly, for each of the other tasks within the model" [35]. A design mode can typically identify a critical subset of design activities or a subproblem of the overall development process. Using the eigenvalues and eigenvectors described in the previous section, we can use the WTM to identify design modes.

Eigenvalues and eigenvectors are well developed concepts in linear algebra [15]. They play an important role in the solution of systems of linear differential equations and in observing the behavior of various complex systems. Smith and Eppinger write [35]:

> *"The interpretation of the eigenvalues and eigenvectors for design problems is similar to the eigenstructure analysis used to examine the dynamic motion of a physical system. In the discrete time description of linear dynamic systems, each eigenvalue corresponds to a rate of convergence of one of the modes of the system (a natural frequency determining the decay or oscillation of the mode). The eigenvectors identify the mode shapes of natural motion, quantifying the participation of the state variables in each mode."*

In other words, the eigenvalues and eigenvectors describe the rate and the nature of convergence of each design iteration. The magnitude of eigenvalue *n* characterizes the geometric rate of convergence of design mode *n*. The *m-th* entry in the corresponding eigenvector *n* characterizes the relative contribution of design task *m* to design mode *n*. The largest eigenvalue will identify the slowest design mode, or, the design mode that plays the most significant role in determining the geometric rate of convergence. The largest entry in the corresponding eigenvector will identify the design task that has the largest contribution to this design mode. Therefore, by ranking the eigenvalues we can rank the design modes. By ranking the entries in the corresponding eigenvector, we can rank the contributions of design tasks to these modes. In doing so, we can identify the controlling features of each design iteration.

To rank the design modes, Smith and Eppinger recommend using the *ranking factor* or ranking criteria, as shown in Equation 4.10.

$$ranking\_factor_i = \frac{1}{1 - \mathrm{Re}(\lambda_i)} \qquad \text{where} \quad \lambda_i = \text{eigenvalue no. } i$$

Equation 4.10

To rank the design tasks in the corresponding eigenvectors, Smith and Eppinger recommend calculating the *participation factors* or participation vector for each design mode, as shown in Equation 4.11.

$$participation\_factors_i = S_{\bullet i}\frac{1}{1-\lambda_i}\sum_{j=1}^{n} S_{ij}^{-1} \qquad \text{where } S_{\bullet i} = ith \text{ column of } S$$

Equation 4.11

We can define a participation matrix P that consists of all of the participation vectors, and rank the design tasks within each design mode by ranking the corresponding entries in the matrix P.

The final step in using the WTM as an analytical tool is to examine the total work vector U shown in Equation 4.8. By ranking the entries in the total work vector we can identify those design tasks that experience the largest number of iterations, or those design tasks that perform the most work. The design tasks in the total work vector are easily ranked according to their numerical contributions.

As an example, we perform controlling features and total work analysis on the 4 x 4 coupled block identified via partitioning in the previous section, as shown in Figure 4.2. First, we replace the three-level coupling strengths (High (H), Medium (M), and Low (L)) with numerical equivalents of 0.50, 0.25, 0.05 (note that for larger matrices, to perform the eigenstructure decomposition, these values must be scaled down proportionally such that the off-diagonal values in every row or column sum to less than one), producing the 4 x 4 WTM matrix for Tasks E, C, D and B, as shown below:

$$WTM = \begin{bmatrix} 29 & 0.25 & 0.25 & 0 \\ 0 & 25 & 0 & 0.50 \\ 0 & 0 & 18 & 0.50 \\ 0.50 & 0 & 0 & 37 \end{bmatrix} \begin{matrix} E \\ C \\ D \\ B \end{matrix}$$

The matrices A and W are easily determined as follows:

$$A = \begin{bmatrix} 0 & 0.25 & 0.25 & 0 \\ 0 & 0 & 0 & 0.05 \\ 0 & 0 & 0 & 0.50 \\ 0.50 & 0 & 0 & 0 \end{bmatrix} \qquad W = \begin{bmatrix} 29 & 0 & 0 & 0 \\ 0 & 25 & 0 & 0 \\ 0 & 0 & 18 & 0 \\ 0 & 0 & 0 & 37 \end{bmatrix}$$

Diagonalization of A according to Equation 4.4, produces the eigenvector matrix S and eigenvalue matrix Λ as follows:

$$S = \begin{bmatrix} 0.1690+0.4276i & 0.1690-0.4276i & 0.4598 & 0 \\ -0.0677-0.0100i & -0.0677+0.0100i & 0.0685 & 0.7071 \\ -0.6775-0.1005i & -0.6775+0.1005i & 0.6849 & -0.7071 \\ 0.3488-0.4395i & 0.3488+0.4395i & 0.5611 & 0 \end{bmatrix}$$

$$\Lambda = \begin{bmatrix} -0.2048+0.3548i & 0 & 0 & 0 \\ 0 & -0.2048-0.3548i & 0 & 0 \\ 0 & 0 & 0.4097 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Using Equation 4.11, the participation matrix P is calculated as follows:

$$P = \begin{bmatrix} -0.0768 - 0.0513i & -0.0768 + 0.0513i & 1.7164 & 0 \\ 0.0123 - 0.0061i & 0.0123 + 0.0061i & 0.2557 & 0.8182 \\ 0.1234 - 0.0609i & 0.1234 + 0.0609i & 2.5569 & -0.8182 \\ -0.0074 + 0.1125i & -0.0074 - 0.1125i & 2.0950 & 0 \end{bmatrix}$$

Using Equation 4.10, the ranking factors are as follows:

Eigenvalue No.	Ranking Factor
1	0.8300
2	0.8300
3	1.6941
4	0

From the rankings, it is clear that the highest ranked design mode is associated with the third eigenvalue (λ = 0.0819). Looking at its corresponding eigenvector and/or participation vector, we see that the third task (eigenvector value = 0.6849, participation vector value = 2.5569) and fourth task (eigenvector value = 0.5611, participation vector value = 2.0950) contribute the most to the mode (Tasks D and B respectively). We can therefore conclude that the strong relationship between Tasks D and B govern the rate and nature of convergence of this design problem.

Using Equation 4.8 and Equation 4.9, the total work vector U and total time vector T are calculated as follows:

$$U = \begin{bmatrix} 1.7584 \\ 1.0940 \\ 1.9396 \\ 1.8792 \end{bmatrix} \begin{matrix} E \\ C \\ D \\ B \end{matrix} \qquad T = \begin{bmatrix} 50.9933 \\ 27.3490 \\ 34.9128 \\ 69.5302 \end{bmatrix} \begin{matrix} E \\ C \\ D \\ B \end{matrix}$$

We can finally conclude that Tasks D and B perform the most work (experience the largest number of iterations), while Tasks B and E require the most time.

Appendix C shows MATLAB code that implements the eigenstructure analysis performed in these examples by ranking the design modes using the ranking factor, ranking the design tasks using the participation factors, and ranking the entries in the total work vector.

4.5 Research Methodology

To perform partitioning, we used the aforementioned publicly available software package DeMAID from NASA. Because our coupling data was collected on a three-level scale (High, Medium, Low - see Section 2.8 for the definitions of these coupling strengths), it had to be mapped to corresponding DeMAID coupling strengths. In selecting the mapping relationship, we chose to maximize the dynamic range of the software's capabilities, as shown in Table 4.1.

To perform controlling features and total work analysis we had to first ensure that we satisfied the assumptions underlying the WTM model. Once again, because this research was focused specifically on those activities conducted during the rapid prototyping iterations (Design, Integration, and Test phases), we ignored the tasks that occurred during the Requirements phase.

DSM Coupling Strengths		Corresponding DeMAID Mappings	
Symbol	Description	Symbol	Description
H	High	ES	Extremely Strong
		VS	Very Strong
		S	Strong
M	Medium	N	Neutral
		W	Weak
		VW	Very Weak
L	Low	EW	Extremely Weak

Table 4.1: DSM Coupling Strengths and Corresponding DeMAID Mappings

The first assumptions states that all of the design activities occur in parallel. To satisfy the first assumption, we grouped the design tasks accordingly. This was almost the same as grouping the tasks by process flow phase. We therefore grouped the tasks in the Design phase together (Hardware and Firmware), and we grouped the tasks in the Integration and Test phases together. The reason we grouped the tasks in the Integration and Test phases together was because it was believed that although these activities were slightly separated in time, they closely resembled parallel iteration. This was particularly true given the concurrent engineering nature of the research environment. This argument was also supported by the high reciprocal coupling between the Integration and Test phases revealed during coupling analysis in Chapter 3 (see Section 3.7.2 for the coupling analysis). The majority of the tasks that occurred within these groupings approximated parallel iteration. The only clear exceptions were the tasks associated with Analog Design and Digital Design, which were executed sequentially.

The second assumption states that the parameters in the matrix are time-invariant. Within the research environment, there were several iterations of the Design, Integration, and Test phases. However, we modeled all of the iterations as a single iteration, and performed our analysis only once for each grouping. In others words, we assumed that the couplings that existed during each prototype iteration were the same for every prototype iteration. Therefore, this second assumption was valid inasmuch as the couplings identified during data collection remained constant during the time between prototype iterations, and remained constant across each of the several prototype iterations. Because we segmented our analysis into parallel activities that occurred repeatedly for each prototype iteration, and because the time between each interval was relatively short, this second assumption was likely to hold.

The third assumption states that rework performed in the current iteration stage is a linear function of the work performed in the previous iteration stage. Because we were modeling each prototype iteration as a single prototype iteration, we needed only be concerned with rework that was performed between these intervals, as opposed to across all of the intervals. In other words, since there were likely to be very few iterations of design tasks between each prototype iteration, this third assumption was also reasonable as most of the work was performed during the first iteration.

Finally, to perform controlling features and total work analysis, we again replaced the three ratings (High, Medium, and Low) with corresponding numerical values of 0.50, 0.25, and 0.05 (again, for large matrices, to perform the eigenstructure decomposition, these values must be scaled down proportionally such that the off-diagonal values in every row or column sum to less than one - consequently, these values were actually scaled down by a factor of five). Smith and Eppinger have shown that the results of controlling features and total work analysis are not sensitive to minor changes in these values [35].

In the following sections, we perform partitioning and controlling features/total work analysis on the DSM at the sponsor company. This is done by first searching for strong correlations between design tasks within the process flow groupings (Hardware/Firmware and Integration/Test). In doing so, we seek to identify coupled subsets of design activities that govern the iterations of the product development process. Lastly, we search for similar correlations between functional groups as a means to corroborate earlier findings.

4.6 Hardware and Firmware Design Phase Analysis

The 73 tasks performed during each iteration of the Hardware and Firmware Design phases were focused on the completion of engineering samples (hardware prototypes) and firmware releases (firmware prototypes). Also conducted during these phases were various supportive and integrative activities prior to the Integration and Test phases (see Section 2.6 for a detailed description of the process flow phases). In performing partitioning and controlling features/total work analysis on the tasks within these phases, we sought to identify strongly coupled sets of activities that possibly slowed the process of designing the hardware and firmware for the handset.

4.6.1 Partitioning

Partitioning of the 73 x 73 DSM of design tasks in the Hardware and Firmware Design phases resulted in the partitioned DSM shown in Figure 4.4.

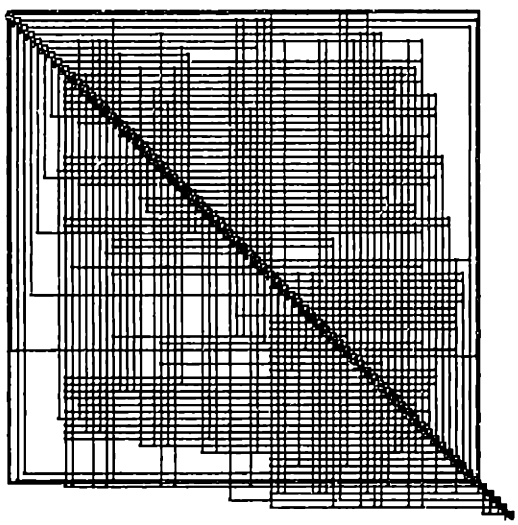


Figure 4.4: Partitioned DSM of the Hardware and Firmware Design Phases

All of the tasks contributed to the partitioning solution. The large coupled block located in the upper left corner of the diagram consisted of 69 tasks. The 4 tasks that were not coupled to the large block were: "SI: Integrate System Components", "MC: Boot Block Support", "MC: NAM Programming Changes", "ID: Industrial Design Release". With the exception of the last task, these activities were either tasks that required very little effort to complete (and therefore had little interaction with other tasks), or tasks that were performed in isolation from other tasks. In the latter case, many of the interviewees misinterpreted the task "ID: Tool Parts Available" as representative of Industrial Design's final deliverable, as opposed to "ID: Industrial Design Release", causing it to be coupled to very few tasks.

4.6.2 Controlling Features Analysis and Total Work Analysis

Controlling features and total work analysis was performed on the 69 x 69 coupled block of Hardware and Firmware Design tasks identified via partitioning. Figure 4.5, Figure 4.6, and Figure 4.7 show the top three ranked design modes as well as the top twelve ranked design tasks within each mode for the Hardware and Firmware Design phases.

Design Mode #1 was fairly cross-functional including tasks from Manufacturing, Analog Design, Industrial Design, and Supply Chain Management. The common element among these tasks was the acquisition of parts and manufacture of the analog printed circuit board (PCB). The analog PCB consisted of analog electronic components that were placed (or printed) on a circuit board with traced interconnections. Industrial Design was responsible for providing a Mechanical Computer Aided Design (MCAD) release to

Analog Design. The information in this release described the chassis dimensions and was heavily influenced by the component chassis test (Task #6). Without this information, the Analog Design group could not place/route the components on the circuit board and perform a design rule check. These tasks immediately preceded generating the artwork (solder stencil) for Manufacturing, preparing documents (bill of materials) for Supply Chain Management, and formatting the Electronic Computer Aided Design (ECAD) release for the circuit board supplier (all three activities encompassed in Task #2). Meanwhile, Manufacturing performed Design for Assembly (DFA) and Design for Manufacturing (DFM) reviews (Tasks #3 and #5) in preparation for the build. Finally, once Supply Chain Management had received the bill of materials (Task #7) and secured standard and custom parts (Tasks #10 and #12), and the circuit board had been received from the supplier, Manufacturing built preliminary prototypes (Task #4) and final engineering samples (Task #5). We called this design mode "Analog Printed Circuit Board Parts and Build". In identifying this design mode as the top ranked design activity that governed the rate and nature of convergence of the design effort, one of our recommendations to the sponsor company was to establish a cross-functional strategic design team centered around the radio frequency PCB. Recommendations are summarized in Chapter 7.

As a result of Digital Design's participation in the design mode (Task #8), we were inclined to include both the analog PCB and digital PCB in the design mode. Our decision to select the analog PCB solely was based on our observation in Chapter 3 using coupling analysis that there was a stronger reciprocal relationship between Analog Design and Manufacturing, as opposed to Digital Design and Manufacturing (see Section 3.6.2 for the coupling analysis). Also, although they were ranked high within the design mode, we did not consider Task #9 (Industrial Design: Component LCD Test), and Task #11 (Reliability: Board Test) as being particularly relevant to the mode.

Design Mode #2 involved tasks from Microcontroller Design, System Integration, and Digital Signal Processor Design. With the exception of Task #3 (System Integration: Develop User Interface Test Cases), the top eight ranked tasks were directly associated with Type 1 firmware. Type 1 was the software that exchanged messages with the base stations and enabled calls to be originated and terminated. Programming changes were required for code that adhered to an old standard, in order to adhere to a new standard (Task #2). To save power, the handset is powered down when it is not in use and it is not in its slot (Task #6). To establish service with the base station, a standard protocol is established for system determination which enables the handset to find a service provider (Task #8). Normally, the handset would have to perform an exhaustive search of the entire frequency space to establish such service. To accelerate this process a method call Roaming (Task #5) was devised, which uses a database to perform the search. One way to acquire this database is via Type 3 service provisioning, where the database is downloaded to the handset via the radio path (Task #7). Another way to obtain the database is by manufacturing it with the handset. Finally, the Microcontroller Design firmware release (Task #1) was strongly coupled to the Digital Signal Processor Design firmware release (Task #4) and layer 2 processes (Task #10) which perform error detection and retransmission of the signal to the base stations, and System Integration's development of Type 1 tests (Task #12). We called this design mode "Type 1 Design and Test". In identifying this design mode as the second ranked design mode, one of our recommendations to the sponsor company was to establish a cross-functional strategic design team centered around Type 1 firmware. Recommendations are summarized in Chapter 7.

Design Mode #3 included Reliability component-level and board-level tests (Tasks #1 and #3), and Manufacturing Design for Testing (DFT) reviews and post-build analysis (Tasks #2 and #4). It described feedback from Reliability to Manufacturing regarding process-related problems that were identified during testing. This design mode was not considered as strong as the previous two, and therefore did not result in a formal recommendation.

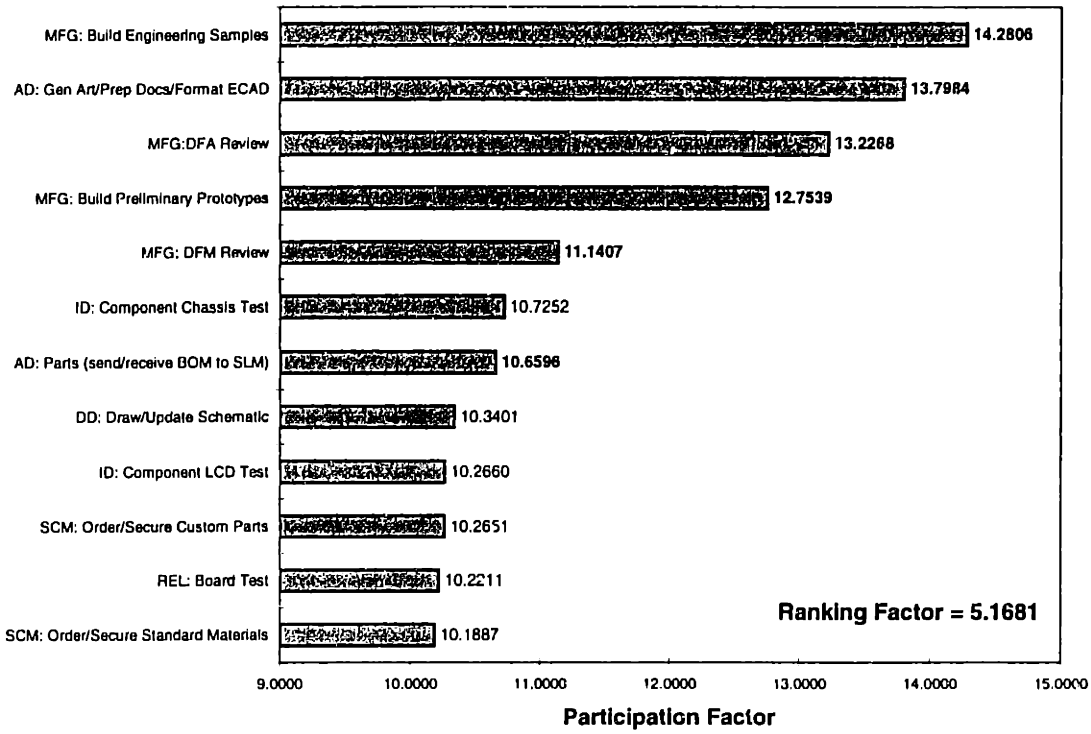


Figure 4.5: Hardware and Firmware Design Phase - Design Mode #1

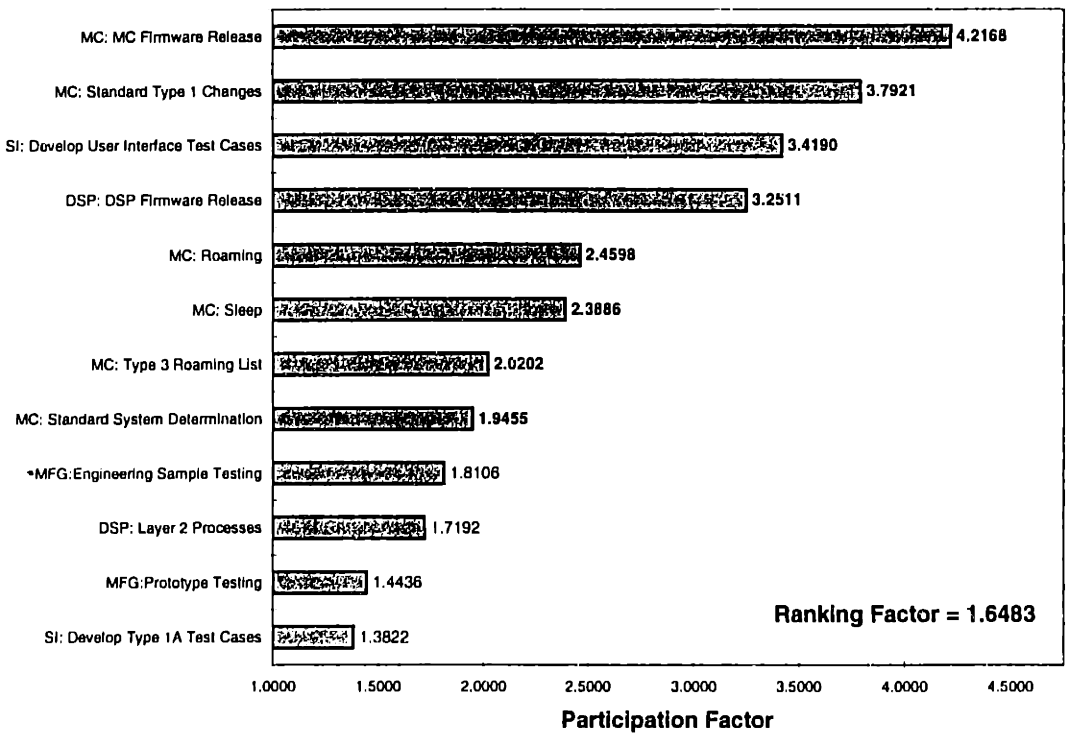


Figure 4.6: Hardware and Firmware Design Phase - Design Mode #2

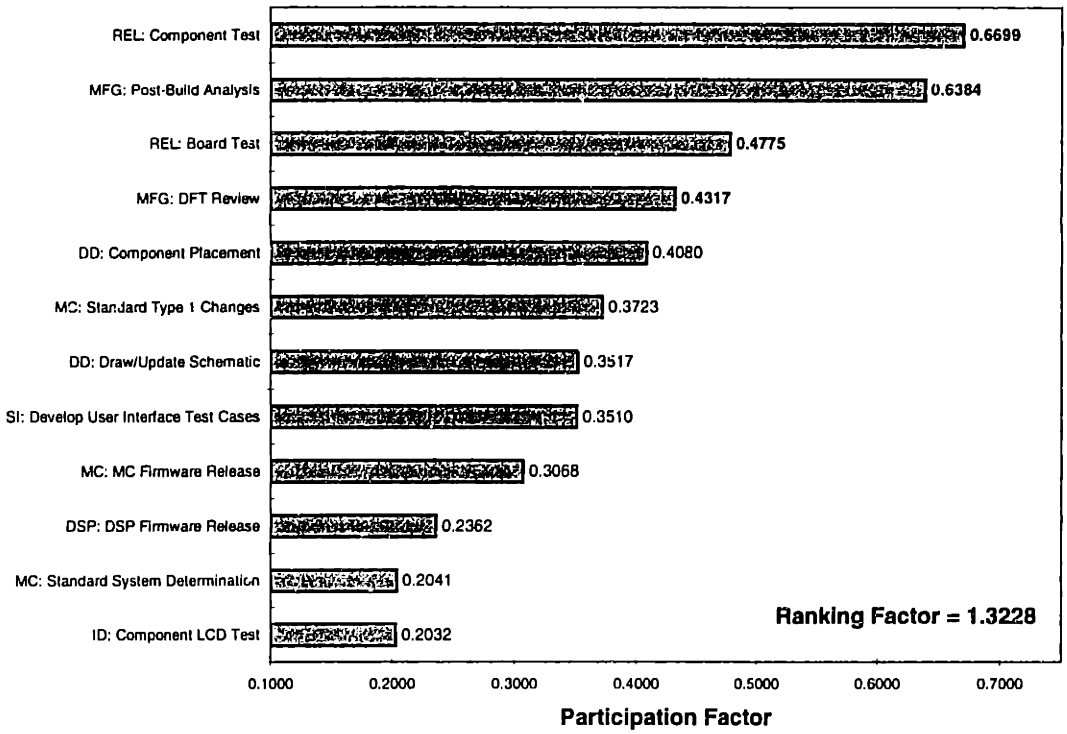


Figure 4.7: Hardware and Firmware Design Phase - Design Mode #3

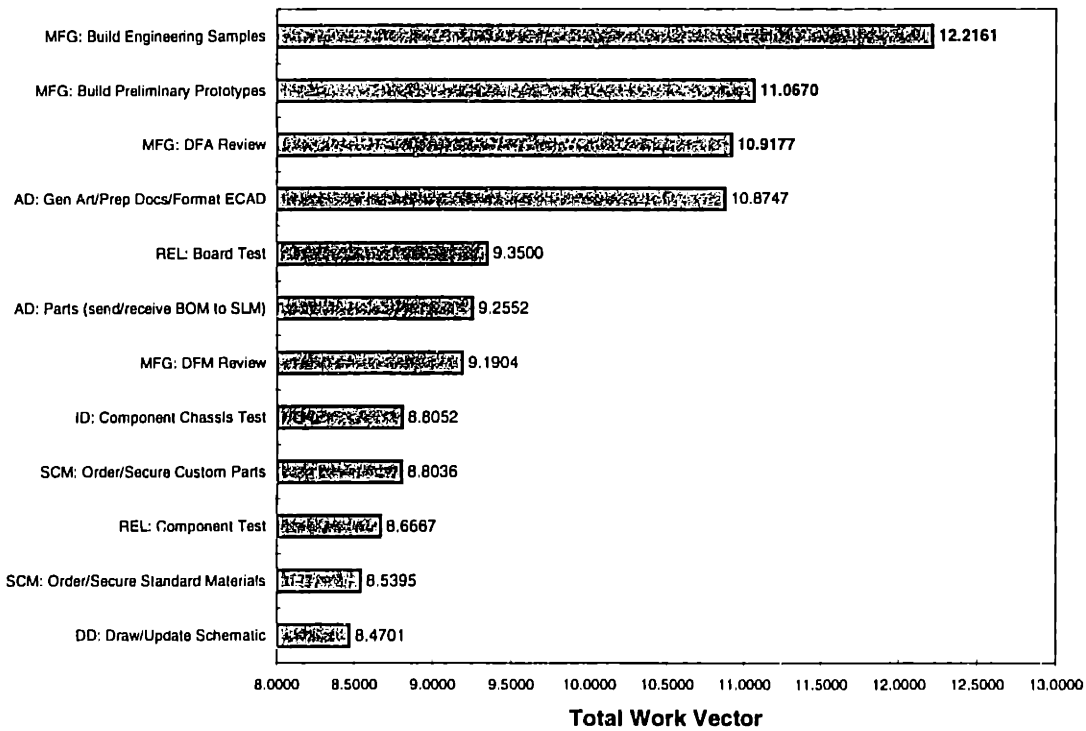


Figure 4.8: Hardware and Firmware Design Phase - Total Work Vector

We have previously mentioned that the sponsor company made a strategic decision to decouple hardware and firmware. As further evidence of its manifestation, both of the top ranked design modes were centered on one of these areas to the exclusion of the other. Design Mode #1 was specifically related to a hardware design activity. Design Mode #2 was specifically related to a firmware design activity. This was a useful observation from three perspectives. First, it once again confirmed the effective implementation of this strategy. Second, it provided evidence that our initial segmentation of tasks by process flow phase was successful in capturing parallel iteration. Third, it further verified the WTM model's ability to identify the main features of an actual design process.

Figure 4.8 shows the top twelve ranked design tasks in the total work vector for the Hardware and Firmware Design phases. According to the data, tasks performed by Manufacturing performed the most work. The top three ranked design tasks in the total work vector were all Manufacturing activities including building engineering samples and preliminary prototypes (Tasks #1 and #2). This underscored Manufacturing's importance in the design process. In fact, it was observed in the research environment that a delay in the manufacturing schedule was inevitably a corresponding delay in the product development process.

The total work vector rankings also supported earlier conclusions regarding the importance of the analog printed circuit board design mode. Tasks from this design mode reappeared in these rankings as twelve of the top thirteen design tasks. To accentuate the strength of its importance and the importance of hardware design, the highest ranked design task from the Type 1 design mode and firmware design was ranked 25th in the vector. All of the preceding tasks in the total work vector were hardware related.

4.7 Integration and Test Phases Analysis

The 33 tasks performed during each iteration of the Integration and Test Design phases were focused on integration and system-level testing of the hardware and firmware deliverables (engineering samples and firmware releases). Also conducted during these phases were additional environmental and user-related tests (see Section 2.6 for a detailed description of the process flow phases). In performing partitioning and controlling features/total work analysis on the tasks within these phases, we sought to identify strongly coupled sets of activities that possibly slowed the process of integrating and testing the different elements of the handset.

4.7.1 Partitioning

Partitioning of the 33 x 33 DSM of design tasks in the Integration and Test Design phases resulted in the partitioned DSM shown in Figure 4.9.

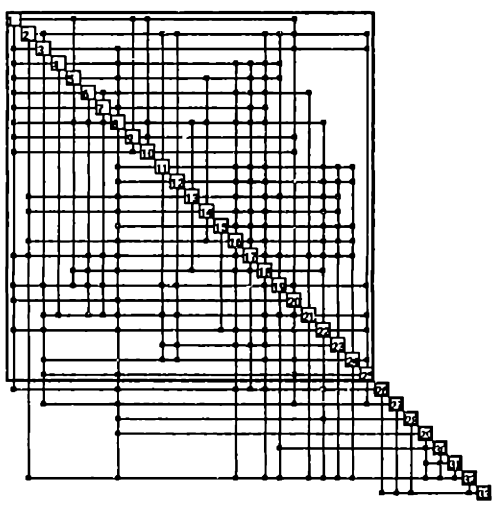


Figure 4.9: Partitioned DSM of the Integration and Test Phases

All of the tasks contributed to the partitioning solution. The large coupled block located in the upper left corner of the diagram consisted of 25 tasks. The 7 tasks that were not coupled to the large block were: "SI: System Integration Release", "SE: Regulatory Emission Test", "SE: Regulatory Scan", "SE: Regulatory Compliance Certification", "ST: System Test Release", "REL: Reliability Release", "SI: Execute Accessories Test Cases". All of these activities were either final deliverables that fed information back to earlier phases, or tasks that were performed in isolation from other tasks.

4.7.2 Controlling Features Analysis and Total Work Analysis

Controlling features and total work analysis was performed on the 25 x 25 coupled block of Integration and Test tasks identified via partitioning. Figure 4.10, Figure 4.11, and Figure 4.12 show the top three ranked design modes as well as the top ten ranked design tasks within each mode for the Integration and Test phases.

Design Mode #1 was considered indigenous to the System Integration group. The top three ranked tasks in the design mode were all associated with this group (Field Test Support, Execute Support Tools Test Cases, and Execute User Interface Test). However, we were unable to identify any underlying relationship between these activities.

Design Mode #2 characterized unit and system-level testing of the audio/acoustics system. This mode included tasks from Industrial Design, Reliability, Human Factors, and System Test. The audio/acoustics system on the handset included a microphone, receiver, and alerter. Essentially, each of these functional groups evaluated the audio/acoustics system from a different perspective. Industrial Design evaluated the performance (e.g. frequency response) of the system and its response to various environmental conditions (Task #1 and Task #3). Reliability also evaluated the system's response to environmental conditions (Task #2). Human Factors evaluated the subjective performance of the system to human listeners using Mean Opinion Scores (MOS), the overall performance of the handset (including audio/acoustics) in the field, and verified that the alerter was functioning appropriately (Task #4, Task #6, and Task #9). System Test also evaluated the performance of the system in a number of areas (e.g. frequency response, Mean Opinion Scores (MOS), and attenuation) (Task #5). Given these results there was a noticeable duplication of effort associated with many of these tests. In recognizing this redundancy, one of our recommendations to the sponsor company was to reduce the number of audio/acoustics related tests. Recommendations are summarized in Chapter 7.

Design Mode #3 was clearly associated with the handset's user interface. This mode included tasks from System Test, Human Factors, and System Integration. The user interface on the handset included a display (Liquid Crystal Display (LCD)), keypad/buttons, and audio/acoustics system. The user interface related tests were designed to verify proper functioning of the menus on the LCD and the alerter, in response to various selections on the keypad/buttons. Each of these functional groups played a role in evaluating the user interface. System Test performed tests that specifically evaluated the user interface (Task #1). Human Factors evaluated the alerter tones, the user interface itself, and the overall performance of the handset (including the user interface) in the field (Task #2, Task #3, and Task #4). Lastly, System Integration also evaluated the user interface itself (Task #6). Once again, given these results there was noticeable duplication of effort associated with many of these tests. In recognizing this redundancy, one of our recommendations to the sponsor company was to reduce the number of user interface related tests. Recommendations are summarized in Chapter 7.

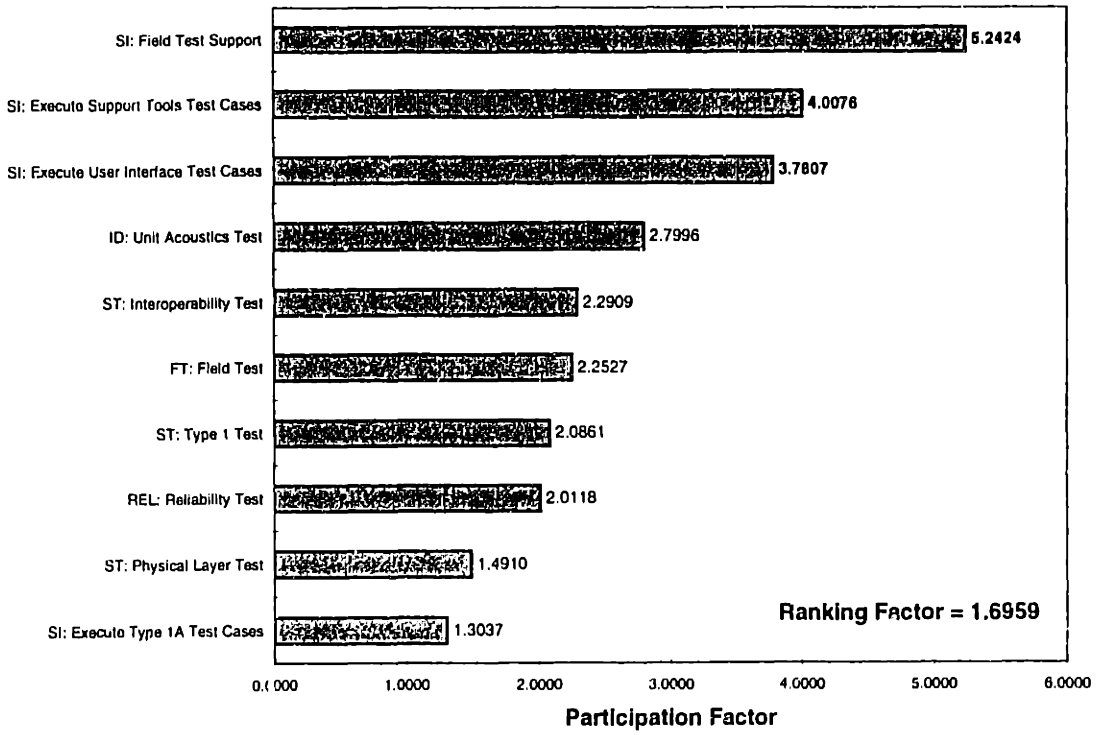


Figure 4.10: Integration and Test Design Phase - Design Mode #1

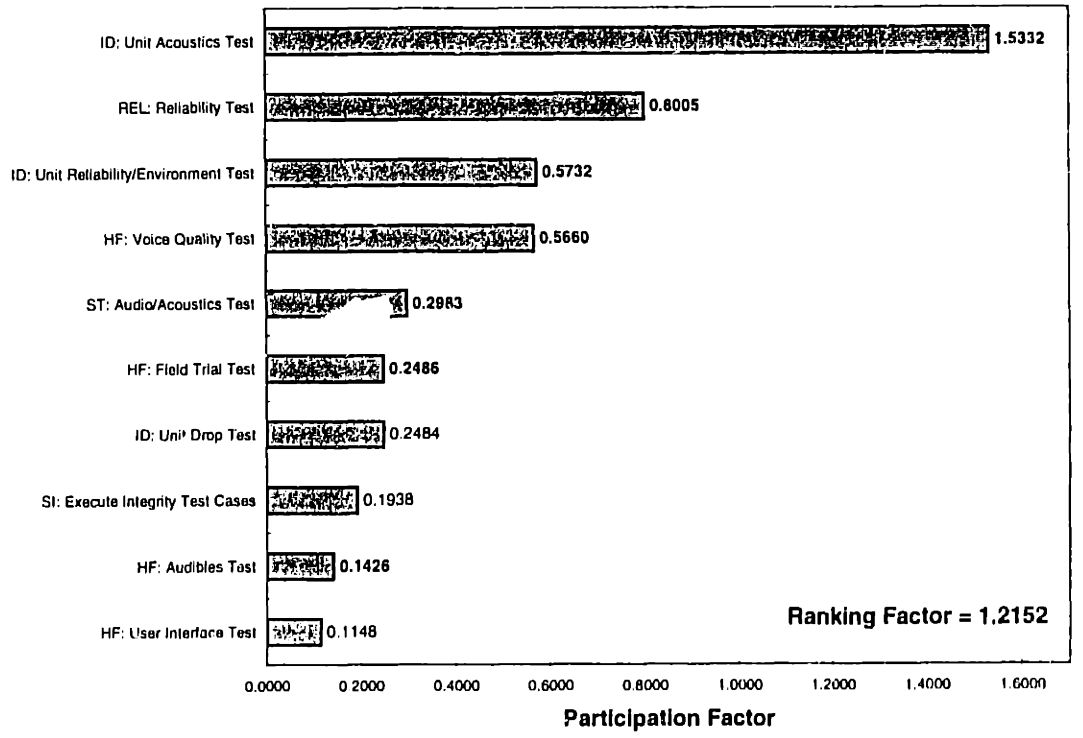


Figure 4.11: Integration and Test Design Phase - Design Mode #2

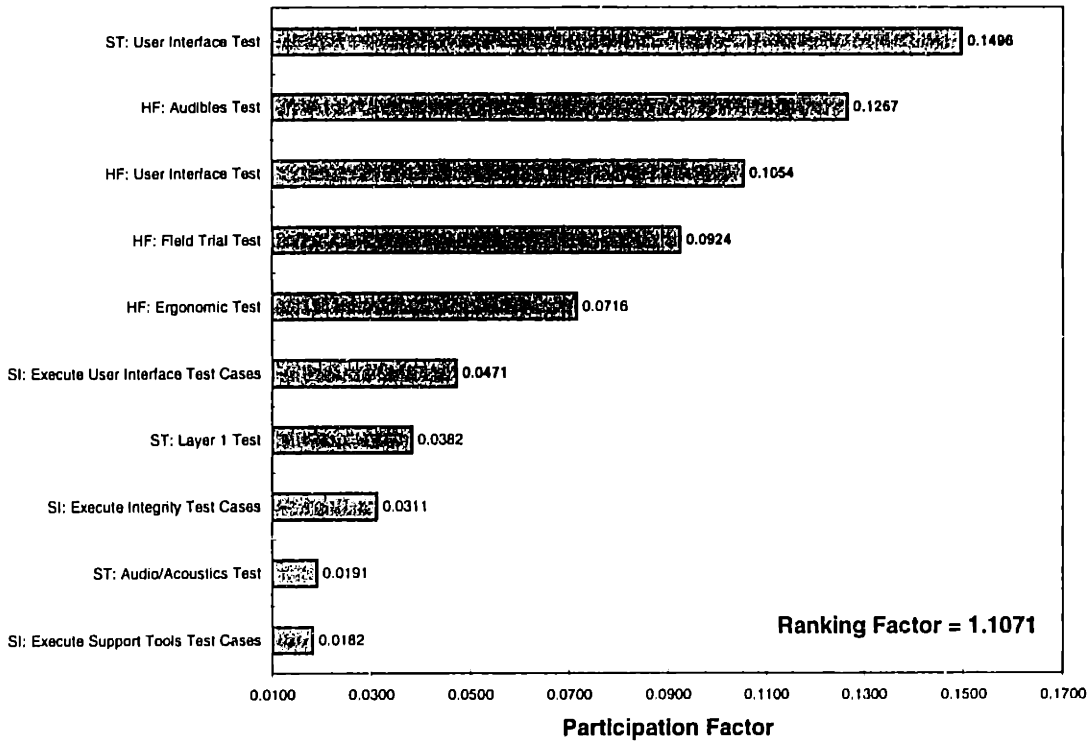


Figure 4.12: Integration and Test Design Phase - Design Mode #3

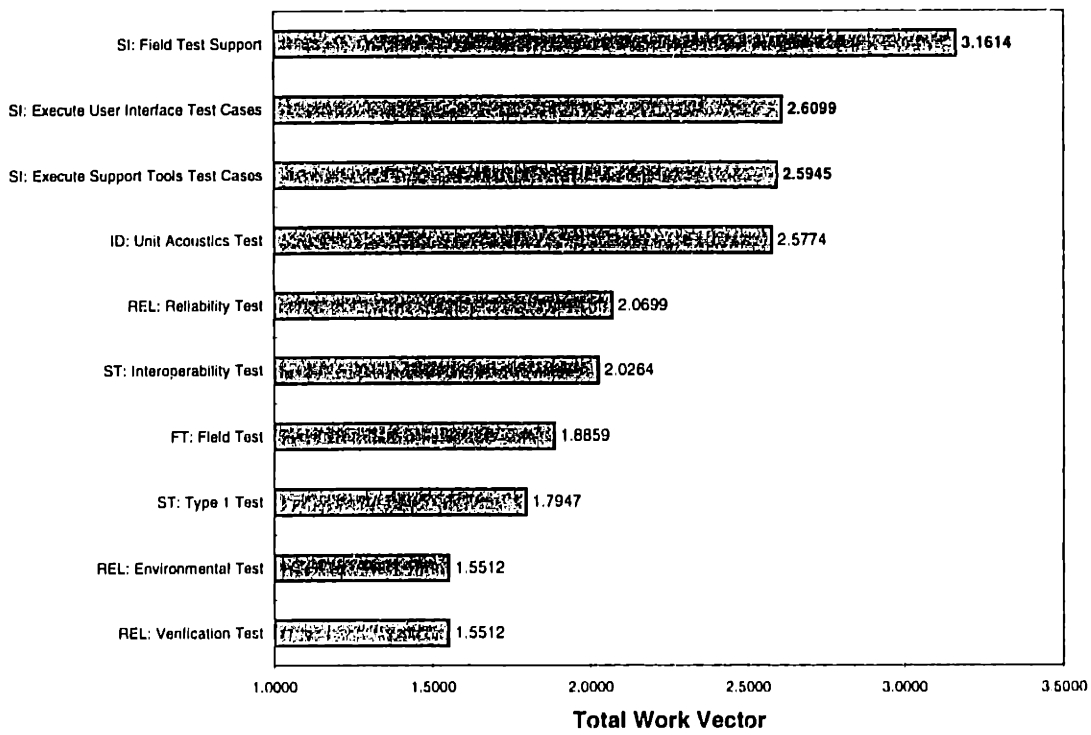


Figure 4.13: Integration and Test Design Phase - Total Work Vector

Figure 4.13 shows the top twelve ranked design tasks in the total work vector for the Integration and Test phases. According to the data, tasks conducted by System Integration performed the most work. The top three ranked design tasks in the total work vector were all System Integration activities including field test support (Task #1), execution of the user interface test cases (Task #2), and execution of support tools test cases (Task #3). We could have perhaps concluded that the completion of these activities was of critical importance to the integration and test process. However, there were two reasons why we considered this data to be inconclusive. First, none of the tasks were along the critical path. Second, the first and third tasks did not represent activities that were central to the handset's completion, but rather activities that supported the handset's development.

Generally speaking, the rankings in Design Mode #1 and the total work vector did not yield useful results. It is believed that this was a result of the fact that many of the design tasks in the Integration and Test phases were performed in relative isolation from one another, once operational hardware and/or firmware was delivered from the design phases. However, Design Mode #2 and Design Mode #3 were considered representative of a strong interrelationship between a number of design activities. It therefore appeared that our decision to group the tasks in the Integration and Test phases together, as opposed to separately, produced worthwhile results. As an exercise, controlling features and total work analysis was performed on the Integration and Test phases separately. Not surprisingly, the results yielded already recognizable groupings of tasks that were largely contained within the System Integration and System Test groups, for the Integration and Test phases respectively. Nonetheless, given these results and the results from Design Mode #1 and the total work vector, we concluded at minimum, that System Integration played a very important role in the design process.

4.8 Functional Group Analysis

In performing partitioning and controlling features/total work analysis on the fifteen functional groups of the group DSM, we sought to identify strongly coupled sets of functional groups that possibly slowed the overall product development process (see Section 2.5 for a detailed description of the functional groups). Our main objective here was to identify sets that corroborated earlier findings.

4.8.1 Partitioning

Partitioning of the 15 x 15 group DSM of functional groups resulted in the partitioned DSM shown in Figure 4.14.

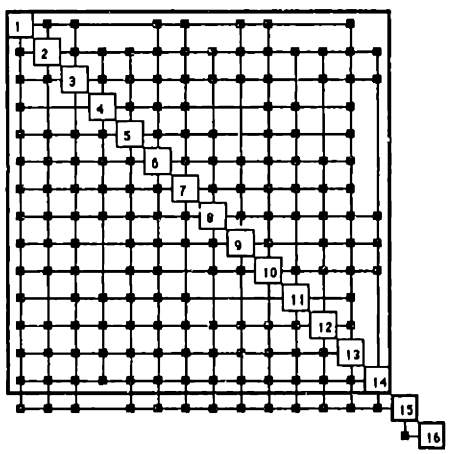


Figure 4.14: Partitioned Group DSM of Functional Groups

All of the tasks contributed to the partitioning solution. The large coupled block in the upper left corner of the diagram consisted of fourteen functional groups. The single functional group that was not coupled to the large block was Product Management. This was due to the fact that Product Management was represented by a single task which was not dependent on other tasks (a customer requirements document).

4.8.2 Controlling Features Analysis and Total Work Analysis

Controlling features and total work analysis was performed on the 14 x 14 coupled block of functional groups identified via partitioning. Figure 4.15, Figure 4.16, and Figure 4.17 show the top three ranked design modes as well as the top ten ranked functional groups within each mode.

Design Mode #1 revealed that System Integration and Microcontroller Design were very closely related functional groups. This also supported an earlier observation that was made in Chapter 3 using the DSM representation by functional regarding the need for strong communication between these groups (see Section 3.6.1 for the DSM representation analysis). This reinforced the importance of strong communication between these groups.

Design Mode #2 confirmed our earlier recommendation to form a cross-functional strategic design team around analog printed circuit boards. Here, where we analyzed relationships at the functional group level rather than the task level, the same functional groups represented by the tasks were identified as closely related including: Manufacturing, Industrial Design, Analog Design, Reliability, and Supply Chain Management.

Design Mode #3 did not yield useful information.

Figure 4.18 shows the top twelve ranked functional groups in the total work vector. These results once again underscored System Integration's and Manufacturing's importance to the design process, as both groups appeared at the top of the rankings.

4.9 Discussion

In this chapter, we performed Controlling Features and Total Work Analysis of the product development process at the sponsor company. Controlling features analysis identifies design modes, which are groups of closely related design tasks, while total work analysis identifies those design tasks that experience large amounts of iteration. We also provided an introduction to Partitioning, an analytical tool that uses the DSM framework to identify coupled sets of design tasks, and the Work Transformation Matrix (WTM), a DSM-based model that provides the foundation for controlling features and total work analysis, using the results of partitioning and eigenstructure analysis.

In performing our analysis, a few simplifying assumptions were made. The product development process at the sponsor company spanned several iterations of the Design, Integration, and Test phases. We modeled the process as if it consisted of a single global iteration of these phases, with various local iterations of the design tasks. We also relaxed some of the basic assumptions that underlie the WTM model such as parallel iteration and deterministic amounts of rework. Despite these measures, partitioning and controlling features/total work analysis demonstrated considerable robustness.

McCord and Eppinger describe the WTM tools as "suggestive rather than definitive", in that they provide a good starting point for reorganizing tasks as opposed to a well defined solution [18]. This was clearly the case in this research. One could argue that in our interpretation of the design modes we created "artificial" relationships between design activities. However, each of the design modes were presented to management at the sponsor company who confirmed the potential for their existence based on intuition and experience. Furthermore, our observations from coupling analysis were helpful in refining our interpretation of the design modes, while our results from eigenstructure analysis at the functional group level were helpful in corroborating our interpretation of the design modes. Both of these methods can serve as a supplement to the WTM model.

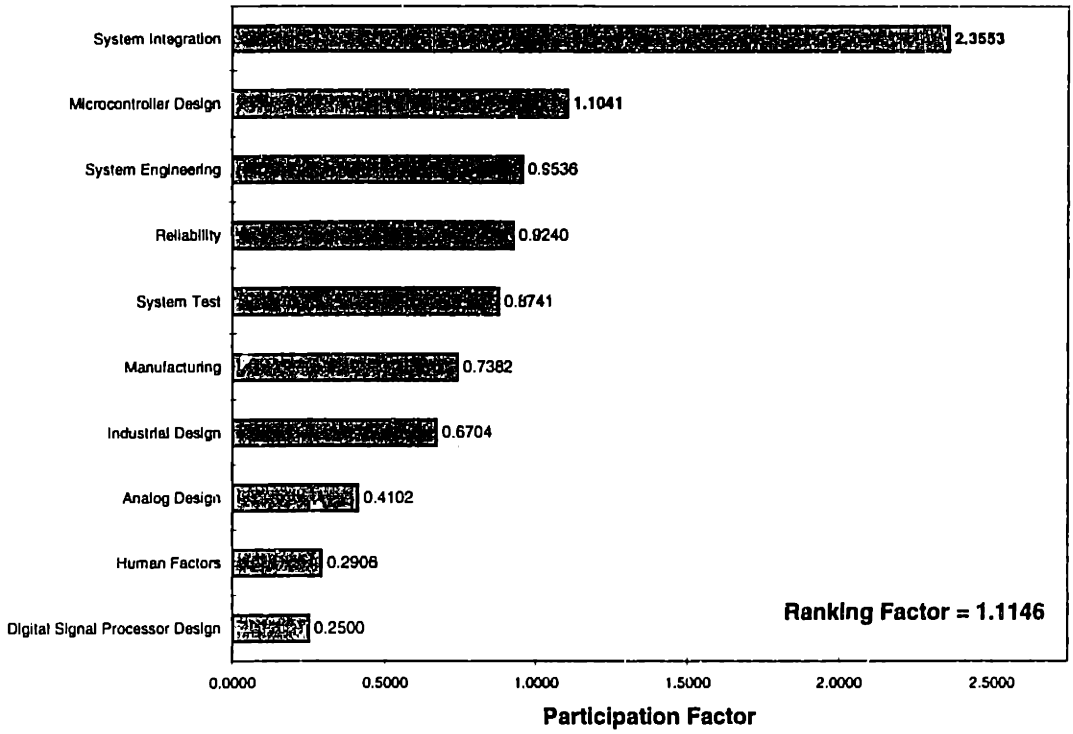


Figure 4.15: Functional Groups - Design Mode #1

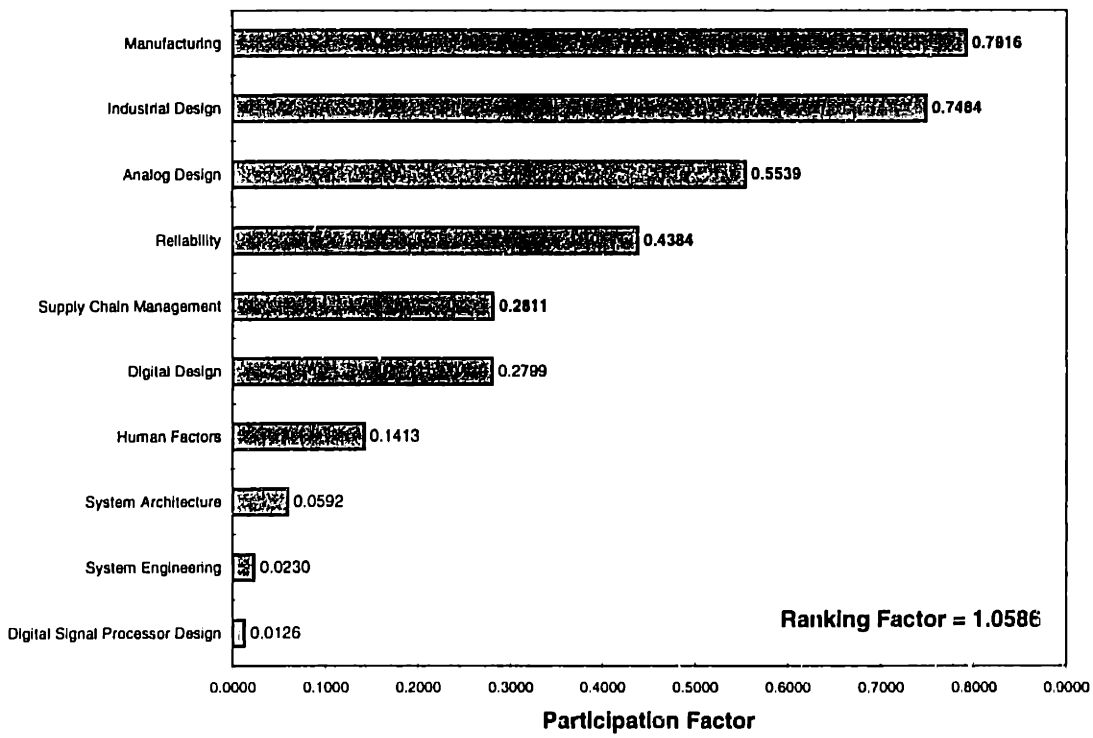


Figure 4.16: Functional Groups - Design Mode #2

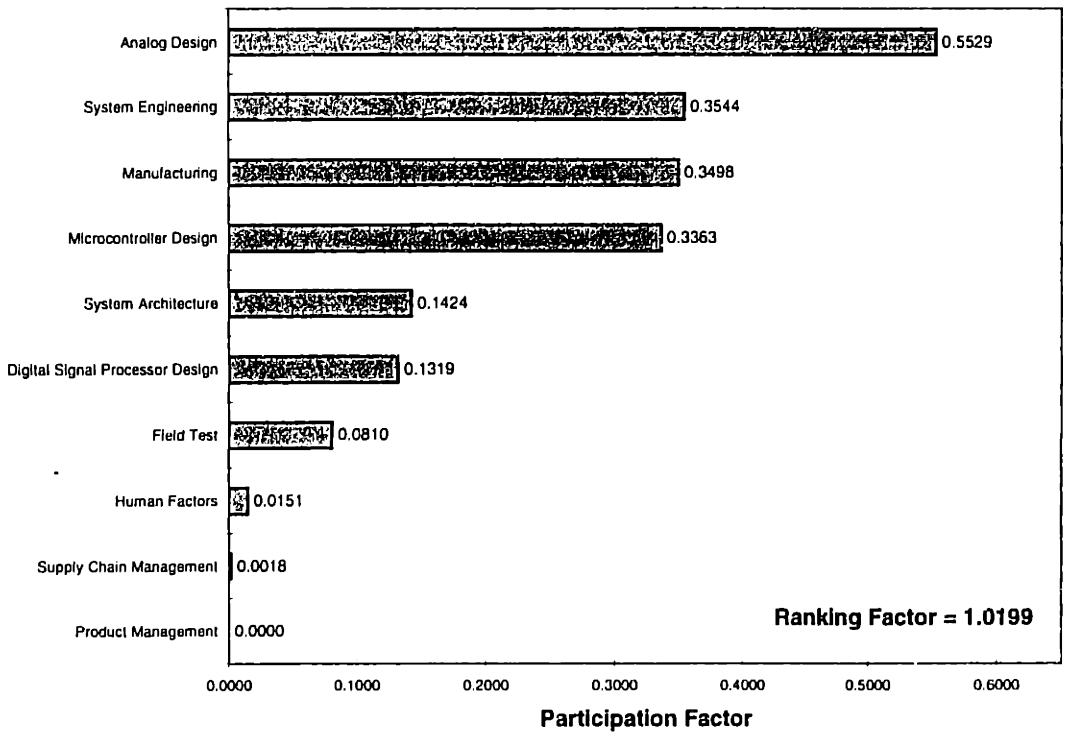


Figure 4.17: Functional Groups - Design Mode #3

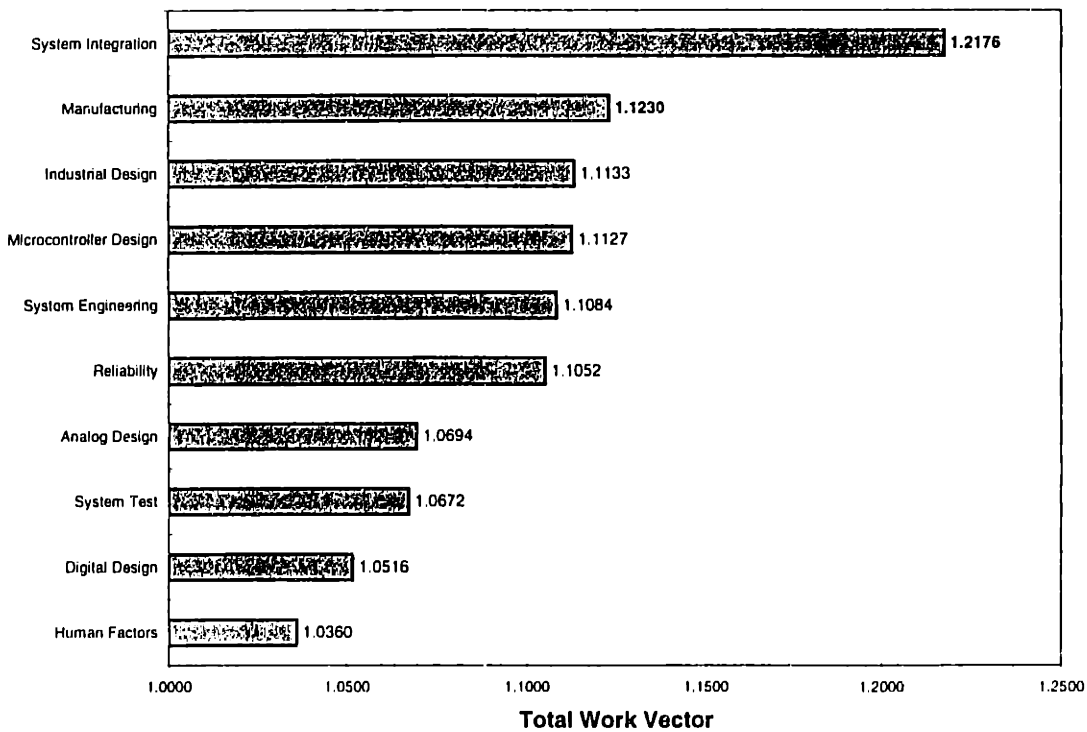


Figure 4.18: Functional Groups - Total Work Vector

To perform this analysis we used a Task-Level DSM whose elements consisted of design tasks. This was in contrast to a Parametric-Level DSM whose elements consist of technical parameters. As mentioned earlier, consistent with a rapid prototyping approach, both the hardware and firmware on the handset experienced increasing levels of functionality. These functions were easily captured by the task-level DSM in the firmware design groups, because design activities and handset functions were inextricably linked within these groups. For example, Caller Number Identification and Presentation (CNIP or caller ID) was both a design task in that programmers had to write and test CNIP code, and a handset function in that the CNIP feature was introduced at a specific point in the handset's evolution. On the other hand, they were not so easily captured in the hardware design groups where design activities stood separate from handset functions. For example, the ability to transmit a signal on the radio path is a function of the analog printed circuit board. This function was introduced at a specific point in the handset's evolution. However, the design tasks associated with this function range from drawing and updating a schematic to placing and routing components. While the task-level DSM was useful in providing insight to the product development process, a parametric-level DSM (where the parameters would be represented by the various functions) may have yielded even greater insight by possibly suggesting an optimal ordering for introducing the functions [37].

Nonetheless, in using the task-level DSM to perform controlling features and total work analysis, partitioning and the WTM model were shown to be powerful tools in generating recommendations for the sponsor company such as the reorganization of tasks and the establishment of strategic design teams.

CHAPTER 5: MODELING HARDWARE COMPLETION TIME

5.1 Modeling Hardware Development

Hardware denotes the physical electronic components of a device or apparatus. For a digital wireless telephone, hardware specifically refers to the *printed circuit board* (PCB) or *printed wiring board* (PWB) resident in the handset that includes analog and digital circuitry.

The design and manufacture of a PCB is a fairly well understood process. First, the circuit is designed typically using *computer-aided design* (CAD) tools that offer a library of electronic components such as resistors, capacitors, diodes, and integrated circuits (ICs). Using CAD tools, the circuit can be simulated and its performance can be evaluated and verified against specifications.

Second, using CAD and *computer-aided manufacturing* (CAM) tools, a simulation is performed that models the placement of components onto the target circuit board and the routing of interconnections between components. This is usually a challenge given the limited amount of space that is available on the circuit board, and the often unanticipated interactions that can occur between components that are within close proximity to one another.

Finally, once the blank circuit board (also called "unpopulated" circuit board) with interconnect traces is obtained, and the necessary components have been secured, the PCB is manufactured by physically placing the components and soldering their connections (e.g. pins or legs) onto the circuit board (also called "populating" the circuit board). Soldering is typically performed using one of two manufacturing approaches: *reflow soldering* and *wave soldering*.

Reflow soldering involves *surface mount technology* (SMT), where the leads of the components sit on pads on the circuit board [7]. A solder paste is applied to the pads which, when heated in a "reflow" oven, combines with the molten solder on the pads to form a joint with the surface mount devices. Wave soldering involves *through-hole technology*, and is the most economical means of mass soldering leaded components into holes on a circuit board [7]. The circuit board travels along a conveyor system. It is first passed over a flux station, then preheated, and finally passed over a "wave" of molten solder. The joints are formed between the component's leads and the pads, or plated through-holes of the board. Figure 5.1a shows a simplified hardware process flow diagram.

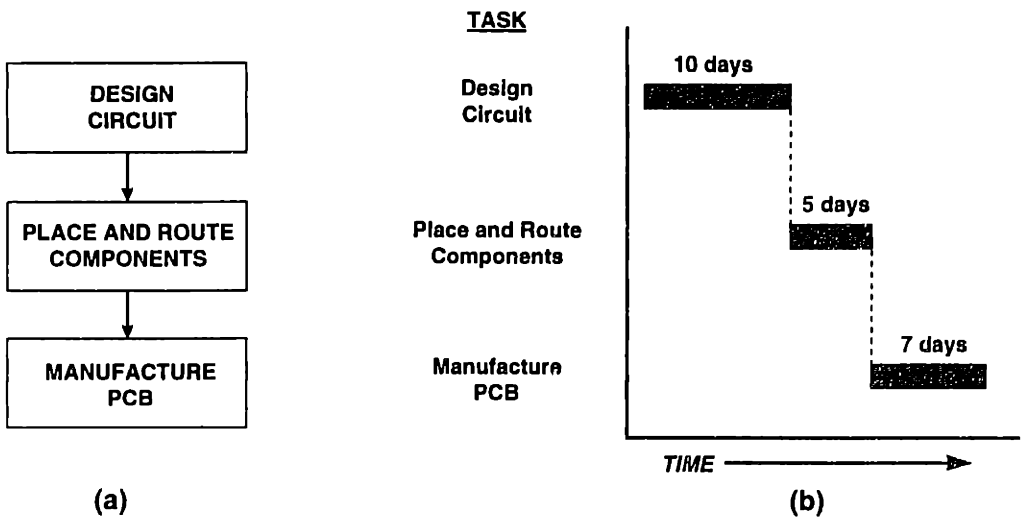


Figure 5.1: Simplified Hardware Process Flow Diagram (Sequential) and Project Management Representation

In theory, hardware development is a fairly sequential process. There is a series of tasks, each with a strong precedence relationship to their predecessors, that constitute the steps from start to finish. To model this process, a typical approach is to use traditional project management tools such as the Program Evaluation Review Technique/Critical Path Method (PERT/CPM) or GANTT charts. PERT/CPM charts depict tasks, task durations, and task dependencies, where the set of dependent tasks that together require the longest amount of time (the "critical path") are indicated in the chart [21]. A GANTT chart is a representation of PERT/CPM in a matrix form where the vertical axis lists all of the tasks to be performed and the horizontal axis is headed by columns that indicate the estimated duration for each task, skill requirements, and the names of the individuals(s) assigned to the task [21]. Within the matrix itself are horizontal bars that connect the starting period and ending period for each task. Figure 5.1b shows a simplified project management representation of the hardware process flow diagram (note that we assume task times of 10 days, 5 days, and 7 days, for designing the circuit, placing and routing the components, and manufacturing the PCB, respectively, this is done for later discussion).

In practice, hardware development is a sequentially iterative process. Information obtained during downstream tasks, can typically cause the repetition of upstream tasks. As an example, the following conditions could cause iteration: 1) while placing and routing the components an engineer discovers an incompatibility between two components, causing the circuit to be redesigned, 2) while manufacturing the PCB a technician determines that the spacing between two components is too close to perform soldering, causing the components to be placed and routed again. Figure 5.2 depicts these conditions for the simplified hardware process flow diagram shown in Figure 5.1 (note the following: 1) the tasks are lettered A, B, and C, and numbered 1 through 5 including a start task and a finish task, and 2) we assume a probability of 0.3 for the first iteration condition and a probability of 0.1 for the second iteration condition, this is done for later discussion).

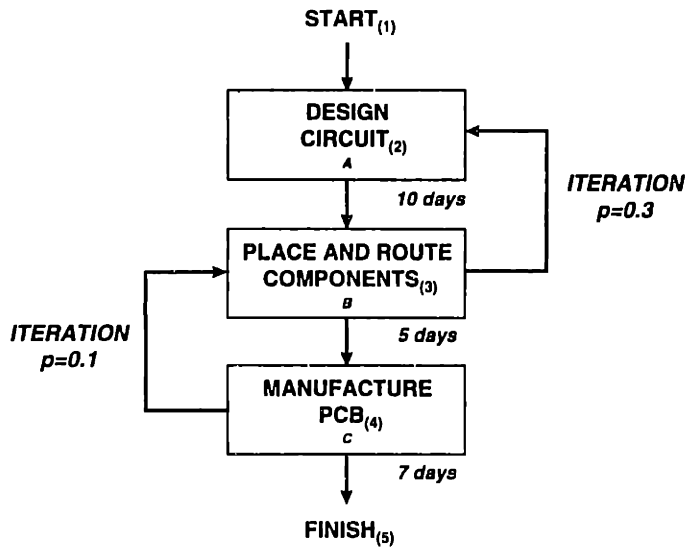


Figure 5.2: Simplified Hardware Process Flow Diagram (Sequential Iteration)

Despite their usefulness, tools such as PERT/CPM and GANTT are no longer applicable given the presence of iteration and its potential impact on completion time. Consequently, there exists a need for more flexible project management tools. Two such tools that have been used to model sequential iteration are the Signal Flow Graph and the Reward Markov Chain (or Reward Markov Model). The signal flow graph is typically used in electrical engineering and is used for circuit and system analysis to model discrete event systems [11]. The reward Markov chain is typically used in operations management, manufacturing, reliability, and telecommunications network and computer modeling to conduct queuing and system analysis for performance evaluation [5].

In this chapter, we use the signal flow graph and the reward Markov chain to model hardware completion. First, we describe each of these tools in detail, and then we present the results from applying them to one

of the PCBs at the sponsor company. We conclude with a discussion of the strengths and weaknesses of both models.

5.2 Signal Flow Graph

A signal flow graph is a network of directed *branches* that connect at *nodes* [11, 22]. Branch *jk* denotes a branch originating at node *j* and terminating at node *k*, with the direction from *j* to *k* being indicated by an arrowhead on the branch, as shown in Figure 5.3.

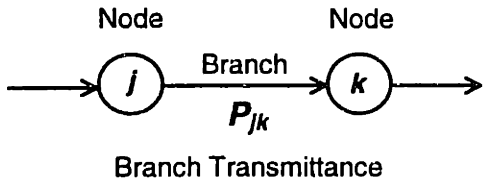


Figure 5.3: Signal Flow Graph Nodes, Branches, and Branch Transmittance

In modeling hardware completion time, the branches represent tasks, while the nodes serve to connect related tasks. Associated with each branch is a branch transmittance P_{jk} that includes the transition probability represented by the branch p_{jk}, the task time represented by the branch t_{jk}, and the transform variable z, as shown in Equation 5.1.

$$P_{jk} = p_{jk} z^{t_{jk}} \quad \text{where} \quad \begin{array}{l} p_{jk} = \text{transition probability} \\ t_{jk} = \text{task time} \end{array}$$

Equation 5.1

The Laplace transform is a generalization of the Fourier transform in the continuous-time domain. In similar fashion, the z-transform is a generalization of the Fourier transform in the discrete-time domain. The z-transform plays an important role in the analysis and representation of discrete-time systems. Eppinger, Nukala, and Whitney write [11]:

> *"[The z-transform is] used to connect the physical system (time domain) to the quantities used in the analysis (transform domain). The z-transform simplifies the algebra, as it enables us to incorporate the quantities to be multiplied ([transition] probabilities) in the coefficient of the expression, and to include quantities to be added (task times) in the exponent. The resulting system is then analogous to a discrete sampled data system, and the body of literature on this subject can be applied for the analysis thereof."*

As an example, Figure 5.4 shows a signal flow graph representation of the hardware process flow diagram shown in Figure 5.2 (using the numbered convention 1 through 5).

We define the start node, or input node, as $X(z)$. We define the finish node, or output node, as $Y(z)$. The system function, or transfer function $H(z)$, is defined as the input-output relationship between $X(z)$ and $Y(z)$, as shown in Equation 5.2.

$$H(z) = \frac{Y(z)}{X(z)}$$

Equation 5.2

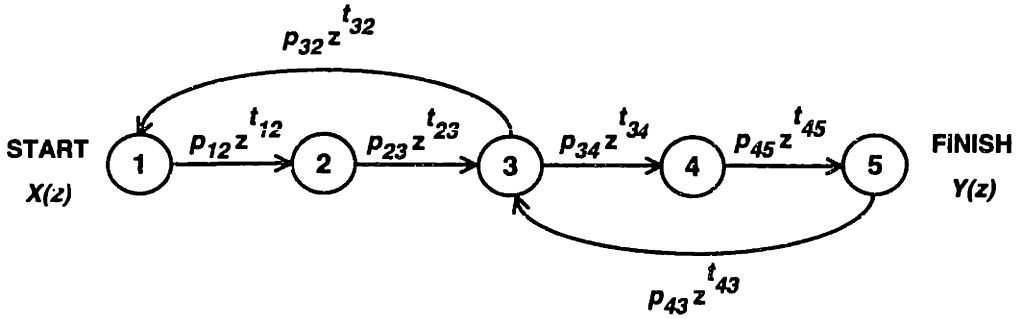


Figure 5.4: Signal Flow Graph Representation of the Hardware Process Flow Diagram

The behavior of the signal flow graph network is captured by the transfer function $H(z)$. The impulse response is defined as the system response to a unit-sample sequence. In modeling hardware completion time, it is also a function representing the probability distribution of the completion time for the network. Therefore, by determining the transfer function, we can determine the completion time.

5.2.1 Completion Time

To determine the completion time we must determine the transfer function $H(z)$. To analyze a signal flow graph and determine its transfer function, we can use standard operations on signal flow graphs [11] and block diagram algebra [6]. Another useful technique for determining the transfer function of a signal flow graph, particularly one with several feedback loops, is *Mason's Gain Formula* [6]. *Mason's Gain Formula* states that the transfer function $H(z)$ from any input node $X(z)$ to any output node $Y(z)$ can be determined using the relation and definitions shown in Equation 5.3.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{\Delta}\sum_{i=1}^{n} G_i \cdot \Delta_i$$

where Δ = 1 - (sum of all loop gains) + (sum of products of loop gains over all sets of *two* nontouching loops) - (sum of products of loop gains over all sets of *three* nontouching loops) + ...

G_i = gain of the *ith* forward path (i = 1, 2, ..., n) from input node $X(z)$ to output node $Y(z)$

Δ_1 = value of Δ if all loops that touch that *ith* forward path from $X(z)$ to $Y(z)$ are excepted

and *Forward Path* is a simple, continuously directed path leading from a node X to a node Y.
Loop is a forward path where the nodes X and Y coincide.
Gain is the product of the transfer functions over all branches comprising a forward path or loop.
Nontouching denotes a set of loops such that no two of the loops have a node in common.

Equation 5.3

We can calculate the expected value of the completion time $E[T]$ by differentiating the transfer function $H(z)$, and substituting $z=1$, as shown in Equation 5.4.

$$E[T] = \left.\frac{dH(z)}{dz}\right|_{z=1}$$

Equation 5.4

This result is possible due to the properties of the expression for the branch transmittance. The terms of the transfer function are of the form $p_{jk}z^{t_{jk}}$. Taking the derivative, the terms are now of the form $p_{jk}t_{jk}z^{t_{jk}-1}$. Substituting $z=1$, the terms are now of the form $p_{jk}t_{jk}$. The sum of these terms (paths) represents the expected value of the completion time for the process. Similarly, we can calculate the standard deviation of the completion time $STDEV[T]$, as shown in Equation 5.5.

$$STDEV[T] = \sqrt{\left.\frac{d\left(z\frac{dH(z)}{\partial z}\right)}{dz}\right|_{z=1} - (E[T])^2}$$

Equation 5.5

To calculate these derivatives, we can often use the quotient rule as shown in Equation 5.6 [41].

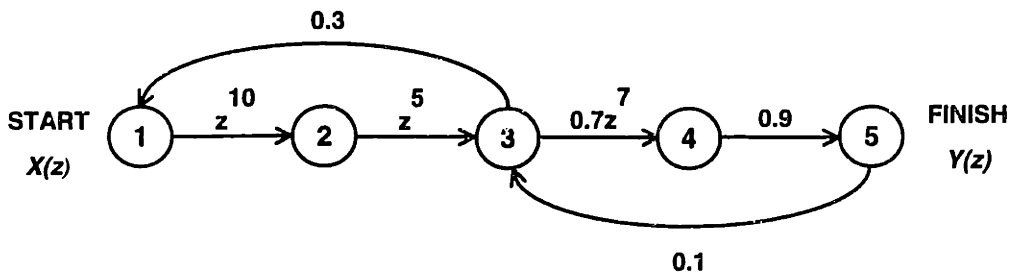
$$\frac{dH(z)}{dz} = \frac{d}{dz}\left(\frac{Y(z)}{X(z)}\right) = \frac{X(z)\frac{dY(z)}{dz} - Y(z)\frac{dX(z)}{dz}}{X(z)^2}$$

Equation 5.6

Finally, given the transfer function, we can determine the Probability Distribution Function (PDF) and the Cumulative Distribution Function (CDF) of the completion time using any number of methods. These methods include monte carlo simulation of the signal flow graph network or polynomial division of the transfer function.

Using the signal flow graph, similar distributions can be generated for time and cost [4]. This is not performed here. Furthermore, the driving factors of an iterative process can also be identified for a signal flow graph using eigenstructure decomposition [11, 24]. This analysis is almost identical to the identification of design modes that was performed during controlling features analysis in Chapter 4. This is also not performed here.

As an example, we assume that the hardware development process shown in Figure 5.2 and Figure 5.4 is modeled with task times t_{12} = 10, t_{23} = 5, and t_{34} = 7, and transition probabilities p_{32} = 0.3, p_{34} = 0.7, p_{43} = 0.1, and p_{45} = 0.9. All of the remaining task times and transition probabilities are zero. These values are interpreted as follows: circuit design requires 10 days, placing and routing the components requires 5 days, manufacturing the PCB requires 7 days, there is a probability of 0.3 that we will have to repeat circuit design after placing and routing the components, and there is a probability of 0.1 that we will have to repeat placing and routing the components after manufacturing the PCB. The signal flow graph representation is as follows:



Mason's Gain Formula from Equation 5.3, is applied as follows:

$$\Delta = 1 - (z^{10} \cdot z^5 \cdot 0.3 + z^5 \cdot 0.7z^7 \cdot 0.1) = 1 - 0.07z^{12} - 0.3z^{15}$$

$$G_1 = z^{10} \cdot z^5 \cdot 0.7z^7 \cdot 0.9 = 0.63z^{22}$$

$$\Delta_1 = 1$$

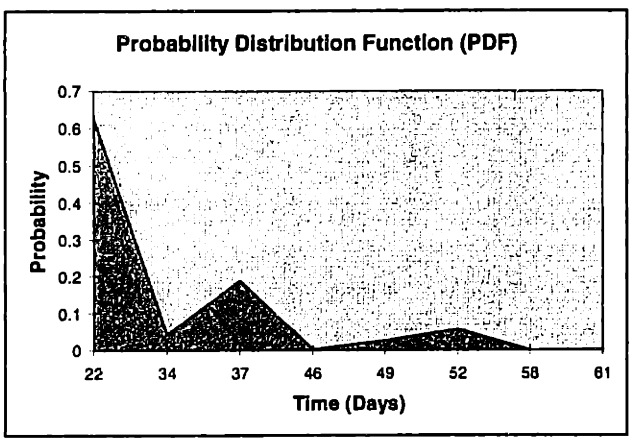
Finally, the transfer function *H(z)* is as follows:

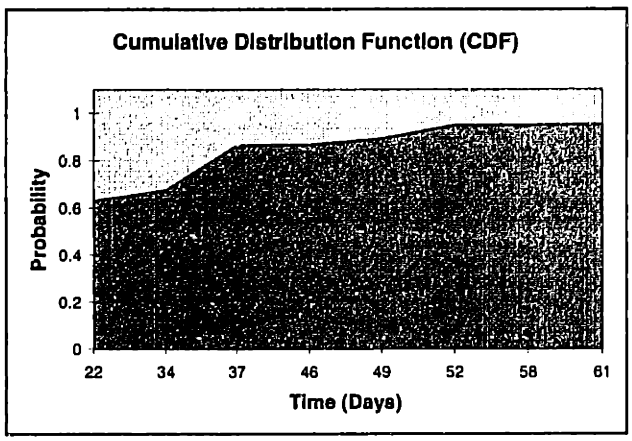
$$H(z) = \frac{Y(z)}{X(z)} = \frac{0.63z^{22}}{1 - 0.07z^{12} - 0.3z^{15}}$$

Using Equation 5.4 and Equation 5.5, the expected value of the completion time *E[T]* and the standard deviation of the completion time *STDEV[T]* are as follows:

$$E[T] = 30.5 \text{ days} \qquad STDEV[T] = 14.0 \text{ days}$$

We conclude that the expected completion time for the hardware process is 25.2 days, with a variance of 5.2 days. Performing polynomial division of the transfer function *H(z)*, the Probability Distribution Function (PDF) and the Cumulative Distribution Function (CDF) of the completion time are as follows:





Appendix C shows MATLAB code that implements the signal flow graph calculations performed in this example (expected value of the completion time, variance of the completion time (using the quotient rule in Equation 5.6 to perform differentiation), and polynomial division of a transfer function to return the PDF and CDF).

5.2.2 Sensitivity Analysis

The sensitivity of the completion time T with respect to each parameter l (task times t_{jk} and transition probabilities p_{jk}), is defined as shown in Equation 5.7.

$$S_l^T = \frac{\Delta E[T] / E[T]}{\Delta l / l}$$

Equation 5.7

We can use this sensitivity analysis to evaluate the effect of changes to the task times and transition probabilities on the completion time. This can be useful in identifying tasks that may require special attention, assessing risk, or determining which tasks if reduced (or increased) have the greatest potential to reduce (or increase) the completion time. Continuing our previous example, using Equation 5.7, the sensitivities of the completion time with respect to the task times and transition probabilities are ranked as follows:

Time	Sensitivity
t_{12}	0.4844
t_{34}	0.2604
t_{23}	0.2552

Probability	Sensitivity
p_{32}	0.3363
p_{43}	0.0747
p_{45}	-0.6652
p_{34}	-0.7735

A positive sensitivity denotes a parameter that if increased, will increase the completion time A negative sensitivity denotes a parameter that if increased, will reduce the completion time. Two conclusions can be drawn from this sensitivity analysis. First, a reduction in the task time for circuit design (t_{12} = 10) will yield the largest reduction in the completion time, relative to the other task times (sensitivity = 0.3363). Second, an increase in the probability of manufacturing the PCB (p_{34} = 0.7), without having to repeat circuit design, will yield the largest reduction in the completion time, relative to the other transition probabilities (sensitivity = -0.7735).

Appendix D shows MATLAB code that implements the signal flow graph sensitivity analysis performed in this example.

5.3 Reward Markov Chain

A reward Markov chain is comprised of *states (or nodes)* and *stages* [5, 36]. A new stage is begun when a state is entered for the first time. A reward R_i is associated with each state i, and a state variable S_{jk} is associated with the process of moving from state j to state k, as shown in Figure 5.5.

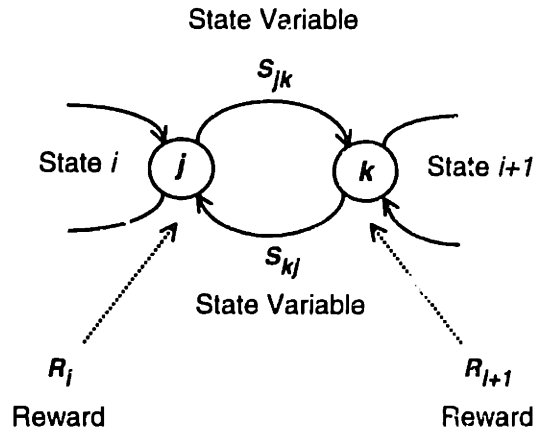


Figure 5.5: Reward Markov Chain States, Rewards, and State Variables

The state variable can be assigned in one of two ways - by associating a rate with state occupancies or by associating an impulse with state transitions. The former represents a continuous-time reward Markov chain, while the latter represents a discrete-time reward Markov chain. Here, we opt for the latter as a means to capture the probabilistic nature of the process we seek the model. Jawad and Johnsen write [14]:

"[Many] systems are probabilistic in their behavior over time, therefore stochastic processes are used to model their behavior. Stochastic processes are 'mathematical models of systems which vary in time in a random manner.' Markov process representation, a type of stochastic process, allows the consideration of time in a very controlled manner. The Markov property is such that 'given the present state, the past states have no influence on the future.' So, in general Markov chains are used, which are 'discrete parameter Markov processes whose state space is finite or countably infinite'."

In modeling hardware completion time, the states correspond to a task within each stage of the process. A new stage is begun the first time a task is attempted. There are as many states in each stage as there are tasks which may be undertaken in that stage, and this number increases by one for each new stage [36]. The state variables correspond to the transition probabilities p_{jk} that if task j is attempted, that task k will have to be repeated. The reward of each state captures the amount of time spent in each state, or the time spent performing the associated task, and is a function of the task times t_i.

For purposes of our analysis, the matrix M will represent the reward Markov chain by storing the task times t_i in the on-diagonal positions and the transition probabilities p_{jk} in the off-diagonal positions, for the process we seek to model. An empty entry in an off-diagonal position denotes a null probability. The information contained in this matrix is almost identical to the information contained in the Numerical Design Structure Matrix (NDSM) described in Chapter 2 and the Work Transformation Matrix (WTM) described in Chapter 3, only with a slightly different interpretation. Here, the off-diagonal entries denote the probabilities associated with tasks that are coupled, as opposed to coupling strengths between tasks (NDSM) or amounts of rework resulting from tasks that are coupled (WTM). As an example, the hardware process flow diagram shown in Figure 5.2, would be captured as follows (using the lettered convention A through C):

$$M = \begin{bmatrix} t_a & p_{ba} & \\ p_{ab} & t_b & p_{cb} \\ & p_{bc} & t_c \end{bmatrix}$$

In order for a reward Markov chain interpretation to be valid, the off-diagonal values (transition probabilities) in any single column of M can add up to no greater than one. Figure 5.6 shows the corresponding reward Markov chain representation.

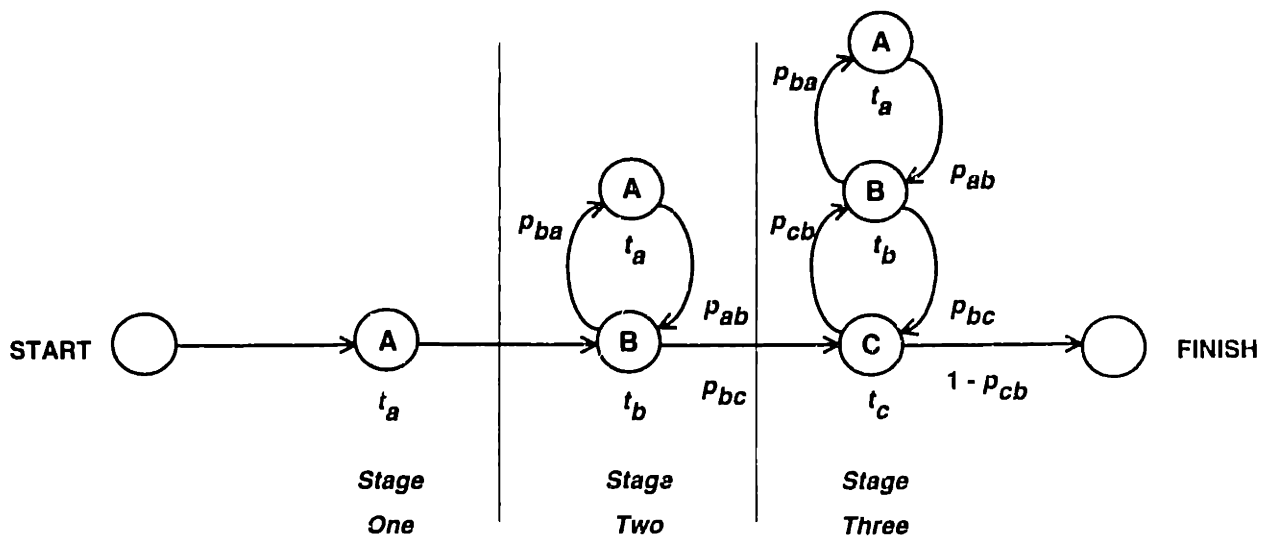


Figure 5.6: Reward Markov Chain Representation of the Hardware Process Flow Diagram

The expected reward of the Markov chain is the expected length of time that the process will spend in the stages of the chain. By calculating this length, we determine the completion time.

5.3.1 Completion Time

To determine the completion time we must determine the expected reward of the Markov chain. We begin at the finish end of the chain and work backwards across the stages. The total expected time is determined by summing the expected length of time that the process will spend in each stage, given its initial state. The $n \times n$ matrix M representing the reward Markov chain corresponds to $(n^2 + n) / 2$ states. To determine the total expected time through the entire network, one would typically have to solve $(n^2 + n) / 2$ simultaneous equations. Smith and Eppinger present a calculation procedure which only involves solving n simultaneous equations. Algebraically, the calculation is similar to Gaussian elimination (since the expected time can be written as a set of linear equations), although the back substitution is atypical [36]. Their *Efficient Length Computation Algorithm* is presented here [36].

Again, the matrix M stores the task times t_i in the on-diagonal positions and the transition probabilities p_{jk} in the off-diagonal positions, for the process we seek to model. First, we place the negative transpose of the $n \times n$ matrix M, in an $n \times n$ matrix P, inserting ones in the on-diagonal positions, as shown in Equation 5.8.

$$P_{ij} = \begin{cases} -M_{ji} & i \neq j \\ 1 & i = j \end{cases} \qquad \text{where} \quad \begin{matrix} i = 1 \ldots n \\ j = 1 \ldots n \end{matrix}$$

Equation 5.8

The matrix P describes the reward Markov chain. Second, we place the task times in the off-diagonal positions of M in a column vector B, as shown in Equation 5.9.

$$B_i = M_{ii} \qquad \text{where} \quad i = 1 \ldots n$$

Equation 5.9

Third, using Gaussian elimination we extract the unique lower triangular, diagonal, and upper triangular matrices of the matrix P, such that the relation shown in Equation 5.10 holds true (*diagonalization*).

$$P = LDU \qquad \text{where} \quad \begin{matrix} L = \text{lower triangular matrix} \\ D = \text{diagonal matrix} \\ U = \text{upper triangular matrix} \end{matrix}$$

Equation 5.10

Fourth, using modified back substitution we calculate the vector X', which contains the total expected time spent in each stage of the reward Markov chain, as shown in Equation 5.11.

$$X' = D^{-1}L^{-1}B$$

Equation 5.11

Finally, we add the elements in the vector X' to get the completion time T, as shown in Equation 5.12.

$$T = \sum_{i=1}^{n} X'_i$$

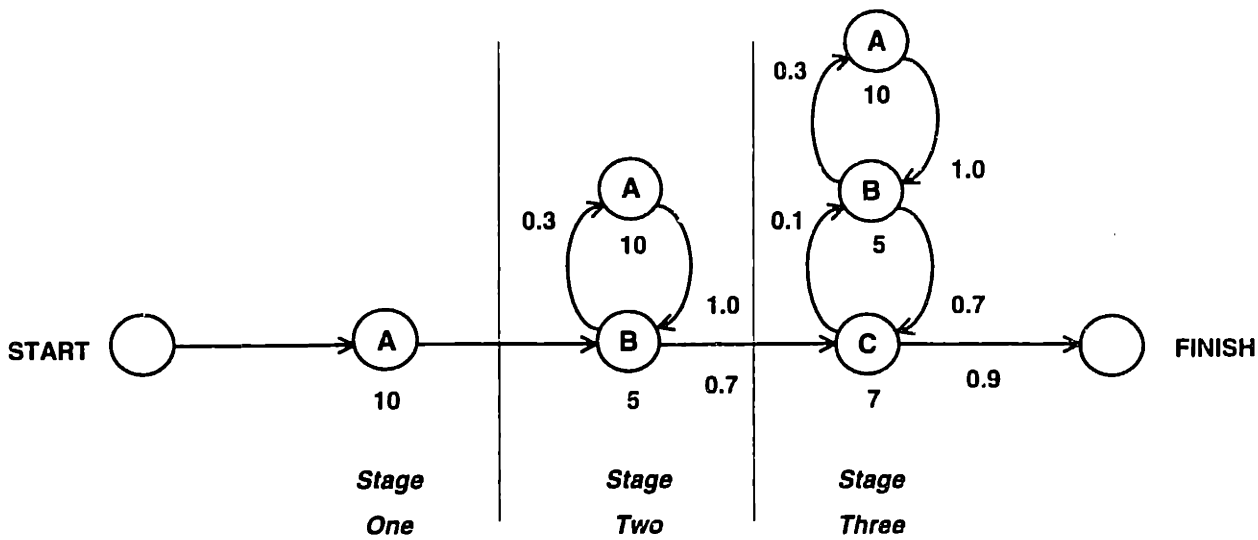
Equation 5.12

A variance can be calculated for a reward Markov chain using any number of methods such as monte carlo simulation [23]. This is not performed here.

As an example, we assume that the hardware development process shown in Figure 5.2 and Figure 5.6 is modeled with task times $t_a = 10$, $t_b = 5$, and $t_c = 7$, and transition probabilities $p_{ab} = 1.0$, $p_{ba} = 0.3$, $p_{bc} = 0.7$, and $p_{cb} = 0.1$. All of the remaining transition probabilities are zero. These values are interpreted as follows: circuit design requires 10 days, placing and routing the components requires 5 days, manufacturing the PCB requires 7 days, there is a probability of 0.3 that we will have to repeat circuit design after placing and routing the components, and there is a probability of 0.1 that we will have to repeat placing and routing the components after manufacturing the PCB. Continuing our example, the matrix M is as follows:

$$M = \begin{bmatrix} 10 & 0.3 & 0 \\ 1.0 & 5 & 0.1 \\ 0 & 0.7 & 7 \end{bmatrix}$$

The reward Markov chain representation is as follows:



In order to preserve the strict precedence relationship between circuit design (Task A) and placing and routing the components (Task B), the transition probability p_{ab} is set equal to 1.0 (such that the sum of the off-diagonal elements in the first column of M is equal to 1.0, given the null probability in row 3/column 1). Any value less than 1.0 would allow the possibility that manufacturing the PCB could occur immediately after circuit design, during the transition from stage 2 to stage 3, or that the process could be completed after circuit design, during the transition from stage 3 to the finish end of the chain. Similarly, to preserve the strict precedence relationship between placing and routing the components (Task B) and manufacturing the PCB (Task C), the transition probability p_{bc} is set equal to 1.0 - 0.3 = 0.7 (such that the sum of the off-diagonal elements in the second column of M is equal to 1.0, given the null probability in row 1/column 3). Any value less than 1.0 would allow the possibility that the process could be completed after placing and routing the components, during the transition from stage 3 to the finish end of the chain.

To preserve all of the strict precedence relationships between tasks in an $n \times n$ matrix M representing a reward Markov chain, the sum of the off-diagonal elements in the first n-1 columns must be equal to 1.0. This is done by setting the transition probability below the diagonal in each column to *1-(sum of the transition probabilities above the diagonal)*. All of the remaining transition probabilities below the diagonal in the first n-2 columns must be set equal to 0. Figure 5.7 shows the matrix M for a reward Markov chain that maintains strict precedence relationships.

$$M = \begin{bmatrix} t_a & p_{ba} & p_{ca} & p_{da} & \cdots & p_{za} \\ 1 & t_b & p_{cb} & p_{db} & \cdots & p_{zb} \\ 0 & 1-p_{ba} & t_c & p_{dc} & \cdots & p_{zc} \\ 0 & 0 & 1-p_{ca}-p_{cb} & t_d & \cdots & p_{zd} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & 0 & \cdots & t_z \end{bmatrix}$$

Figure 5.7: Strict Precedence Relationships in a Reward Markov Chain

Using Equation 5.8 and Equation 5.9, the matrix P and the column vector B are easily determined as follows:

$$P = \begin{bmatrix} 1.0 & -1.0 & 0 \\ -0.3 & 1.0 & -0.7 \\ 0 & -0.1 & 1.0 \end{bmatrix} \qquad B = \begin{bmatrix} 10 \\ 5 \\ 7 \end{bmatrix}$$

Equation 5.10, the lower triangular matrix L, diagonal matrix D, and upper triangular matrix U, are as follows:

$$L = \begin{bmatrix} 1.0 & 0 & 0 \\ -0.3 & 1.0 & 0 \\ 0 & -0.14 & 1.0 \end{bmatrix} \qquad D = \begin{bmatrix} 1.0 & 0 & 0 \\ 0 & 0.7 & 0 \\ 0 & 0 & 0.9 \end{bmatrix} \qquad U = \begin{bmatrix} 1.0 & -1.0 & 0 \\ 0 & 0.7 & -0.7 \\ 0 & 0 & 0.9 \end{bmatrix}$$

Using Equation 5.11 and Equation 5.12 the total expected time for each stage X' and the completion time T are as follows:

$$X' = \begin{bmatrix} 10.0 \\ 11.4 \\ 9.1 \end{bmatrix} \qquad T = 30.5 \text{ days}$$

We conclude that the expected completion time for the hardware development process is 30.5 days.

Appendix E shows MATLAB code that implements the reward Markov chain calculations performed in this example (expected value of the completion time using the efficient length computation algorithm).

5.3.2 Sensitivity Analysis

The sensitivity of the completion time T with respect to each parameter l (task times t_i and transition probabilities p_{jk}), is defined as shown in Equation 5.13.

$$S_l^T = \frac{\Delta E[T] / E[T]}{\Delta l / l}$$

Equation 5.13

We can use this sensitivity analysis to evaluate the effect of changes to the task times and transition probabilities on the completion time. This can be useful in identifying tasks that may require special attention, assessing risk, or determining which tasks if reduced (or increased) have the greatest potential to reduce (or increase) the completion time. Continuing our previous example, using Equation 5.13, the sensitivities of the completion time with respect to the task times and transition probabilities are ranked as follows:

Time	Sensitivity
t_a	0.4844
t_b	0.2604
t_c	0.2552

Probability	Sensitivity
p_{ba}	0.3363
p_{cb}	0.0747
p_{eb}	N/A
p_{bc}	-0.7735

A positive sensitivity denotes a parameter that if increased, will increase the completion time A negative sensitivity denotes a parameter that if increased, will reduce the completion time. Two conclusions can be drawn from this sensitivity analysis. First, a reduction in the task time for circuit design (t_a = 10) will yield the largest reduction in the completion time, relative to the other task times (sensitivity = 0.3363). Second, an increase in the probability of manufacturng the PCB (p_{bc} = 0.7), without having to repeat circuit design, will yield the largest reduction in the completion time, relative to the other transition probabilities (sensitivity = -0.7735). The sensitivity with respect to the transition probability p_{ab} = 1.0 is not applicable because a reduction in this value would violate the strict dependence between circuit design and placing and routing the components, while an increase in this value would violate the validity of the reward Markov chain (the sum of the off-diagonal elements of M in the first column would be greater than 1.0).

Appendix E shows MATLAB code that performs the reward Markov chain sensitivity analysis performed in this example.

5.4 Comparison of Both Models

Based on examples in the previous sections, we present a brief comparison of the signal flow graph model and the reward Markov chain model.

The completion time that was calculated using the signal flow graph was exactly the same as the completion time that was calculated using the reward Markov Chain (30.5 days). This was explained by the fact that the signal flow graph and the reward Markov chain both allowed for the possibility of "returning" to all of the earlier tasks as a result of performing any of the later tasks. For example, in the signal flow graph model and the reward Markov chain model, if placing and routing the components was repeated as a result of manufacturing the PCB, it was also possible that circuit design was repeated as a result of placing and routing the components. In other words, both models captured the same phenomenon. The signal flow graph calculated the expected time *to traverse the process* and the reward Markov chain calculated the expected time *spent within the process*.

Also, the sensitivity analysis values of the reward Markov chain model were almost identical to the sensitivity of the signal flow graph model. They only differed in one subtle, yet important way. Using the signal flow graph we were able to calculate a sensitivity for the transition probability from manufacturing

the PCB to the finish state. Using the reward Markov chain we were unable to calculate a sensitivity for the transition probability from manufacturing the PCB to the finish state. This was explained by the fact that this transition probability was derived from the reward Markov chain structure (the probability of completing the process was implicitly $1\text{-}p_{cb}$, whereas with the signal flow graph structure this probability was explicitly p_{45}). Otherwise, the sensitivity values were exactly the same.

5.5 Research Methodology

A digital wireless telephone consists of two kinds of hardware: an analog PCB and a digital PCB. Two functional groups were responsible for designing and testing these PCB's, the Analog Design group and the Digital Design group, respectively. To assess the signal flow graph and reward Markov chain models, both were applied to the PCB in the Analog Design group only.

Figure 5.9 shows the hardware process flow diagram for the analog PCB (incidentally, it was the same as the process flow diagram for the digital PCB). This process flow diagram was constructed from the tasks that were identified during earlier efforts to collect data (see Section 2.8 for a description of the data collection process). A more detailed description of these tasks can be found in Section 2.7, with the exception of "Manufacturing: Ready-to-Build". This task was not identified initially, but was later added to the process flow diagram to capture the time associated with pre-manufacturing final reviews. The tasks are numbered one through sixteen, with sixteen associated task times t_1 through t_{16}, and eight associated transition probabilities p_1 through p_8. The tasks in the light-colored boxes (t_1, t_3, t_5, t_5, t_6, t_7, t_9, and t_{16}) denote design activities that were performed internal to the Analog Design group. The tasks in dark-colored boxes (t_2, t_4, t_8, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}, and t_{15}) denote design activities that were performed external to the Analog Design group (the functional group that actually performed the task is identified by capital letters). The transition probabilities represented either potential delays (p_1 and p_2), potential iterations (p_3, p_4, p_5, p_6, and p_7), or a choice among parallel activities (p_8, p_9, and p_{10}). Consistent with a rapid prototyping approach, once a PCB was designed, manufactured, and tested, the next prototype would begin development (denoted by the dash line). This global iterative loop was not taken into account during modeling.

After completing the process flow diagram, data was collected on task times and transition probabilities for three past prototypes of the analog PCB (an early prototype, an intermediate prototype, and a later prototype, during the handset's development). In other words, three separate sets of sixteen task times and eight transition probabilities were collected. The three prototypes that were selected for modeling represented a good cross-section of the prototypes that had been performed. Each had its own unique characteristics (e.g. functions, features, completion time, etc.).

Table 5.1 shows the estimated task time, total completion time, and estimated transition probability data that were collected for the hardware prototypes. A member of the Project Management group estimated the task times and provided the total completion time for each prototype. The task times were *approximations* of the original estimates of the task times, prior to each prototype (e.g. without knowledge of the outcome). The total completion time was an *exact measurement* of the actual completion time for each prototype (critical path). The manager of the Analog Design group estimated the transition probabilities for each prototype. The transition probabilities were *approximations* based on what would have been predicted, had they been asked to provide such information prior to each prototype (e.g. without knowledge of the outcome). Consistent with a rapid prototyping approach, the transition probabilities were also based on the nature of the functionality being introduced for each prototype. A less challenging set of features was subject to smaller values, while a more challenging set of features was subject to larger values. In other words, difficult functions and features had a high likelihood of causing iteration (note that the prototypes in Table 5.1 are ordered by decreasing levels of difficulty).

Time	Prototype #1	#2	#3
t_1	30	22	4
t_2	40	0	0
t_3	4	0	0
t_4	26	0	0
t_5	34	32	44
t_6	8	2	2
t_7	0	0	0
t_8	4	2	2
t_9	0	0	0
t_{10}	6	80	8
t_{11}	60	0	154
t_{12}	6	6	8
t_{13}	2	16	4
t_{14}	4	12	4
t_{15}	14	16	38
t_{16}	46	22	24
TOTAL[†]	**272**	**204**	**276**

[†] *total completion time (critical path).*

Probability	Prototype #1	#2	#3
p_1	0.50	0.20	0.05
p_2	0.80	0.40	0.05
p_3	0.30	0.20	0.05
p_4	0.40	0.10	0.10
p_5	0.20	0.20	0.10
p_6	0.20	0.20	0.10
p_7	0.30	0.10	0.10
p_8	0.05	0.05	0.08
p_9	0.90	0.90	0.84
p_{10}	0.05	0.05	0.08

Table 5.1: Task Times and Transition Probabilities for the Hardware Prototypes

Once the data were collected, three models were constructed: a static signal flow graph model, where the task times remained fixed, a dynamic signal flow graph model, where the task times were subject to change, and an adjusted reward Markov chain model, where the task times and transition probabilities were slightly modified to maintain the reward Markov chain structure. In the following sections, we will describe these models in greater detail, and apply each model to the three past prototypes of the analog PCB at the sponsor company.

5.5.1 Static Signal Flow Graph Model

The static signal flow graph model was based on the assumption that the task times remained fixed. For example, a task that required 10 days during its first-pass, would also require 10 days if it were repeated. Figure 5.8 shows the archetype for the static signal flow graph model.

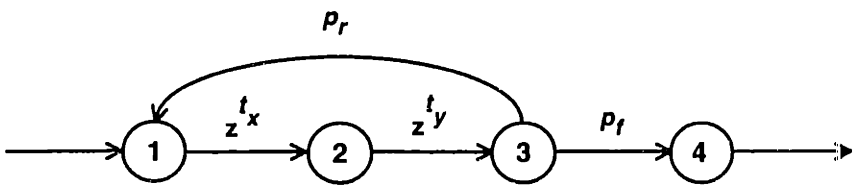


Figure 5.8: Archetype for the Static Signal Flow Graph Model

There is a probability p_r (repeat probability) that Task X will have to be repeated after Task Y is completed, and a probability p_f (forward probability) that the process will move forward (note that $p_r + p_f = 1.0$). The durations for Tasks X and Y are static. For subsequent iterations, Tasks X and Y will require the same amount of time that they required the first time they were performed. Using this archetype, we modeled the task times in hardware development process shown in Figure 5.9, that could potentially be repeated (t_4, t_5, t_6, t_7, and t_8).

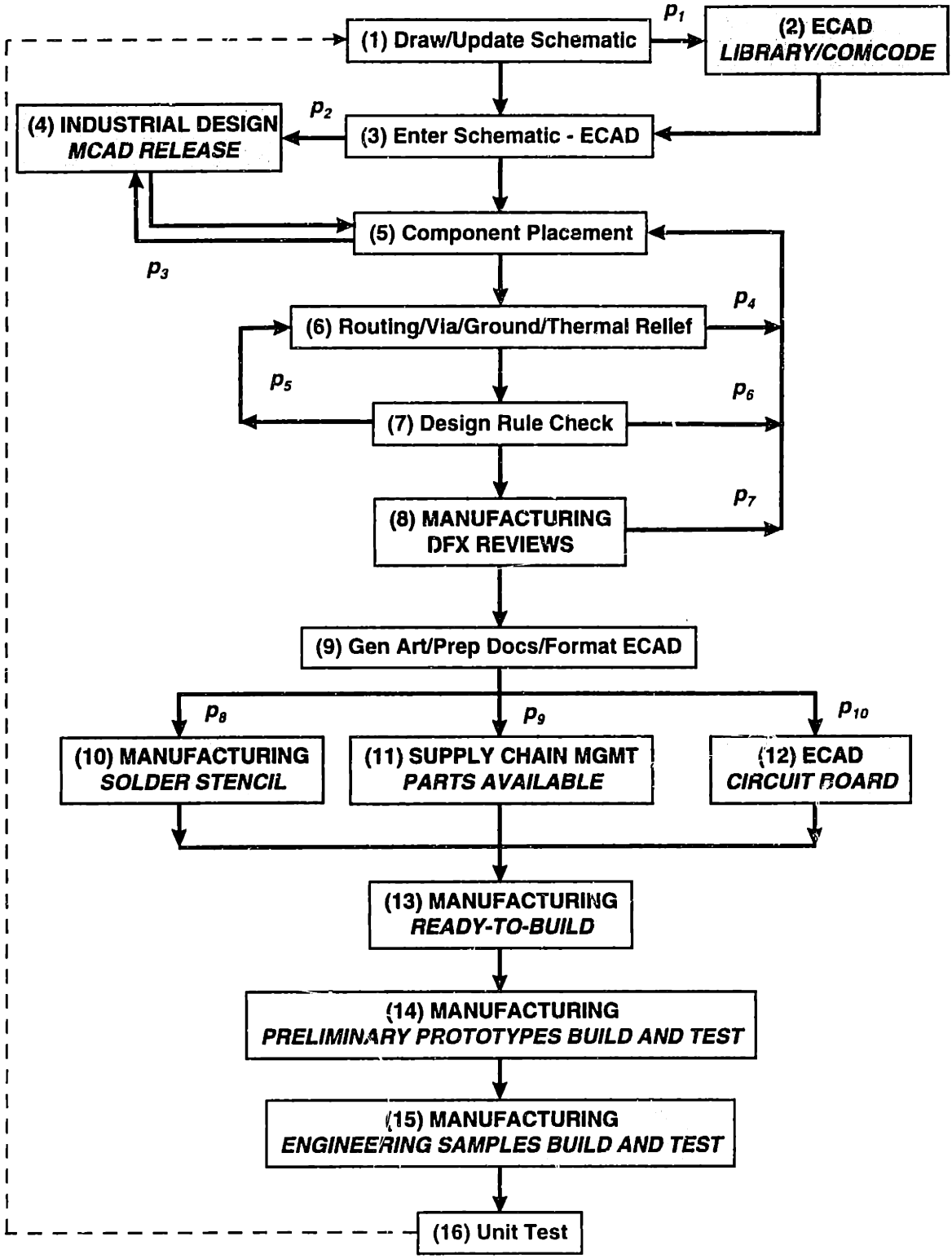


Figure 5.9: Hardware Process Flow Diagram

5.5.2 Dynamic Signal Flow Graph Model

The dynamic signal flow graph model was based on the assumption that the task times were subject to change. For example, a task that required 10 days during its first-pass, could require 5 days if it were repeated. Figure 5.10 shows the archetype for the dynamic signal flow graph model.

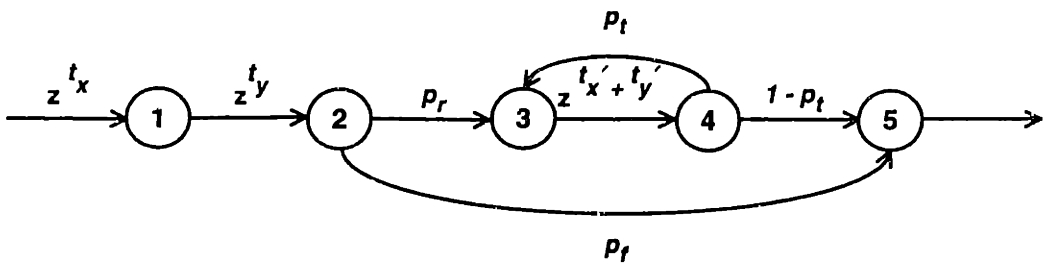


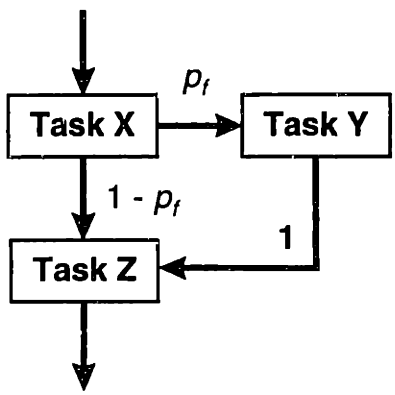
Figure 5.10: Archetype for the Dynamic Signal Flow Graph Model

There is a probability p_r (repeat probability) that Task X will have to be repeated after Task Y is completed the first time, a probability p_f (forward probability) that the process will move forward (note that $p_r + p_f = 1.0$), and a probability p_t (terminal probability) that Task X will have to be repeated after Task Y is completed a second time, and all subsequent iterations. The durations for Tasks X and Y are dynamic. For their first iterations, Tasks X and Y require times t_x and t_y, respectively. For their second and subsequent iterations, Tasks X and Y require times t_x' and t_y', respectively. We call the second iteration task times, terminal task times, because they denote the time required for the second iteration and all subsequent iterations. Using this archetype, we modeled the tasks times in the hardware development process shown in Figure 5.9, that could potentially be repeated (t_4, t_5, t_6, t_7, and t_8).

For the dynamic signal flow graph model, we set $p_t = 0.01$, such that the terminal probability for all of the tasks was equal to 0.01, and $t_i = 0.5t_i'$, for all i, such that all of the terminal task times were 50% of their corresponding first-pass task times. This was done for all three prototypes.

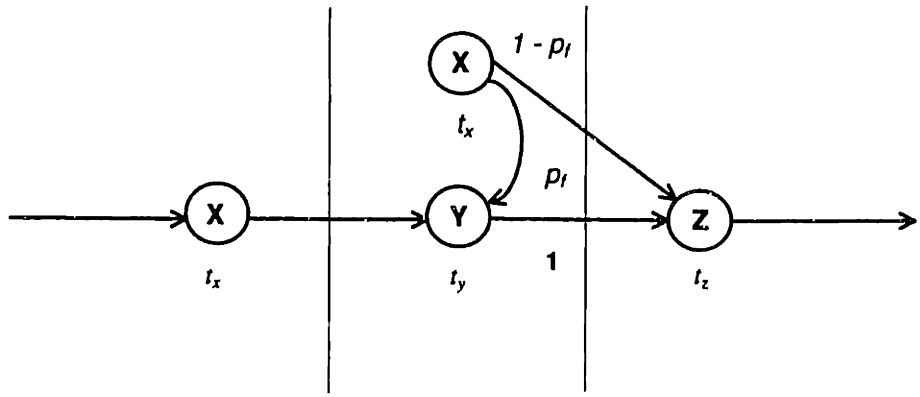
5.5.3 Adjusted Reward Markov Chain Model

The adjusted reward Markov chain modified the task times and transition probabilities to maintain the reward Markov chain structure. The reason for these adjustments was that the reward Markov chain could not capture forward transition probabilities and associated task times. A forward transition probability is essentially a branch. It is a probabalistic choice that captures the possibility of one task being followed in sequence by another task, versus being followed in sequence by a different task. This scenario is depicted as follows:



We can interpret the forward transition probability p_f as the probability that if Task X is performed, that Task Y will have to be performed. Maintaining the strict precedence relationship between Task Y and Task Z (ensuring that the sum of the off-diagonal elements in the second column of M are equal to 1), the reward Markov chain would look as follows:

$$M = \begin{bmatrix} t_x & & \\ p_f & t_y & \\ & 1 & t_z \end{bmatrix}$$



Clearly, such a representation does not accurately depict the reward associated with Task Y. This is because the off-diagonal values in the reward Markov chain structure denote probabalistic selections on *previous* tasks, as opposed to *subsequent* tasks. In our example, since Task X is followed by Task Y, the transition probability $p_f = p_{xy}$ is the probability that if Task X if performed that Task Y will have to be repeated, as opposed to performed for the first time. To overcome this challenge, we adjust the reward Markov chain by modeling forward transition probabilities and associated task times as expected values. Figure 5.11 shows the archetype for the adjusted reward Markov chain model (using the same convention as our example).

$$M = \begin{bmatrix} t_x & & \\ 1 & p_f t_y & \\ & 1 & t_z \end{bmatrix}$$

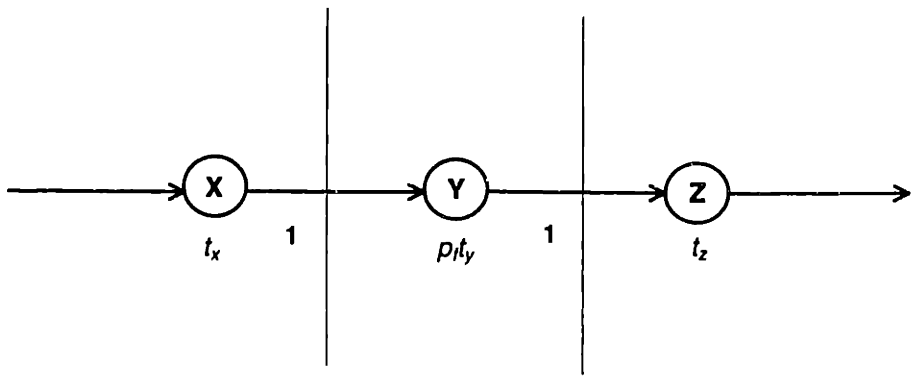


Figure 5.11: Archetype for the Adjusted Reward Markov Chain

The forward transition probability p_f and the associated task time t_y are modeled whereby the state represented by Task Y has a reward that is equal to its expected value ($p_f t_y$). In making this adjustment, we continue to maintain the strict precedence relationship between Task Y and Task Z (the sum of the off-diagonal elements in the second column of M are still equal to 1). Using this archetype, we modeled the forward transition probabilities and associated task times of the hardware development process shown in Figure 5.9 (p_1, p_2, p_8, p_9, p_{10}, and t_2, t_4, t_{10}, t_{11}, t_{12}, respectively).

5.6 Results

Table 5.2 and Table 5.3 show the completion times and variances that were predicted by applying the static signal flow graph model, the dynamic signal flow graph model, and the adjusted reward Markov chain model, to each of the hardware prototypes.

	COMPLETION TIME (Days)					
	Prototype #1		Prototype #2		Prototype #3	
MODEL	Value	% Error	Value	% Error	Value	% Error
Static	421.4	+54.9%	167.4	-17.9%	216.0	-27.9%
Dynamic	261.0	-4.0%	146.2	-28.3%	224.2	-18.8%
Adjusted	413.6	+52.1%	166.2	-18.5%	215.8	-21.8%
ACTUAL	**272.0**	-	**204.0**	-	**276.0**	-

Table 5.2: Completion Time Results for the Hardware Prototypes

	VARIANCE (Days)					
	Prototype #1		Prototype #2		Prototype #3	
MODEL	Value	% Mean	Value	% Mean	Value	% Mean
Static	224.8	53.3%	54.4	32.5%	53.8	24.9%
Dynamic	33.0	12.6%	53.8	36.8%	27.6	12.3%
Adjusted	-	-	-	-	-	-

Table 5.3: Variance Results for Hardware Development Process

From the completion time results it was clear that the dynamic signal flow graph model was the most accurate model. It exhibited errors of -4.0%, -28.3%, and -18.8%, for each prototype respectively. The model outperformed the static signal flow graph model and the adjusted reward Markov chain model on each prototype, except Prototype #2. Both the static signal flow graph model and the reward Markov chain model tended to overestimate the actual completion time (by as much as 54.9% and 52.1% respectively). This was explained by the fact that both of these models assumed fixed task times. Therefore, if a task was repeated, it required the same amount of time for each subsequent iteration. In reality, second-pass task times were typically less than their first-pass counterparts. Interestingly, and in support of this argument, both of these models performed almost identically with respect to their predictions for completion time, with the largest difference being only 7.8 days.

From the variance results it was concluded that the dynamic signal flow graph model once again provided the most useful information. Variances were not calculated for the reward Markov chain, however, the static signal flow graph model returned variances as high as 53% of the mean. This was not considered reflective of the environment's variability. On the other hand, the dynamic signal flow graph returned variances as low as 12.3% and as high as 36.8% of the mean, which were considered much more consistent with preliminary expectations.

As mentioned earlier, the dynamic signal flow graph model predicted the completion time for Prototype #1 with particularly impressive accuracy (-4.0% error). Figure 5.12 shows the PDF and CDF generated by the dynamic signal flow graph model for Prototype #1.

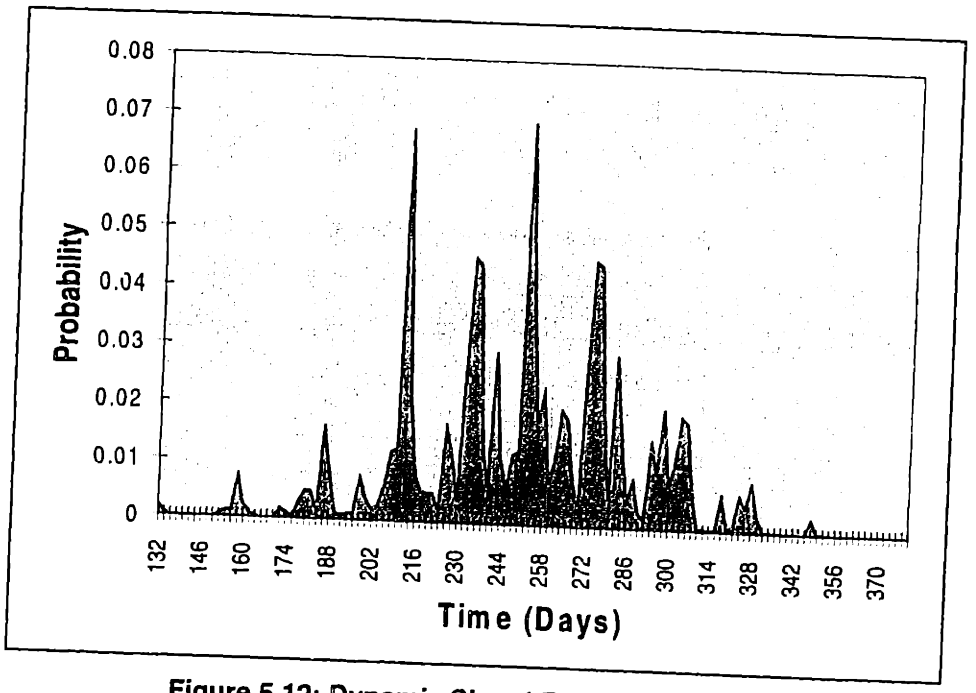


Figure 5.12: Dynamic Signal Flow Graph Model
Probability Distribution Function (PDF) of Prototype #1

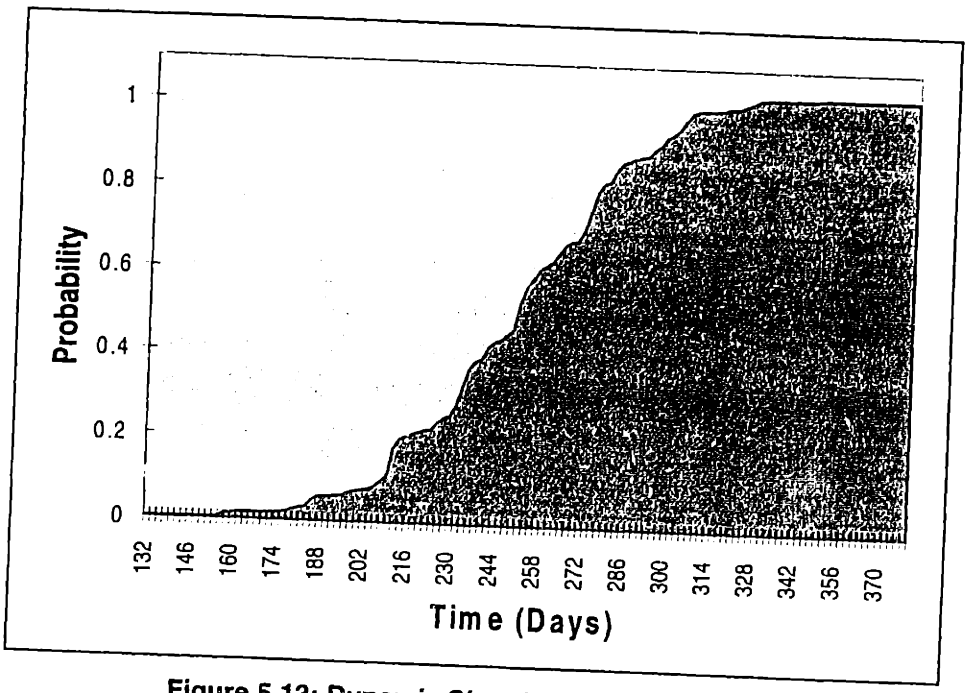


Figure 5.13: Dynamic Signal Flow Graph Model
Cumulative Distribution Function (CDF) of Prototype #1

From this PDF and the CDF we concluded that the probability density was fairly evenly distributed. Similar observations were made regarding the PDF's and CDF's of the Prototypes #2 and #3. Table 5.5 and Table 5.4 show the sensitivity analysis results with respect to the task times and transition probabilities for Prototype #3 (task time rankings 1-3 and 9-11 only (rankings 12-16 were zero) and transition probability rankings 1-3 and 14-16 only).

	Static Signal Flow Graph Model		Dynamic Signal Flow Graph Model		Adjusted Reward Markov Chain Model	
RANK	Time	Value	Time	Value	Time	Value
1	t_{11}	0.5987	t_{11}	0.6042	t_{11}	0.5993
2	t_{15}	0.1759	t_{15}	0.1775	t_{15}	0.1760
3	t_{16}	0.1111	t_{16}	0.1121	t_{16}	0.1112
9	t_8	0.0103	t_8	0.0093	t_8	0.0103
10	t_{10}	0.0030	t_{10}	0.0030	t_{10}	0.0030
11	t_{12}	0.0030	t_{12}	0.0030	t_{12}	0.0030

Table 5.4: Sensitivity Analysis Results of the Task Times for Prototype #3

	Static Signal Flow Graph Model		Dynamic Signal Flow Graph Model		Adjusted Reward Markov Chain Model	
RANK	Probability	Value	Probability	Value	Probability	Value
[1]					p_{10}	33.7017
[2]					p_8	11.1500
1	p_9	0.6560	p_9	0.6601	p_9	0.9326
2	p_7	0.0058	p_t	0.0467	p_7	0.0058
3	p_6	0.0052	p_7	0.0009	p_6	0.0052
14	p_2	0.0005	p_5	-0.0010	p_4	0.0049
15	p_{10}	-0.0423	p_{10}	-0.0428	p_5	0.0025
16	p_8	-0.0466	p_8	-0.0471	p_3	0.0014

†*These rankings are actually 1-5 and 11-13.*

Table 5.5: Sensitivity Analysis Results of the Transition Probabilities for Prototype #3

From the sensitivity analysis with respect to the task times, it was clear that all three models performed identically by ranking order, and were extremely close to one another by value (note that once again, the static signal flow graph model and the adjusted reward Markov chain model performed almost identically). All of the models returned positive sensitivities, which was interpreted to mean that an increase in any of these task times would increase the completion time. Based on past experience, it was known that Prototype #1 was particularly sensitive to the Supply Chain Management group's ability to procure parts, and the Analog Design group's ability to complete the unit test. Both of the task times associated with these activities, t_{11} and t_{16}, were ranked first and third by all three models. Similar observations were made regarding the sensitivities of Prototypes #1 and #2 with respect to the task times.

From the sensitivity analysis with respect to the transition probabilities, it was clear that all three models performed similarly, with some slight differences in their results. The adjusted reward Markov chain model did not produce any negative sensitivities (this was true for all of the prototypes). This was explained by the fact that forward transition probabilities were reinterpreted as expected values (in other words, an increase in the probability would automatically increase the completion time) and repeat probabilities already served to increase the completion time. Consequently, the sensitivity analysis of the adjusted reward Markov chain tended to skew the sensitivities of forward transition probabilities. For example, the first and second ranked sensitivities of the adjusted reward Markov chain model were excessively large (transition probability p_{10} at 33.7017 and transition probability p_8 at 11.1500). Both of these probabilities were forward transition probabilities. Therefore, one of the drawbacks of the adjusted reward Markov chain model was the loss of the ability to accurately assess the sensitivity of the

completion time with respect to forward transition probabilities. We also observed that the static signal flow graph model and the dynamic signal flow graph model produced identical results by ranking order and extremely close results by value. This was explained by the fact that while both of these models captured task times in different ways, they were very similar in the way they captured transition probabilities. In particular, both models captured forward transition probabilities identically. Both models also returned negative sensitivities (again, the reward Markov chain model did not), which was interpreted to mean that an increase in any of these transition probabilities would decrease the completion time. As expected, the sensitivity analysis of the dynamic signal flow graph model ranked the terminal probability p_t fairly high (second). Finally, as mentioned earlier, it was known that Prototype #1 was particularly sensitive to the Supply Chain Management group's ability to procure parts. The transition probability associated with this task (p_9) was ranked first by all three models (ignoring the skewed sensitivities from the adjusted reward Markov chain model). Similar observations were made regarding the sensitivities of the other prototypes with respect to the transition probabilities.

In summary, the dynamic signal flow graph model demonstrated good performance in predicting the completion time, predicting the variance, and producing a sensitivity analysis of the hardware development process. One of the recommendations to the sponsor company was to implement the dynamic signal flow graph model as a tool for modeling hardware completion time in the future. Recommendations are summarized in Chapter 7.

5.7 Discussion

In this chapter, we introduced the Signal Flow Graph and the Reward Markov Chain, analytical tools that were used to predict the completion time of analog printed circuit boards (PCBs) at the sponsor company. Using analytical techniques such as Mason's Gain Formula for signal flow graphs, and Smith and Eppinger's Efficient Length Computation Algorithm for reward Markov chains, we constructed three separate models. The static signal flow graph model assumed fixed task times. The dynamic signal flow graph model assumed that task times were subject to change. The adjusted reward Markov chain model modified the task times and transition probabilities to maintain the reward Markov chain structure. All three models also yielded a sensitivity analysis with respect to task times and transition probabilities. We also calculated variances for the static and dynamic signal flow graph models. We did not calculate variances for the adjusted reward Markov chain model. The main advantage of these tools, when compared to more traditional project management tools such as PERT/CPM and GANTT charts, was the ability to model iteration, and quantify its effect on completion time.

The static signal flow graph model and the adjusted reward Markov chain model both produced noticeably similar results for completion times and sensitivity analyses with respect to task times. This was explained by similarities in the way these models captured task times. All three models performed almost identically for sensitivity analyses with respect to task times, while the static signal flow graph model and the dynamic signal flow graph model performed similarly for sensitivity analyses with respect to transition probabilities. This was explained by similarities in the way these models captured transition probabilities. Both the static signal flow graph model and the dynamic signal flow graph model produced positive and negative sensitivities with respect to transition probabilities. The adjusted reward Markov chain was unable to do so, as a result of modifications that were made to the model in order to maintain its structure.

Overall, the dynamic signal flow graph demonstrated the best performance. It (as well as the static signal flow graph model) also offered the additional benefits of a variance, a Probability Distribution Function (PDF), and a Cumulative Distribution Function (CDF). Using data that were collected from managers at the sponsor company, the dynamic signal flow graph model was able to predict the completion time of past prototype within 4% of the actual completion time. It was also able to identify some of the critical sensitivities for this prototype with respect to task times and transition probabilities, based on past experience.

It was believed that the success of the dynamic signal flow graph model was a direct result of its ability to capture a very real phenomenon - task times that change with subsequent iterations. The static signal flow graph model and the adjusted reward Markov chain model used completely different mathematical constructs, yet were based on the assumption that task times did not change. The fact that both of these

models consistently returned results that were similar to one another (completion time predictions as well as sensitivity analyses), was a testament to the fact that such an assumption can have a tremendous impact on the results that are generated by a model (these models overestimated the completion time by as much as 55%).

Despite its impressive performance, the dynamic signal flow graph model presented here only allowed a single dynamic change for each task. The second-pass task time, which differed from the first-pass task time, was the terminal task time. No further changes could be captured. Looking forward, a more elaborate dynamic signal flow graph model could easily be designed to allow for the third-pass, or even fourth-pass task times to differ from their predecessors as well. We should mention that the reward Markov chain is capable of modeling dynamic task times [36], however, such an approach requires simulation to be performed. The methods presented here were strictly computational in nature.

Finally, our recommendation to the sponsor was to implement the dynamic signal flow graph model as a tool for modeling hardware completion time in the future. This represented an opportunity for the model to be further refined and benchmarked in a number of different scenarios, which would hopefully provide even greater insight on how such a framework could be leveraged best.

CHAPTER 6: MODELING FIRMWARE COMPLETION TIME

6.1 Modeling Firmware Development

Software denotes a set of instructions written in a programming language (e.g. C++) that are executed by a hardware device (e.g. microprocessor). Firmware denotes the binary/hexadecimal versions of the software as read-only memory (ROM) files for use in manufacturing. For a digital wireless telephone, firmware specifically refers to the software resident in the handset that programs the microcontroller and the digital signal processor (DSP).

Methods for modeling firmware development have received a great deal of attention within the past decade. This has been in response to the increasingly critical role that software has played in the development of a vast array of products. In a number of industries, particularly the communications industry, electronic components have become central to a product's architecture. This has caused software to be one of the primary agents in realizing value-added functions and features. As a result of this trend, a number of efforts have been made to codify methodologies for estimating the cost, determining the number of errors (or faults), assessing the risk, and predicting the completion time of firmware projects.

Historically, attempts to measure these phenomena have been somewhat ad-hoc, including those employed by the sponsor company. Putnam and Myers describe such approaches as "...an intuitive process, subject to great variation, when done by different people [27]." Consequently, we developed the Aggregate-Code Model, a tool that was designed specifically for the sponsor company to remove some of the "guessing" associated with predicting completion time and predicting the amount of time spent repairing faults on large-scale firmware projects. The model served two purposes. First, to provide a methodology for extracting key performance metrics from the firmware product administration database. Second, to provide a starting point for the development of more advanced models that characterize firmware behavior. In other words, the model was not intended to supplant their existing techniques, but rather supplement them, while at the same time establish a framework to implement better models.

In this chapter, we present the theory underlying the aggregate-code model, the methodology for extracting the necessary inputs to the model from the sponsor company's firmware product administration database, and the results from applying the model to a subset of the firmware on the digital wireless telephone at the sponsor company. We conclude with a discussion of the model's benefits and limitations.

6.2 Aggregate-Code Model

The aggregate-code model is a simple model that can be used to predict firmware completion time and the amount of time spent repairing faults (repair time). The model can also be used to analyze how changes to variables such as the number of lines of code or staffing levels can impact completion time. We first describe the relationships upon which the model is based, and then we present the theory underlying the model itself.

6.2.1 Model Relationships

The aggregate-code model is based on two simple relationships:

$$\text{(1)} \qquad \textit{Completion Time} = \frac{\textit{Code}}{\textit{Productivity Rate}}$$

$$\text{(2)} \qquad \textit{Repair Time} = \textit{Code} \cdot \textit{Fault Density} \cdot \textit{Time-To-Repair Faults}$$

The model variables are defined as follows:

- ***Completion Time*** represents the duration to complete the project and is typically measured in months.
- ***Code*** represents a measure of project completeness and is typically measured in KNCSL (thousands of non-commentary source lines) or SLOC (source lines of code).
- ***Productivity Rate*** represents a myriad of factors such as the human capital required for the project, the level of quality of requirements, the programming language in use, the state of technology in the software environment (e.g. software tools, development equipment, and machine capabilities), the skills and experience of the programmers, and the complexity of the project, and is typically measured in KNCSL/month [27].
- ***Repair Time*** represents the time spent repairing faults and is typically measured in months.
- ***Fault Density*** represents a measure of errors or faults per source line of code and is typically measured in faults/KNCSL.
- ***Time-to-Repair Faults*** represents the mean time-to-repair faults and is typically measured in months/fault.

6.2.2 Model Theory

Figure 6.1 shows the basic framework that governs the theory underlying the aggregate-code model.

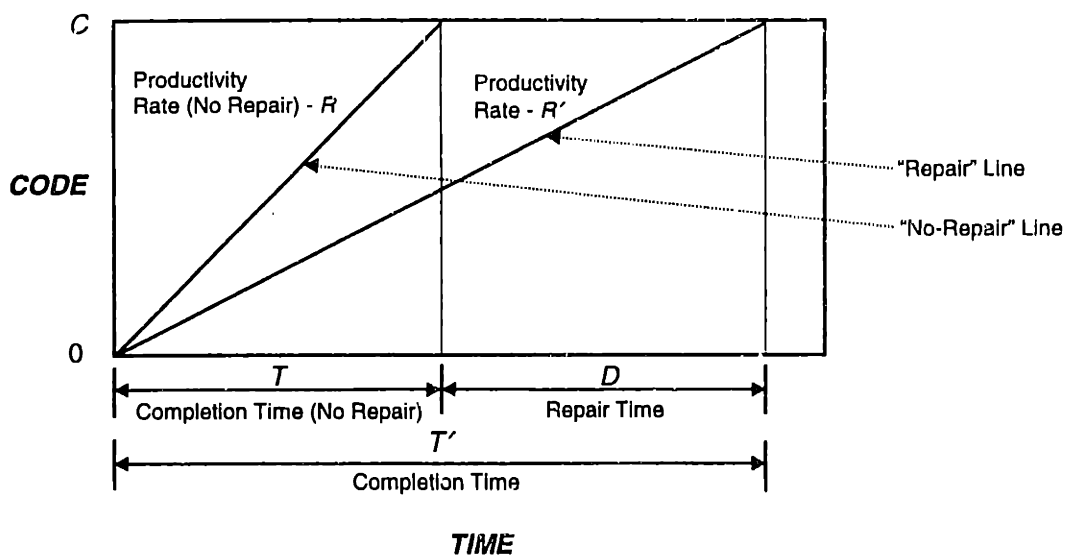


Figure 6.1: Aggregate-Code Model

The above diagram captures two scenarios for a firmware project - one without errors, where no time is spent repairing faults (an unrealistic scenario, that we will refer to as the "no repair" scenario) and one with errors, where time is spent repairing faults (a realistic scenario, that we will refer to as the "repair" scenario). The elements of the "no repair" scenario are the following: the number of lines of code C, the productivity rate (no repair) R, and the completion time (no repair) T. The elements of the "repair" scenario are: the number of lines of code C, the productivity rate R', the completion time T', and the repair time D.

With the absence of faults in the "no repair" scenario, given the number of lines of code C, it would be significantly easier to predict the completion of a firmware project, relatively speaking. The productivity rate (no repair) R would be estimated based on past experience or historical performance, and the completion time (no repair) T could then be predicted. This is represented by the "no-repair" line to the left of the above diagram. However, the existence of faults in the "repair" scenario (and therefore time spent repairing faults D) reduces the productivity rate to R' and extends the completion time to T'. This is

represented below by the "repair" line to the right of the above diagram. The completion time (no repair) T, completion time T', and repair time D, are governed by the relation shown in Equation 6.1

$$T' = T + D$$

Equation 6.1

The repair time D is calculated as the product of the number of lines of code C, the fault density F, and the mean time-to-repair faults μ_D, as shown in Equation 6.2 (the number of faults M is equal to the product of the number of lines of code C and the fault density F).

$$D = C \cdot F \cdot \mu_D$$

Equation 6.2

In the "no repair" scenario, the completion time (no repair) T, the number of lines of code C, and the productivity rate (no repair) R, are governed by the relation shown in Equation 6.3. In the "repair" scenario, the completion time T', the number of lines of code C, and the productivity rate R', are governed by the relation shown in Equation 6.4.

$$T = \frac{C}{R}$$

Equation 6.3

$$T' = \frac{C}{R'}$$

Equation 6.4

Given two of the variables in either of these equations (either scenario) and the repair time D, we can determine all of the remaining variables. We are primarily concerned with the "repair" scenario, so we will define the model parameters as C, R', D, and T'.

The aggregate-code model seeks to predict the completion time T' and the repair time D. Fortunately, there are a number of existing methods to predict the number of lines of code C including fuzzy logic sizing [43], function point sizing [2, 3], standard component sizing [26], and change sizing [40]. As we will show in the following sections, we can estimate the productivity rate R' from historical data on the productivity of past projects, and use this estimate to predict the completion time T'. We will also show that we can estimate the repair time D from historical data on the fault density F and the mean time-to-repair faults μ_D of past projects. Finally, we will show that if these historical statistics are aggregated (e.g. classified by the nature of the code) at an appropriate level of abstraction, then we can use this data to characterize specific categories of code, as depicted in Figure 6.2.

By aggregating the code into different categories (e.g. Type 1 code, Type 3 code, and User Interface code), we achieve greater flexibility in that we can predict the behavior of specific types of code. For example, the performance of a project that is comprised of a subset of the historical categories could be modeled using only the metrics that characterize those categories.

We now present a generalized version of the aggregate-code model that determines the necessary inputs using data that characterizes the behavior of an entire past project. This is followed by a categorized version of the aggregate-code model that determines the necessary inputs using data that characterizes the behavior of specific categories of code on the same project.

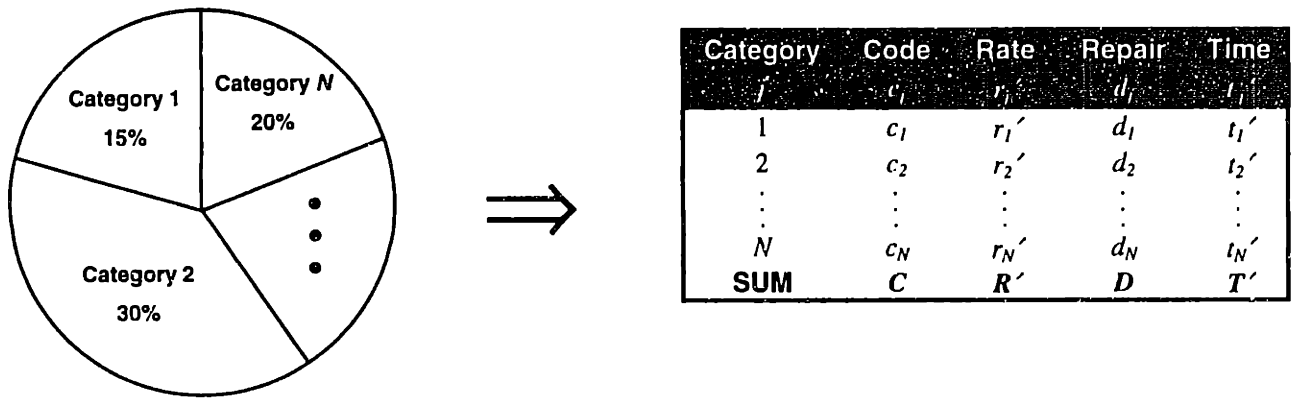


Figure 6.2: Aggregating the Code

In the following sections, we will use the superscript * to distinguish between current or predicted model parameters (C, R', D, T', M) and historical model parameters (C^*, R^*, D^*, T^*, M^*) as well as their lowercase counterparts (e.g. C is a current model parameter that denotes the number of lines of code on the *current* project, while C^* is a historical model parameter that denotes the number of lines of code on a *past* project). All of the remaining variables (F, μ_D, etc., except for sensitivity values) as well as their lowercase counterparts, are based entirely on historical data and will not be superscripted.

6.3 Generalized Version of the Aggregate-Code Model

The generalized version of the aggregate-code model characterizes the behavior of an entire firmware project based on historical data. For example, a past project that consisted of Type 1 code, Type 3 code, and User Interface code, would have metrics associated with the entire project as a means to characterize their collective behavior in the future.

The required historical parameters for this version of the model are the following: the historical number of lines of code C^*, the historical productivity rate R^*, the historical fault density Γ, and the historical mean time-to-repair faults μ_D. The required current parameter for this version of the model is the current number of lines of code C (which can be estimated using any of the aforementioned sizing methods). Given these parameters, the model outputs T' (completion time) and D (repair time) are easily determined, and a sensitivity analysis can performed with respect to the current number of lines of code C.

Table 6.1 lists the variables for the generalized version of the aggregate-code model.

6.3.1 Completion Time

To predict the completion time T' we require the following inputs: the historical productivity rate R^* and the current number of lines of code C. The historical productivity rate R^* is used to predict the current productivity rate R'. It is calculated by dividing the historical number of lines of code C^* by the historical completion time T^*, as shown in Equation 6.5.

$$R' \rightarrow R^* = \frac{C^*}{T^*}$$

Equation 6.5

We also call the generalized version of the aggregate-code model the fixed rate version because the productivity rate does not change over time. The predicted completion time T' is then calculated by dividing the current number of lines of code C by the current productivity rate R', as shown in Equation 6.6.

$$T' = \frac{C}{R'}$$

Equation 6.6

As an example, suppose a past project had a number of lines of code C^* = 52.37 KNCSL and a completion time T^* = 8.94 months. Using Equation 6.5, the historical productivity rate R^* is calculated as follows:

$$R^* = \frac{52.37}{8.94} = 5.86 \text{ KNCSL / month}$$

We then assume that the current productivity rate R' is equal to the historical productivity rate R^*. Continuing our example, suppose the next generation of the same project is estimated to have a number of lines of code C = 63.72 KNCSL. Using Equation 6.6, the predicted completion time T' is calculated as follows:

$$T' = \frac{63.72}{5.86} = 10.87 \text{ months}$$

We conclude that the current project will be completed in 10.87 months. In the next section, we describe in detail our approach to estimating the historical productivity rate R^* using data from a past project at the sponsor company.

Variable	Description	Determination	Units
C	Number of Lines of Code†	Current	*KNCSL*
C^*	Number of Lines of Code†	Historical	*KNCSL*
R'	Productivity Rate (No Repair)	Current	*Code/Month*
R	Productivity Rate (Repair)	Current	*Code/Month*
R^*	Productivity Rate (Repair)†	Historical	*Code/Month*
D	Repair Time	Predicted	*Months*
D^*	Repair Time	Historical	*Months*
T	Completion Time (No Repair)	Predicted	*Months*
T'	Completion Time (Repair)	Predicted	*Months*
T^*	Completion Time (Repair)	Historical	*Months*
M	Number of Faults	Predicted	*Faults*
M^*	Number of Faults	Historical	*Faults*
F	Fault Density†	Historical	*Faults/KNCSL*
μ_D	Mean Time-to-Repair†	Historical	*Months/Fault*
E	Repair Effort	Historical	*Person-Months*
μ_E	Mean Effort-to-Repair	Historical	*Person-Months/Fault*
N_D	Faults Per Repair Day	Historical	*Faults Per Repair Day*

†*denotes a required parameter.*

Table 6.1: Variables for the Generalized Version of the Aggregate-Code Model

6.3.2 Repair Time

To predict the repair time D we require the following inputs: the historical mean time-to-repair faults μ_D, the historical fault density F, and the current number of lines of code C. The predicted repair time D is calculated as the product of the predicted number of faults M and the historical mean time-to-repair faults μ_D, where the predicted number of faults M is the product of the current number of lines of code C and the historical fault density F, as shown in Equation 6.7.

$$D = M \cdot \mu_D \quad \text{(a)} \qquad M = C \cdot F \quad \text{(b)} \qquad D = C \cdot F \cdot \mu_D \quad \text{(c)}$$

Equation 6.7

Continuing our previous example, suppose the past project exhibited a fault density F = 2.03 faults/KNCSL and a mean time-to-repair faults μ_D = 0.05 months/fault. Using Equation 6.7, the predicted number of faults M and the predicted repair time D are calculated as follows:

$$M = 2.03 \cdot 63.72 = 129 \text{ faults} \qquad D = 129 \cdot 0.05 = 6.45 \text{ months}$$

We conclude that 41% (4.42 months) of the completion time will be spent writing new code, while 59% (6.45 months) of the completion time will be spent repairing 129 faults. In the next section, we describe in detail our approach to estimating the historical mean time-to-repair faults μ_D and the historical fault density F using data from a past project at the sponsor company.

6.3.3 Sensitivity Analysis

The sensitivity of the completion time T' with respect to the number of lines of code C, is defined as shown in Equation 6.8.

$$S_C^{T'} = \frac{\Delta T'}{\Delta C}$$

Equation 6.8

For example, we can calculate the sensitivity as the change in the completion time T' in response to a 1 KNCSL increase in the number of lines of code C, as shown in Equation 6.9.

$$S_C^{T'} = \frac{C+1}{R^*} - \frac{C}{R^*} = \frac{1}{R^*}$$

Equation 6.9

We can use this sensitivity analysis to evaluate the effect of increases in the number of lines of code on the completion time. Continuing our previous example, using Equation 6.9, the sensitivity is calculated as follows:

$$S_C^{T'} = \frac{1}{5.86} = 0.17 \text{ months / KNCSL}$$

We conclude that a 1 KNCSL increase in the number of lines of code C, will cause a 0.17 month increase in the completion time T'.

The sensitivity of the completion time (no repair) T with respect to the number of lines of code C could also be calculated. This is not performed here.

6.4 Categorized Version of the Aggregate-Code Model

The categorized version of the aggregate-code model characterizes the behavior of an entire firmware project, by sub-characterizing the behavior of specific categories of firmware based on historical data. For example, a past project that consisted of Type 1 code, Type 3 code, and User Interface code, would have metrics associated with each of these categories as a means to characterize their individual and collective behavior in the future.

The categorized version of the aggregate-code model predicts performance over $i=1...m$ time periods (e.g. 1 month, 60 days, etc., where a new time period is defined by a change in the staffing level) across $j=1...n$ categories (e.g. Type 1, Type 3, User Interface, etc., where a category denotes a collection of files common to some function or feature).

The required historical parameters for the categorized version of the model are the following: the historical productivity effort rp_j' for each category j, the historical fault density f_j for each category j, and the historical mean time-to-repair faults μ_{d_j} for each category j. The required current parameters for the categorized version of the model are the following: the current number of lines of code c_j for each category j (which can be estimated using any of the aforementioned sizing methods) and the staffing level p_{ij} for each category j during each time period i (which are typically defined by managers). Given these parameters, the model outputs T' (completion time) and D (repair time) are easily determined, and a sensitivity analysis can be performed with respect to the current number of lines of code C and the staffing levels p_{ij}.

Table 6.2 lists the variables for the generalized version of the aggregate-code model.

Variable	Description	Determination	Units
c_j	Number of Lines of Code by Category†	Current	*KNCSL*
c_j*	Number of Lines of Code by Category	Historical	*KNCSL*
r_i'	Variable Productivity Rate by Time Period (Repair)	Historical	*KNCSL/Month*
r_{ij}'	Variable Productivity Rate by Time Period and by Category (Repair)	Historical	*KNCSL/Month*
rc_j'	Coding Rate by Category (Repair)	Historical	*KNCSL/Month*
rp_j'	Productivity Effort by Category (Repair)†	Historical	*KNCSL/Person-Month*
d_j	Repair Time by Category	Predicted	*Months*
d_j*	Repair Time by Category	Historical	*Months*
t_i	Time Period	Defined	*Months*
m_j	Number of Faults by Category	Predicted	*Faults*
m_j*	Number of Faults by Category	Historical	*Faults*
f_j	Fault Density by Category†	Historical	*Faults/KNCSL*
μ_{d_j}	Mean Time-to-Repair Faults by Category†	Historical	*Months/Fault*
e_j	Repair Effort by Category	Historical	*Person-Months*
μ_{e_j}	Mean Effort-to-Repair Faults by Category	Historical	*Person-Months/Fault*
p_{ij}	Staffing Level by Time Period and by Category†	Defined	*People*
μ_{p_j}	Mean Staffing Level by Category	Historical	*People*
α_j	Scaling Factor by Category	Defined	*Percent*
n_{p_j}	Faults Per Person by Category	Historical	*Faults/Person*

†*denotes a required parameter.*

Table 6.2: Variables for the Categorized Version of the Aggregate-Code Model

6.4.1 Completion Time

To predict the completion time T' we require the following inputs: the historical productivity effort rp_j' for each category j, the current number of lines of code c_j for each category j, and the staffing level p_{ij} for each category j, during each time period i.

The historical productivity effort rp_j' is used to predict the current productivity rate R'. The current productivity rate R' is calculated as a series of variable productivity rates r_i' with a different productivity rate for each time period i. The variable productivity rate for each time period r_i' is calculated as the sum of the variable productivity rates r_{ij}' for each category j, during that time period i, as shown in Equation 6.10.

$$R' \rightarrow r_i' = \sum_{j=1}^{n} r_{ij}'$$

Equation 6.10

The variable productivity rate for each category r_{ij}' is calculated as the product of the productivity effort rp_i' for that category and the staffing level for that category, during that time period, p_{ij}, as shown in Equation 6.11.

$$r_{ij}' = rp_j' \cdot p_{ij}$$

Equation 6.11

The historical productivity effort for each category rp_j' is calculated by dividing the historical coding rate for that category rc_j by the mean staffing level for that category μ_{p_j}, as shown in Equation 6.12.

$$rp_j' = \frac{rc_j'}{\mu_{p_j}}$$

Equation 6.12

The historical coding rate for each category rc_j is calculated by multiplying the historical productivity rate R^* referenced in the generalized version by a scaling factor α_j that characterizes the contribution of that category to the historical productivity rate, as shown in Equation 6.13.

$$rc_j' = \alpha_j \cdot R^*$$

Equation 6.13

The scaling factors α_j sum to 1, as shown in Equation 6.14, such that the sum of the historical coding rates rc_j is equal to the historical productivity rate R^* referenced in the generalized version, as shown in Equation 6.15.

$$\sum_{j=1}^{n} \alpha_j = 1$$

Equation 6.14

$$\sum_{j=1}^{n} rc_j' = \sum_{j=1}^{n} \alpha_j \cdot R^* = R^*$$

Equation 6.15

One approach to determine the scaling factors α_j is to assume that the contribution made by a category to the historical productivity rate is proportional to that categories percentage of the total number of lines on the past project. For example, if a category was 50% of the code on the past project, its scaling factor would be 0.50.

Finally, substituting Equation 6.13 into Equation 6.12, then Equation 6.12 into Equation 6.11, then Equation 6.11 into Equation 6.10, the variable productivity rate r_i' is calculated as shown in Equation 6.16.

$$r_i' = \sum_{j=1}^{n} \frac{\alpha_j \cdot R^*}{\mu_{p_j}} \cdot p_{ij}$$

Equation 6.16

If average staffing levels are maintained throughout the duration of the project, the variable productivity rate is equal to the fixed productivity rate. In other words, if $p_{ij} = \mu_{pj}$, the expression for r_i' in Equation 6.16 reduces to R^*. We also call the generalized version of the aggregate-code model the variable rate version because the productivity rate changes over time. A different productivity rate characterizes each time period, as fluctuations in the staffing level causes fluctuations in the productivity rate. The number of lines of code generated during each time period c_i is calculated as the product of the variable productivity rate r_i' and the duration of the time period t_i', as shown in Equation 6.17.

$$c_i = r_i' \cdot t_i'$$

Equation 6.17

When the sum of the number of lines of codes for each time period c_i (the cumulative sum of written code to-date) is equal to the final number of lines of code on the current project C (a figure which typically changes over the duration of the project, and represents the current best estimate of the number of lines of code that will be required for the entire project), the predicted completion time T' is calculated as the sum of each time period t_i', as shown in Equation 6.18.

$$T' = \sum_{i=1}^{m} t_i'$$

Equation 6.18

As an example, in the previous section we described a past project that exhibited a number of lines of code C^* = 52.37 KNCSL and a productivity rate R^* = 5.86 KNCSL/month, while the next generation of the same project was estimated to have a number of lines of code C = 63.72 KNCSL. If the past project was comprised of three categories of code (X, Y, and Z) with numbers of lines of code c_1^* = 16.76 KNCSL (X), c_2^* = 25.66 KNCSL (Y), and c_3^* = 9.95 KNCSL (Z), and mean staffing levels μ_{p_1} = 2.5 people (X), μ_{p_2} = 3.1 people (Y), and μ_{p_3} = 1.7 people (Z), the necessary parameters are calculated using Equation 6.12 and Equation 6.13 as follows (note the following: 1) the sum of the individual number of lines of codes c_j^* is equal to the total number of lines of code C^*, 2) the sum of the individual productivity rates rc_j' is equal to the total productivity rate R^*, and 3) here, we assume that the scaling factors α_j for each category are directly proportional to the percent of the total number of lines of code for that category):

Category	Code (KNCSL) c_j^*	Code (KNCSL %) α_j	Staffing (People) μ_{p_j}	Coding Rate (KNCSL/mo.) rc_j'	Prod. Effort (KNCSL/pm) rp_j'
X	16.76	0.32	2.5	1.88	0.752
Y	25.66	0.49	3.1	2.87	0.926
Z	9.95	0.19	1.7	1.11	0.653
SUM	**52.37**	**1.00**	-	**5.86**	-

Using Equation 6.16, the variable productivity rate r_i' is calculated as follows:

$$r_i' = 0.752 p_{i1} + 0.926 p_{i2} + 0.653 p_{i3}$$

Continuing our example, we assume that the next generation of the same project consists of the same categories (X, Y, and Z) with estimated numbers of lines of code c_1 = 20.46 KNCSL (X), c_2 = 32.94 KNCSL (Y), and c_3 = 10.32 KNCSL (Z), and constant staffing levels p_{i1} = 2 people (X), p_{i2} = 3 people (Y), and p_{i3} = 2 people (Z). Using Equation 6.17 and Equation 6.18, it is easy to show that the predicted completion time T' is calculated as follows:

$$T' = \frac{20.46 + 32.94 + 10.32}{(0.752 \cdot 2) + (0.926 \cdot 3) + (0.653 \cdot 2)} = \frac{63.72}{5.59} = 11.40 \text{ months}$$

We conclude that the current project will be completed in 11.40 months. In the next section, we describe in detail our approach to estimating the historical productivity effort for each category rp_j' using data from a past project at the sponsor company.

6.4.2 Repair Time

To predict the repair time D we require the following inputs: the historical fault density f_j for each category j, the historical mean time-to-repair faults μ_{d_j} for each category j, and the current number of lines of code c_j for each category j, The predicted repair time D is calculated as the sum of the predicted repair times d_j for each category j, as shown in Equation 6.19.

$$D = \sum_{j=1}^{n} d_j$$

Equation 6.19

Similarly, the predicted number of faults M is calculated as the sum of the predicted number of faults m_j for each category j, as shown in Equation 6.20.

$$M = \sum_{j=1}^{n} m_j$$

Equation 6.20

The predicted repair time for each category d_j is calculated as the product of the predicted number of faults for that category m_j and the historical mean time-to-repair faults for that category μ_{d_j}, where the predicted number of faults for each category is the product of the historical fault density for that category f_j and the current number of lines of code c_j for that category, as shown in Equation 6.21.

$$d_j = m_j \cdot \mu_{d_j} \quad \text{(a)} \qquad m_j = c_j \cdot f_j \quad \text{(b)} \qquad d_j = c_j \cdot f_j \cdot \mu_{d_j} \quad \text{(c)}$$

Equation 6.21

Continuing our previous example, suppose the past project exhibited fault densities f_1 = 2.79 faults/KNCSL (X), f_2 = 0.94 faults/KNCSL (Y), and f_3 = 4.17 faults/KNCSL (Z), and mean times-to-repair faults μ_{d_1} = 0.06 months/faults (X), μ_{d_2} = 0.05 months/fault (Y), and μ_{d_3} = 0.16 months/fault (Z). Using Equation 6.21, the predicted number of faults for each category m_j and the predicted repair time for each category d_j are calculated as follows:

$$m_1 = (2.79 \cdot 20.46) = 57 \text{ faults} \qquad d_1 = (57 \cdot 0.06) = 3.29 \text{ months}$$
$$m_2 = (0.94 \cdot 32.94) = 31 \text{ faults} \qquad d_2 = (31 \cdot 0.05) = 1.55 \text{ months}$$
$$m_3 = (4.17 \cdot 10.32) = 43 \text{ faults} \qquad d_3 = (43 \cdot 0.16) = 2.03 \text{ months}$$

We conclude that category X is likely to cause the largest number of errors (57 faults) and require the largest amount of time to repair these faults (3.29 months). Using Equation 6.19 and Equation 6.20, the predicted number of faults M and the predicted repair time D are calculated as follows:

$$M = 57 + 31 + 43 = 131 \text{ faults} \qquad D = 3.29 + 1.55 + 2.03 = 6.87 \text{ months}$$

We conclude that 40% (4.53 months) of the completion time will be spent writing new code, while 60% (6.87 months) of the completion time will be spent repairing 131 faults. In the next section, we describe in detail our approach to estimating the historical fault density f_j and the historical mean time-to-repair faults μ_{d_j} using data from a past project at the sponsor company.

6.4.3 Sensitivity Analysis

The sensitivity of the completion time T' with respect to the number of lines of code C, is defined as shown in Equation 6.22.

$$S_C^{T'} = \frac{\Delta T'}{\Delta C}$$

Equation 6.22

For example, we can calculate the sensitivity as the change in the overall completion time T' in response to a 1 KNCSL increase in number of lines of code C, as shown in Equation 6.23 (note that we are only concerned with the effect that such an increase has during the final time period m, because it is during this time period that resources would be dedicated to addressing the additional code).

$$S_{c_m}^{T'} = \frac{C - \left(c_{m-1} + r_m' \cdot t_m'\right)}{r_m'} - \frac{(C+1) - \left(c_{m-1} + r_m' \cdot t_m'\right)}{r_m'} = \frac{1}{r_m'}$$

Equation 6.23

We can use this sensitivity analysis to evaluate the effect of code increases on the completion time. Continuing our previous example, using Equation 6.23, the sensitivity is calculated as follows:

$$S_{c_m}^{T'} = \frac{1}{5.59} = 0.18 \text{ months / KNCSL}$$

We conclude that a 1 KNCSL increase in the number of lines of code C, will cause a 0.18 month increase in the completion time T'. The sensitivity of the completion time T' with respect to the staffing level in a category, during a time period, p_{ij}, is defined as shown in Equation 6.24.

$$S_{p_{ij}}^{T'} = \frac{\Delta T'}{\Delta p_{ij}}$$

Equation 6.24

For example, we can calculate the sensitivity as the change in the overall completion time T' in response to a 1 person increase in p_{mj}, the staffing level during the final time period m in category j, as shown in Equation 6.25 (note the following: 1) r_{m_j}'' denotes the new variable productivity rate resulting from an

additional person, and 2) $c_{(m-1)_j}$ denotes the number of lines of code in the next to last time period $m-1$ for category j).

$$S^{T'}_{p_{m_j}} = \frac{C-\left(c_{(m-1)_j}+r'_m\cdot t'_m\right)}{r'_m} - \frac{C-\left(c_{(m-1)_j}+r''_{m_j}\cdot t'_m\right)}{r''_{m_j}} = \frac{rp'_j\cdot\left(c_{(m-1)_j}-C\right)}{r'_m\cdot r''_{m_j}} \qquad \text{where} \quad r''_{m_j} = r'_m + rp'_j$$

Equation 6.25

If we sum these values for each of the m time periods in a given category, we can also calculate the sensitivity of the completion time to a 1 person increase in the staffing level for that category throughout the duration of the project, as shown in Equation 6.26.

$$S^{T'}_{p_j} = \sum_{i=1}^{m} S^{t'_i}_{p_{ij}}$$

Equation 6.26

We can use the sensitivity analysis to evaluate the effect of increasing the staffing level in a category on the completion time, during a specific time period or throughout the duration of a project. Continuing our example, we assume that the time periods t'_i are of equal length (1 month). Since the project is completed in 11.40 months, at the end of the next to last time period $m-1=11$ the amount of code that has been generated is (5.59 KNCSL/month) · (11 months) = 61.49 KNCSL. Using Equation 6.25, the sensitivities during the final time period $m=12$ are calculated as follows:

$$S^{T'}_{p_{m_1}} = \frac{0.752\cdot(61.49-63.72)}{5.59\cdot(5.59+0.752)} = -0.05 \text{ months}$$

$$S^{T'}_{p_{m_2}} = \frac{0.926\cdot(61.49-63.72)}{5.59\cdot(5.59+0.926)} = -0.06 \text{ months}$$

$$S^{T'}_{p_{m_3}} = \frac{0.653\cdot(61.49-63.72)}{5.59\cdot(5.59+0.653)} = -0.04 \text{ months}$$

We conclude that a 1 person increase in the staffing level of category Y during the last time period will yield the largest decrease in the completion time T' from 11.40 months to 11.34 months. Using Equation 6.26 to sum the values for all of the 12 time periods, the sensitivities for the duration of the project are: X (-1.35 months), Y (-1.62 months), and Z (-1.19 months). We conclude that a 1 person increase in the staffing level of category Y throughout the duration of the project will yield the largest decrease in the completion time T' from 11.40 months to 9.78 months.

The sensitivity of the completion time (no repair) T with respect to the number of lines of code C could also be calculated. This is not performed here.

6.5 Research Methodology

A digital wireless telephone consists of two kinds of firmware: microcontroller firmware and digital signal processor firmware. At the sponsor company, two functional groups were responsible for coding and testing this firmware, the Microcontroller Design group and the Digital Signal Processor Design group, respectively. To assess the aggregate-code model's potential utility, it was applied to the firmware in the Microcontroller Design group only.

To determine the historical parameters for both versions of the aggregate-code model, we extracted data from the product administration database at the sponsor company. This database stored the actual source lines of code and managed changes to these files by tracking what are known as modification requests (MRs). MRs represent petitions to change code as a result of malfunctions or errors that have been identified during coding, integration, or testing. The parameters that we were interested in extracting from the database were the number of faults, the mean time-to-repair faults, and the fault density on a past project. These parameters represented necessary inputs to the model. Unfortunately, MRs did not capture these measures. MRs did record a wide range of related data including the creation date (the date the MR was generated), the submission date (the date the changes to code were finished), and the completion time (the difference between the submission date and the creation date). To approximate the model inputs, we made the following basic assumptions:

$$\#\,MRs \approx \#\,Faults \tag{1}$$

$$MR\ Completion\ Time \approx Time\text{-}To\text{-}Repair\ Faults \tag{2}$$

$$\frac{\#\,MRs}{KNCSL} \approx Fault\ Density \tag{3}$$

The first assumption states that the number of modification requests was proportional to the number of faults. In a number of instances, the ratio of MRs to faults was indeed 1 to 1. However, there were also instances where a single fault generated multiple MRs (which would skew the ratio above 1), or a single MR referenced multiple faults (which would skew the ratio below 1). Despite these and other anomalies, this assumption was considered valid based on the opinion that by aggregating the data, an averaging effect would take place, causing the ratio to be close to 1.

The second assumption states that the time to complete an MR was proportional to the mean time-to-repair faults. Presumably, between the time an MR was created and the time it was submitted, the programmer(s) assigned to the MR were working to fix the problem that was identified. Clearly, there is noise associated with this assumption given the fact that programmers were often responsible for multiple MRs, while also attending to other non MR-related responsibilities. However, given the data that was available from the product administration database, this was the closest approximation to the mean time-to-repair faults.

The third assumption states that the number of MRs divided by the number of lines of code was proportional to the fault density. This assumption was valid to the extent that the first assumption was valid because the number of lines of code was an exact statistic. The MR data was calculated directly from the product administration database and the number of lines of code was calculated directly from the source code directories, both using UNIX shell scripts.

In the following sections, when referring to the number of faults, the mean time-to-repair faults, and the fault density, we will actually be referring to the *approximations* to these metrics, unless explicitly stated otherwise. We will also express all time-dimensioned variables in days as opposed to months (e.g. days/fault and person-days/fault, etc.). Using these approximations, we calculated the necessary parameters for the generalized version of the aggregate-code model and the categorized version of the aggregate-code model, from a past project at the sponsor company.

6.5.1 Generalized Version and Parameters

For the generalized version of the aggregate-code model, we calculated model parameters that sought to characterize the collective behavior of the microcontroller firmware on the past project. The four parameters that served as inputs to the model were the following: the historical productivity rate R^*, the historical fault density F, the historical mean time-to-repair faults μ_D, and the current number of lines of code C. Here, we describe our approach to determining the historical parameters.

The historical productivity rate R^* was easily determined from the past project at the sponsor company. Recall from Section 6.3.1, that the historical productivity rate R^* can be calculated by dividing the number of lines of code for the past project C^* by the time period required to complete the past project T^*, as shown in Equation 6.27.

$$R^* = \frac{C^*}{T^*}$$

Equation 6.27

The historical productivity rate R^* was calculated as follows:

$$R^* = \frac{75.43}{455} = 0.166 \text{ KNCSL / day}$$

The historical fault density F was calculated by dividing the number of faults on the past project M^* by the number of lines of code on the past project C^*, as shown in Equation 6.28.

$$F = \frac{M^*}{C^*}$$

Equation 6.28

The historical fault density F was calculated as follows:

$$F = \frac{636}{75.43} = 8.43 \text{ faults / KNCSL}$$

The historical mean time-to-repair faults μ_D was calculated by dividing the repair time on the past project D^* by the number of faults on the past project M^*, as shown in Equation 6.29.

$$\mu_D = \frac{D^*}{M^*}$$

Equation 6.29

The product administration database did not provide the repair time on the past project, instead it provided the repair effort E on the past project (measured in person-days). We postulated that the repair effort was equal to the product of the repair time D^*, the number of faults repaired per day N_D, and the mean effort-to-repair faults μ_E, as shown in Equation 6.30.

$$E = D^* \cdot N_D \cdot \mu_E$$

Equation 6.30

In other words, the effort overstated the amount of time spent repairing faults because of parallel activity. Therefore, for each day spent repairing a fault (repair days), a number of faults were being addressed within that time (faults per repair day), and for each of these faults there was an effort associated with its repair (effort per fault). This expression in Equation 6.30 cancels as follows:

$$\text{Effort} = \frac{\cancel{\text{Repair Days}}}{} \times \frac{\cancel{\text{Faults}}}{\cancel{\text{Repair Day}}} \times \frac{\text{Effort}}{\cancel{\text{Fault}}}$$

Solving Equation 6.30 for D^*, and substituting this expression into Equation 6.30, the final expression for the historical mean time-to-repair faults μ_D is shown in Equation 6.31.

$$\mu_D = \frac{E}{M^* \cdot N_D \cdot \mu_E}$$

Equation 6.31

The number of faults repaired per day N_D was estimated by managers to be 2.50 faults/day. The mean effort-to-repair faults μ_E was estimated from the product administration database to be 20.37 person-days/fault. Finally, the historical mean time-to-repair faults μ_D was calculated as follows:

$$\mu_D = \frac{12956}{636 \cdot 2.50 \cdot 20.37} = 0.40 \text{ days / fault}$$

Table 6.3 summarizes the completion time and repair time parameters for the generalized version of the aggregate-code model.

Variable	Description	Value
R^*	Productivity Rate (Repair)†	0.166 KNCSL/day
E	Repair Effort	12,956 person-days
N_D	Faults Per Repair Day	2.50 faults/day
μ_E	Mean Effort-to-Repair Faults	20.37 person-days/fault
μ_D	Mean Time-to-Repair Faults†	0.40 days/fault
F	Fault Density†	8.43 faults/KNCSL

†denotes an input parameter.

Table 6.3: Generalized Version of the Aggregate-Code Model Completion Time and Repair Time Parameters

This completed the necessary parameters for the generalized version of the aggregate-code model. In summary, the required historical inputs were the historical productivity rate R^*, the historical mean time-to-repair faults μ_D, and the historical fault density F. Given these historical parameters and an estimate of the current number of lines of code C, the model could predict the completion time T' and the repair time D.

6.5.2 Categorized Version and Parameters

For the categorized version of the aggregate-code model, we calculated model parameters that sought to characterize the individual and collective behavior of the microcontroller firmware on the past project. Toward that end, the firmware on the past project (e.g. each file) was segmented into the following six categories *(j=1...6)*:

- **Type 1** - software that exchanged messages with the base stations so that calls could be originated and terminated, and that handoffs could be performed between base stations. Type 1 software also supported Type 1 services software.
- **Infrastructure** - software that formed the run-time environment for the rest of the software including the operating system software, interprocessor communication software, and memory systems management software.
- **Type 1 Services** - software that provided vertical services between the handset and the base station utilizing the basic Type 1 capabilities of exchanging messages. Such services included Type 2 messages, authentication, Type 3 service provisioning, and more.
- **Type 3** - software that allowed the handset to receive and store the parameters associated with service provisioning via Type 3 messaging, instead of being cabled to a computer.

- **User Interface** - software that interacted with the user via display and keypad/buttons, to present services to the user.
- **Include and Other** - files that specified data structures and defined the rules for compiling and building the software.

The four parameters that served as inputs to the model were: the historical productivity effort rp_j' for each category j, the historical fault density f_j for each category j, the historical mean time-to-repair faults μ_{d_j} for each category j, and the current number of lines of code c_j for each category j. Here, we describe our approach to determining the historical parameters.

The historical productivity effort rp_j' was easily determined from the past project at the sponsor company. Recall from Section 6.4.1, that the historical productivity effort rp_j' is defined as shown in Equation 6.32.

$$rp_j' = \frac{\alpha_j \cdot R^*}{\mu_{p_j}}$$

Equation 6.32

The historical productivity rate R^* was already determined for the generalized version of the model. Therefore, the only unknowns in the expression were the scaling factors α_j and the mean staffing levels μ_{p_j}. To determine the scaling factors α_j, we assumed that the contribution made by a category to the historical productivity rate R^* was proportional to that categories percentage of the total number of lines of code on the past project. For example, if a category was 50% of the code on the past project, its scaling factor was 0.50. The mean staffing levels μ_{p_j} were determined from actual personnel records. Table 6.4 lists the completion time parameters for the categorized version of the aggregate-code model.

Category	Code (KNCSL) c_j	Code (KNCSL %) α_j	Staffing (people) μ_{p_j}	Coding Rate (KNCSL/day) rc_j	Prod. Effort (KNCSL/pd) rp_j'
Type 1	20.37	0.27	2.0	0.045	0.022
Infrastructure	16.59	0.22	2.5	0.036	0.015
Type 1 Services	1.51	0.02	2.5	0.003	0.001
Type 3	15.84	0.21	2.0	0.035	0.017
User Interface	12.82	0.17	2.2	0.028	0.013
Include and Other	8.30	0.11	3.3	0.018	0.006
TOTAL	75.43	1.00	-	0.166	-

†denotes an input parameter.

Table 6.4: Categorized Version of the Aggregate-Code Model Completion Time Parameters

The historical fault density for each category f_j was calculated by dividing the number of faults for each category m_j* on the past project by the number of lines of code for each category c_j* on the past project, as shown in Equation 6.33.

$$f_j = \frac{m_j^*}{c_j^*}$$

Equation 6.33

The historical mean time-to-repair faults for each category μ_{d_j} was calculated by dividing the repair time for each category d_j* on the past project by the number of faults for each category m_j* on the past project, as shown in Equation 6.34.

$$\mu_{d_j} = \frac{d_j^*}{m_j^*}$$

Equation 6.34

As mentioned earlier, the product administration database did not provide repair times on the past project, instead it provided repair efforts for each category e_j on the past project (measured in person-days). We postulated that the repair effort for a category was equal to the product of the repair time for that category d_j*, the number of people that worked during each repair day for that category μ_{p_j}, the number of faults being addressed by each person for that category n_{p_j}, and the mean effort-to-repair faults for that category μ_{e_j}, as shown in Equation 6.35.

$$e_j = d_j^* \cdot \mu_{p_j} \cdot n_{p_j} \cdot \mu_{e_j}$$

Equation 6.35

In other words, the effort again overstated the amount of time spent repairing faults because of parallel activity. Therefore, for each day spent repairing a fault (repair days), a number of people were working to repair these faults (people per repair day), and each of these people were addressing a certain number of faults (faults per person), and for each of these faults there was an effort associated with its repair (effort per fault). This expression in Equation 6.35 cancels as follows:

$$\text{Effort} = \frac{\cancel{\text{Repair Days}}}{} \times \frac{\cancel{\text{People}}}{\cancel{\text{Repair Day}}} \times \frac{\cancel{\text{Faults}}}{\cancel{\text{Person}}} \times \frac{\text{Effort}}{\cancel{\text{Fault}}}$$

Solving Equation 6.35 for d_j*, and substituting this expression into Equation 6.34, the final expression for the mean time-to-repair faults μ_{d_j}, is shown in Equation 6.36.

$$\mu_{d_j} = \frac{e_j}{m_j^* \cdot \mu_{p_j} \cdot n_{p_j} \cdot \mu_{e_j}}$$

Equation 6.36

The number of people that worked during each repair day for a category μ_{p_j} was estimated as the mean staffing level for the past project (previously obtained from personnel records for the generalized version of the model). The number of faults worked by each person in a category n_{p_j} was estimated by managers to be 1.15. The mean effort-to-repair faults for a category μ_{e_j} was estimated from the product administration database. Table 6.5 lists the repair time parameters for the categorized version of the aggregate-code model.

This completed the necessary parameters for the categorized version of the aggregate-code model. In summary, the required historical inputs were the historical productivity effort rp_j' for each category j, the historical fault density f_j for each category j, and the historical mean time-to-repair faults μ_{d_j} for each category j. Given these historical parameters and an estimate of the current number of lines of code c_j for each category j, the model could predict the completion time T' and the repair time D.

Category	Repair Effort *(person-days)* e_i	Number of Faults m_i	Faults Per Person μ_{b_i}	Effort-to-Repair Faults *(days/ fault)* μ_{f_i}	Time-to-Repair Faults *(days/fault)* μ_{d_i}	Fault Density[1] *(faults/ KNCSL)* f_j
Type 1	2810	267	1.15	10.52	0.43	13.11
Infrastructure	2584	140	1.15	18.46	0.35	8.44
Type 1 Services	179	6	1.15	29.83	0.35	3.97
Type 3	465	13	1.15	35.77	0.43	0.82
User Interface	4264	89	1.15	47.91	0.40	6.94
Include and Other	2657	121	1.15	21.96	0.27	14.58
TOTAL	**12959**	**636**	-	-	-	-

[1] *denotes an input parameter.*

Table 6.5: Categorized Version of the Aggregate-Code Model Repair Time Parameters

6.6 Results

The aggregate-code model provided valuable insight from two perspectives. First, the metrics that served as inputs to the model were useful in assessing the performance of the past project from which they were obtained. Second, the model's ability to predict future completion times and future repair times was useful in assessing performance on subsequent projects of a similar nature. In the following section, we present observations and conclusions that were drawn as a result of analyzing the aggregate-code model metrics and projections. Finally, we provide suggestions on how to interpret this data.

6.6.1 Model Metrics

The data that was collected to calculate the inputs to the aggregate-code model (e.g. repair effort, number of faults) and the inputs themselves (e.g. productivity rate, fault density, and mean time-to-repair faults), were useful in assessing the performance of the past project from which they were obtained.

Table 6.6 lists the inputs to both versions of the aggregate-code model, derived from the past project at the sponsor company (note that for consistency we have converted the generalized productivity rate R' from KNCSL/day to KNCSL/person-day by dividing 0.166 KNCSL/day by an average staffing level of 14.4 people on the past project).

GENERALIZED VERSION	Productivity *(KNCSL/pd)* R'	Time-to-Repair *(days/fault)* μ_{D}	Fault Density *(faults/KNCSL)* Γ
Overall	**0.012**	**0.40**	**8.43**
CATEGORIZED VERSION	rp_i'	μ_{d_i}	f_j
Type 1	0.022	0.43	13.11
Infrastructure	0.015	0.35	8.44
Type 1 Services	0.001	0.35	3.97
Type 3	0.017	0.43	0.82
User Interface	0.013	0.40	6.94
Include and Other	0.006	0.27	14.58

Table 6.6: Aggregate-Code Model Metrics

The productivity numbers were not particularly useful in that the noticeable, and intuitive, trend was that the categories that had the largest amounts of code on the past project, generally had the highest productivity's. Type 1 had the largest amount of code on the past project (27%) and had the highest productivity (0.022 KNCSL/person-day). Type 1 Services had the smallest amount of code on the past project (2%) and had the lowest productivity (0.001 KNCSL/person-day). This was partially a result of our selection criteria for the scaling factors.

We did notice that the overall fault density for the project was 8.43 faults/KNCSL. This was impressively close to manager's initial estimate of 8 faults/KNCSL at the beginning of the project. We also noticed that the mean time-to-repair faults did not exhibit huge fluctuations across the various categories. The difference between the highest ranked categories (Type 1 and Type 3, both at 0.43 days/fault) and the lowest ranked category (Include and Other at 0.27 days/fault) was only 0.16 days/fault. However, it was important to note that in addition to being ranked first by the mean time-to-repair faults, the Type 1 category was also ranked second by fault density (13.11 faults/KNCSL). This was in sharp contrast to the Type 3 category which was also ranked first by the mean time-to-repair faults, but ranked last by fault density (0.82 faults/KNCSL). We can expand on this observation by examining the remaining data.

Table 6.7 lists the number of lines of code, repair effort, and number of faults for each category of the past project as a percent of the total number of lines of code (75.43 KNCSL), the total repair effort (12,959 days), and the total number of faults (636 faults), respectively, derived from the past project at the sponsor company.

Category	Code (KNCSL) c_j	Repair Effort (person-day) e_j	Number of Faults m_j
Type 1	27%	22%	42%
Infrastructure	22%	20%	22%
Type 1 Services	2%	1%	1%
Type 3	21%	4%	2%
User Interface	17%	33%	14%
Include and Other	11%	20%	19%
TOTAL	**100%**	**100%**	**100%**

Table 6.7: Aggregate-Code Model Data as a Percent of Total

From this data a number of observations were made. First, it was concluded that Type 1 software was the most challenging area on the past project, and an area that would require additional resources on future product generations. Type 1 represented 27% of the code, yet generated almost twice as many faults (42%). As mentioned earlier, the Type 1 category was also ranked second by fault density. It was also the only category to exhibit a double-digit fault density, other than the Include and Other category. The need for particular emphasis to be placed on Type 1 in the future was further bolstered by the fact that one would expect a high fault density from the Include and Other category. This category represented code that defined data structures and rules for compiling and building the software. Consequer[illegible]ase files had a high likelihood of experiencing change, not due to any inherent problem w[illegible]itself, but rather because they included global specifications that had a strong interdepende[illegible]ll of the other categories. A change to a file in any of the other categories typically necessitated a c[illegible]ge to a file in the Include and Other category.

Second, it appeared that performance in the Type 3 category was impressive (21% of the code, yet only 4% of the repair effort, 2% of the faults, and the lowest fault density at 0.89 faults/KNCSL). In reality, these numbers were slightly skewed given the fact that the software in this category had undergone extensive development in a separate environment, prior to being integrated onto the handset. Consequently, many of the potential problems with the Type 3 code were identified and resolved prior to being tracked by the product administration database. This explained why the repair effort, number of faults, and fault density for the Type 3 category were so low.

Third, it appeared that the User Interface software required a significant amount of time to resolve problems (ranked first by repair effort at 4,264 person-days and third by the mean time-to-repair faults at 0.40 days/fault). Upon closer investigation, it was revealed that many of the faults that were identified in this category were actually very easy to fix. For example, a typical User Interface MR referenced some inconsistency between the display text as it appeared during testing and the display text as it was specified in the requirements document. Because fixing problems such as these was relatively straightforward, it was believed that the User Interface's high ranking by repair effort and the mean time-to-repair faults was primarily a result of programmers' tendency to allow the associated MR to remain open, while attending to more pressing matters. These rankings were not believed to be indicative of the difficulty associated with addressing such problems. These measures also had to be tempered by the fact that they did not take into account the amount of code that was changed for a given fault. For example, a User Interface fault may have required changing a single line of code, while a Type 1 fault may have required changing hundreds of lines of code.

Lastly, the performance metrics for the Infrastructure category appeared to be both consistent with the size, and commensurate with the nature of this code (ranked third by productivity, third by the mean time-to-repair faults, and third by fault density, while representing 22% of the code, 20% of the repair effort, and 22% of the faults). In fact, most of the Infrastructure metrics were close, if not identical, to the overall project metrics.

6.6.2 Model Projections

The aggregate-code model's ability to predict future completion times and future repair times represented a useful tool to assess the performance of second and third generation products of a similar nature. For example, the performance of a subsequent project consisting of revisions to Type 1 Services code and Type 3 code, could be easily assessed using the metrics from these categories only. At the time this assignment was performed, the sponsor company was completing the product that was the focus of this research. Therefore, the opportunity to implement the model on the next generation of the same product and benchmark its performance against their traditional approach was unfortunately not available. As an alternative, and as a simple means to demonstrate the model's potential usefulness, we generated model projections on the past project to assess how the model *would have* performed if it had been available to managers at the beginning of the project. In doing so, we were fully cognizant that we were using historical metrics to model the performance of the same project from which the metrics were derived. The primary reason for generating the projections was to demonstrate how the model could be used.

Figure 6.3 shows the projections using the generalized version (fixed rate) of the aggregate-code model.

As expected, the generalized version of the model predicted the completion time for the project exactly (455 days). This was because the productivity rate, which governed the completion time, was based on the number of lines of code and completion time for the project itself. From these projections, we observed that an estimated 254 days (56%) of the completion time were spent repairing faults. We also observed that the proliferation of 636 faults doubled the completion time, extending it from 8 months to 16 months. Finally, the sensitivity of the completion time with respect to the number of lines of code was 6.03 days/KNCSL. In other words, a 1 KNCSL increase in the number of lines of code would have added approximately 6 days to the completion time.

Figure 6.4 shows the projections using the categorized version (variable rate) of the aggregate-code model.

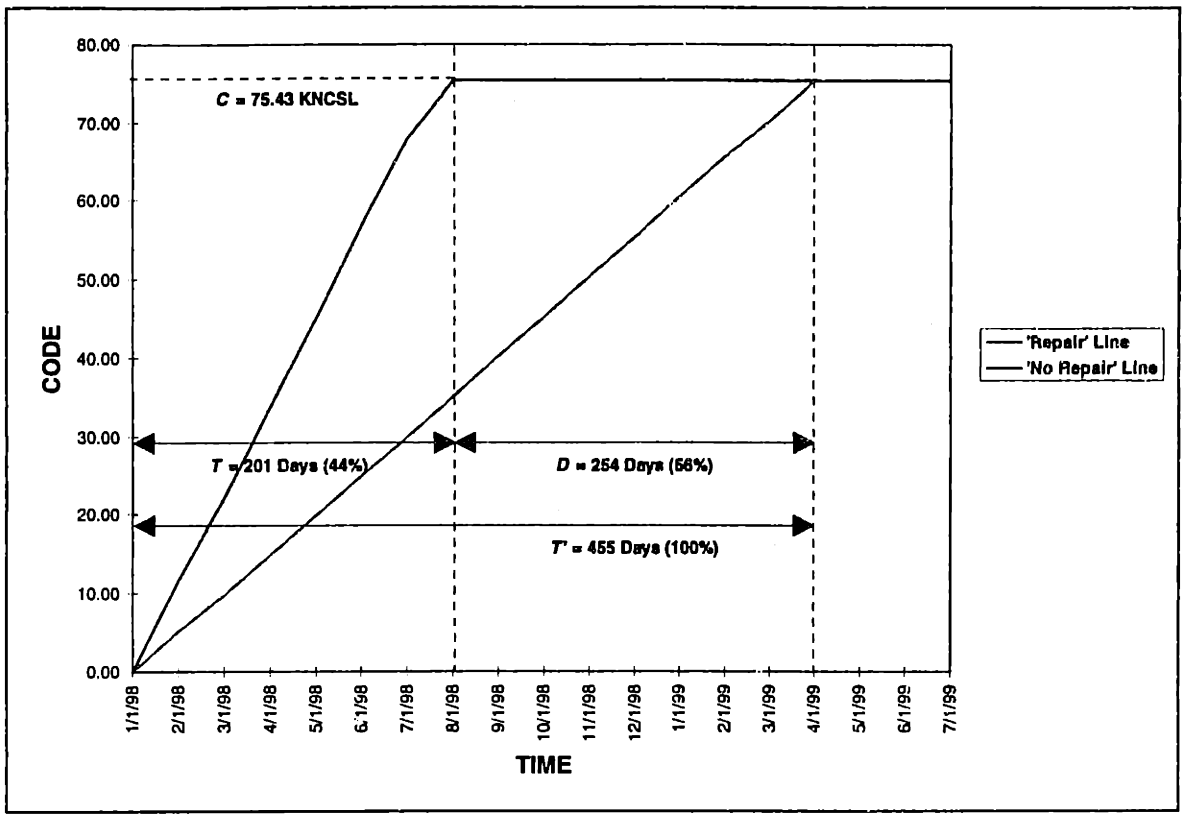


Figure 6.3: Generalized Version of the Aggregate-Code Model Projections

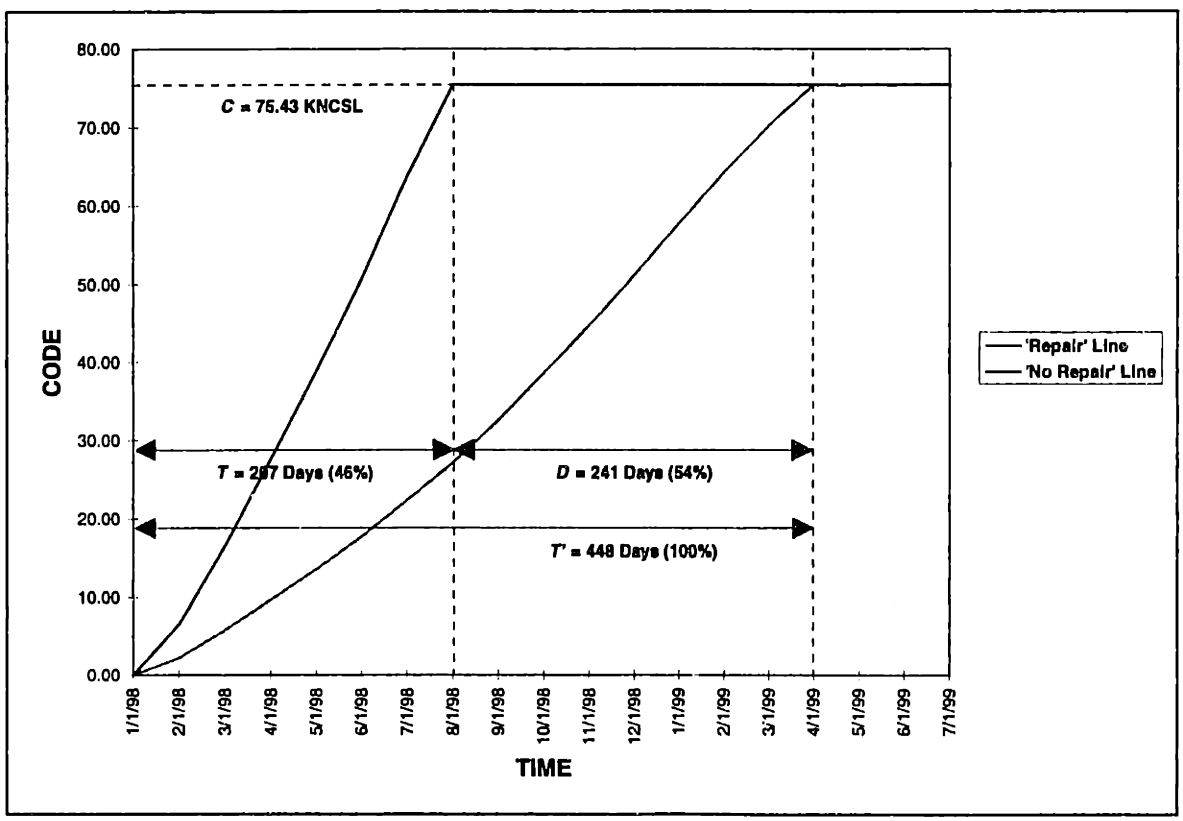


Figure 6.4: Categorized Version of the Aggregate-Code Model Projections

The categorized version of the model predicted the completion time for the project within 7 days (448 days) of the completion time that was predicted by the generalized version of the model. The slight loss of accuracy was the result of the added flexibility the model provided. By segmenting the productivity rate into different categories, we no longer matched the actual productivity rate. The corresponding benefit was the availability of more focused metrics that could be used independently, depending on the nature of the code on subsequent projects. The repair time was estimated within 13 days (241 days or 54% of the completion time) of the repair time that was estimated by the generalized version of the model. The categorized version also determined that the proliferation of 636 faults doubled the completion time, extending it from 8 months to 16 months. The sensitivity of the completion time with respect to the number of lines of code was 4.70 days/KNCSL. In other words, a 1 KNCSL increase in the number of lines of code would have added approximately 5 days to the completion time.

Finally, the sensitivities of the completion time with respect to the staffing levels throughout the duration of the project were ranked as follows: 1. *Type 1* (-42.6 days/person), 2. *Type 3* (-33.9 days/person), 3. *Infrastructure* (-29.3 days/person), 4. *User Interface* (-25.6 days/person), 5. *Include and Other* (-11.4 days/person), and 6. *Type 1 Services* (-2.8 days/person). Again, this reinforced the need for additional resources in the area of Type 1 on future product generations.

6.6.3 Interpreting the Data

A few points must be made in regard to interpreting the aggregate-code data. First, the sensitivity analysis with respect to the number of lines of code is useful in evaluating the impact additional lines of code may have on the completion time. However, it is likely that these sensitivities actually overstate the impact such increases will have. For example, on the past project at the sponsor company there were 75.43 KNCSL written over 455 days. This corresponded to a sensitivity of 6.03 days/KNCSL. There were a number of times throughout the project's development that the number of lines of code was actually stagnant, and staffing resources were solely focused on repairing faults. While this served to increase the completion time, the number of lines of code remained fixed. This would indicate that the sensitivity is actually a worst-case sensitivity, and the real sensitivity is somewhat lower. A best-case sensitivity could be estimated by dividing the number of lines of code by the no-repair completion time of 201 days, which corresponds to a sensitivity of 2.66 days/KNCSL. This probably understates the true sensitivity.

Last, the sensitivity analysis with respect to the staffing level is useful in identifying areas that may require additional resources to generate new code, but it is not useful in identifying areas that may require additional resources to repair faults. These sensitivities are a function of the productivity rates for each category, and are therefore valuable in identifying areas that could benefit from programmers dedicated to writing code, not repairing code. Since these sensitivities are also partially based on historical data, the areas they identify will inevitably be categories that in the past required significant amounts of code to be written. Once again, this was partially the result of our selection criteria for the scaling factors. The fault densities are the best metrics for identifying areas that may experience large numbers of faults, and could therefore benefit from additional resources to improve the ability to repair code. Furthermore, the staffing sensitivities are best interpreted relative to one another, instead of as absolute numbers. For example, on the past project at the sponsor company the highest sensitivity was associated with Type 1 at -42.6 days/person. It is doubtful that the completion time would have been reduced by 43 days if an additional person had been added to the project. However, relative to the other categories, we could conclude that additional resources in the area of Type 1 would have been likely to have the greatest impact on reducing the completion time.

6.7 Discussion

In this chapter, we introduced the Aggregate-Code Model, a tool that was used to predict the completion time and the repair time of large-scale firmware projects at the sponsor company. We presented two versions of the model. The generalized version of the model derived the model inputs by aggregating historical data from all of the code on a past project. The categorized version of the model derived the model inputs by aggregating historical data from specific categories of code on the same project. The model also yielded a sensitivity analysis with respect to the number of lines of code and the staffing levels in each category.

The aggregate-code model served two simultaneous purposes. The model's first purpose was to provide a means to obtain performance metrics from the firmware product administration database at the sponsor company. Based on some simple assumptions regarding the nature of modification requests (MRs), we designed UNIX shell scripts to extract productivity rates, code counts, fault counts, repair efforts, and fault densities, directly from the database. By postulating on how the effort numbers were generated, we were able to derive the mean time-to-repair faults. These metrics were useful to the sponsor company in assessing their performance on the past project. In fact, while developing the model it became clear that the sponsor company could improve their practices for documenting MRs. One of the recommendations to the sponsor company was to record the actual number of faults and the actual time-to-repair faults on firmware projects in the future, to avoid the need for approximating these important metrics in the future. Recommendations are summarized in Chapter 7. The model's second purpose was to establish a framework to supplement existing techniques at the sponsor company for predicting completion time, while also allowing for more sophisticated models to be implemented in the future. In fact, there are examples of software models that require many of the same inputs as the aggregate-code model, such as the Software Life Cycle Model [27]. One of the recommendations to the sponsor company was to implement an advanced model to predict firmware completion time, using the aggregate-code model as a starting point. Recommendations are summarized in Chapter 7.

The aggregate-code model demonstrated a number of strengths. It was based on historical data that was specific to the sponsor company. A number of software models are based on industry statistics, as opposed to firm-specific statistics [27]. These approaches are useful in the absence of indigenous metrics, however, data that is reflective of the actual working environment is almost always preferred. The model also provided considerable flexibility given the fact that the metrics were divided into categories. It allowed the model to maintain its usefulness for subsequent projects that might be limited to only certain types of code. The tradeoff to this flexibility was perhaps a slight loss in accuracy.

On the other hand, the model demonstrated an even greater number of weaknesses. Despite the flexibility offered by categorizing the code, the model did not offer a sensitivity analysis with respect to the amounts of code within these categories. Even the sensitivity analysis in the categorized version of the model was still at an aggregate level. The reason being that the model assumed that the net productivity rate of the individual productivity rates was equally applied to all of the categories of code. A better approach would have been to apply the productivity rates to their respective categories only, and calculate the amount of code being generated in each category. Furthermore, in performing this research we assumed that the contribution of each category to the productivity rate was proportional to its percent of the total number of lines of code on the past project. It is not clear that this was a good assumption as it played a noticeable role in skewing the productivity efforts and the sensitivity analyses with respect to the number of lines of code. It is likely that there may have been a more effective approach to determining the scaling factors using criteria that better characterized these contributions, such as the level of skill and experience of the team members in each category. The model also assumed a linear relationship between the number of faults and the repair time. Several research efforts have shown this relationship to be extremely nonlinear [25]. Lastly, the model assumed that by determining the time spent writing code and repairing faults on past projects, that one could predict the time spent writing code and repairing faults on future projects, without taking into account any measure of project complexity. Looking forward, the model will only be useful if the complexity of future projects is in fact equal to, or comparable to, the complexity of the past project.

There are two primary challenges in developing a firmware model. First, productivity is a difficult measure to quantify. A project could remain fixed at a certain number of lines of code for months and still be considered productive if problems are being detected and fixed. Alternatively, the number of lines of code could actually diminish as a result of more efficient coding techniques that achieve the same performance in fewer instructions. Second, the nature of faults is very difficult to characterize. A single fault can require months to be resolved, while multiple faults can be resolved within days. In developing this framework, it was clear that there was considerable work to be done at the sponsor company to create a model that could overcome these challenges. However, it was also believed that the aggregate-code model was a positive step toward such a realization.

CHAPTER 7: CONCLUSION

7.1 Recommendations

At the completion of this research, a number of strategic recommendations were presented to the sponsor company. These recommendations were as follows:

- *Establish a cross-functional team of representatives from the Industrial Design, Supply Chain Management, Manufacturing, and Reliability groups, charged with the joint responsibility of acquiring parts and qualifying suppliers.* Information transferred between these groups regarding parts and suppliers was identified as a critical component of the handset's design effort. This recommendation was the result of the information flow and dependencies analysis described in Chapter 3.

- *Invest more time at the beginning of the product development process specifying the requirements for firmware, particularly Microcontroller firmware.* The handset's Microcontroller Design group was found to be heavily dependent on requirements-related information that was not properly defined, in order to complete their responsibilities. This recommendation was the result of the information flow and dependencies analysis described in Chapter 3.

- *Establish a cross-functional strategic design team of representatives from the Manufacturing, Analog Design, Industrial Design, and Supply Chain Management groups, centered around the analog printed circuit board (PCB).* Acquiring parts for, and building the analog PCB was identified as a critical component of the handset's design effort. This recommendation was the result of the controlling features and total work analysis described in Chapter 4.

- *Establish a cross-functional strategic design team of representatives from the Microcontroller Design, System Integration, and Digital Signal Processor Design groups, centered around Type 1 firmware.* Designing and testing the handset's Type 1 firmware was identified as a critical component of the design effort. This recommendation was the result of the controlling features and total work analysis described in Chapter 4.

- *Reduce the number of audio/acoustics related tests.* Noticeable redundancy of tests related to the handset's audio/acoustics system was identified in the Industrial Design, Reliability, Human Factors, and System Test groups. This recommendation was the result of the controlling features and total work analysis described in Chapter 4.

- *Reduce the number of user interface related tests.* Noticeable redundancy of tests related to the handset's user interface was identified in the System Test, Human Factors, and System Integration groups. This recommendation was the result of the controlling features and total work analysis described in Chapter 4.

- *Implement the Dynamic Signal Flow Graph Model as a tool for modeling hardware completion time in the future.* The dynamic signal flow graph model predicted completion time of past hardware prototypes within 4% of the actual completion time. This recommendation was the result of the completion time modeling of hardware described in Chapter 5.

- *Establish procedures to record the actual number of faults and the actual time-to-repair faults on firmware projects in the future.* At the time, these metrics were not being collected, yet they represented important measures for assessing firmware performance and implementing models to predict firmware completion time. This recommendation was the result of the completion time modeling of firmware described in Chapter 5.

- *Implement an advanced model to predict firmware completion time, using the Aggregate-Code Model as a starting point.* The aggregate-code model provided a framework to supplement existing techniques, yet allowed for more sophistical models to be implemented. A model that was suggested was the Software Life Cycle Model [27]. This recommendation was the result of the completion time modeling of firmware described in Chapter 6.

Figure 7.1 shows a modified process flow diagram of the product development process at the sponsor company previously shown in Figure 2.4, that captures many of these recommendations (identified by bold italics).

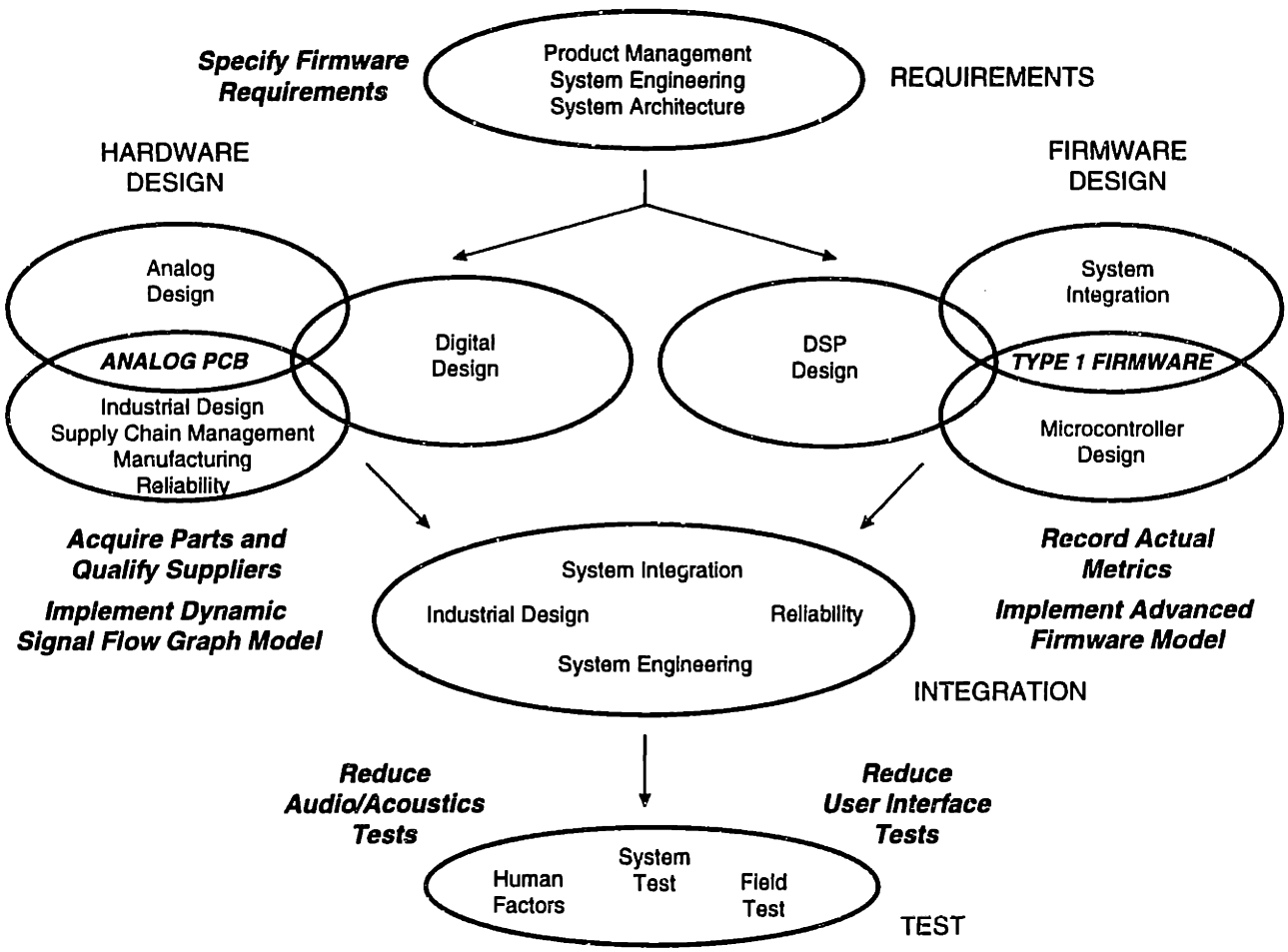


Figure 7.1: Modified Process Flow Diagram of the Product Development Process

7.2 Key Learnings

As a result of this research, a number of key learnings were extracted. These learnings were as follows:

- *The Design Structure Matrix (DSM) was an extremely versatile framework to capture a product development process.* In this research, the DSM was used to successfully model the development process of a fairly complex product - a digital wireless telephone. This process was comprised of 114 design tasks, fifteen functional groups, and four process flow phases, using concurrent engineering in a rapid protoyping environment. This learning was the result the product development process modeling described in Chapter 3.

- *DSM Representations and Coupling Analysis were moderately useful tools in analyzing information flow and dependencies.* In this research, both tools were used to assess the effective use of strategic decoupling, concurrent engineering, and rapid prototyping. This learning was the result of the information flow and dependencies analysis described in Chapter 3.

- *Partitioning and the Work Transformation Matrix (WTM) Model were powerful tools in performing Controlling Features and Total Work Analysis.* In this research, both tools were used to generate recommendations related to the reorganization of tasks and the establishment of strategic design teams. This learning was the result of the controlling features and total work analysis described in Chapter 4.

- *The Dynamic Signal Flow Graph Model outperformed the Static Signal Flow Graph Model and the Adjusted Reward Markov Chain Model for modeling hardware completion time.* In this research, the dynamic signal flow graph model was able to predict the completion time of a past prototype within 4% of the actual completion time. It was believed that the success of the model was a direct result of its ability to capture task times that change with subsequent iterations. This learning was the result of the completion time modeling of hardware described in Chapter 5.

- *The Aggregate-Code Model input metrics (productivity, fault density, and mean time-to-repair faults) for firmware completion time were useful in assessing the performance of past projects, but the model itself only represented a framework to implement more sophisticated models.* In this research, the aggregate-code model served both of these purposes. This learning was the result of the completion time modeling of firmware described in Chapter 6.

7.3 Future Work

As a continuation of this research, there are a number of opportunities to extend the work that has been presented. These opportunities are as follows:

- *Chapter 2: Product Development Process Modeling* - To model the product development process at the sponsor company, we constructed a task-level DSM. A parametric-level DSM of a digital wireless telephone, or comparable product, would yield additional insight to the strengths and weaknesses of both approaches.

- *Chapter 3: Analyzing Information Flow and Dependencies* - To analyze information transfer at the sponsor company, we used coupling analysis. This was a simple, yet relatively new approach to using the DSM. It would benefit from additional use within an industrial environment to further validate (or invalidate) its effectiveness.

- *Chapter 4: Analyzing Controlling Features and Total Work* - To analyze design iteration at the sponsor company, we again leveraged the information captured by a task-level DSM. In this capacity, it was unable to model the technical subtleties of hardware design and firmware design sufficiently. It would be a useful exercise to implement a parametric-level DSM within a rapid prototyping environment, and evaluate whether an optimal ordering for the introduction of functions and features could be identified [37].

- *Chapter 5: Modeling Hardware Completion Time* - To predict the completion time of the analog PCB at the sponsor company, we used the signal flow graph in two configurations (a static model and a dynamic model) and the reward Markov chain in a single configuration (an adjusted model). Investigation of alternative configurations for all three models would be an excellent way to build upon our findings. Some examples include the following: a static signal flow graph model with transition probabilities that are subject to change, a dynamic signal flow graph model that allows for different first-pass, second-pass and third-pass task times, and an adjusted reward Markov chain model that provides more accurate sensitivities for forward transition probabilities.

- *Chapter 6: Modeling Firmware Completion Time* - To predict the completion of the microcontroller firmware at the sponsor company, we used the aggregate-code model. A logical extension to the model would be to incorporate a non-linear relationship between the number of faults and the repair time. However, a more useful way to leverage the framework that was established by the model would be to implement a similar, yet more sophisticated model, such as the Software Life Cycle Model [27].

7.4 Final Remarks

The ability to produce superior products is a source of competitive advantage. This is particularly true for firms competing in fast-paced, high-tech industries, such as the wireless industry. In this thesis, we examined some of the issues associated with producing a digital wireless telephone within this intensely competitive environment. The six major themes of this research were the following:

- Model the product development process.
- Analyze information flow and dependencies between design tasks, functional groups, and process flow phases.
- Analyze the controlling features of engineering design iteration and total work.
- Model hardware completion time of printed circuit boards.
- Model firmware completion time of large-scale projects.

In performing this work, we aimed to achieve two important goals. First, to present useful and insightful recommendations to the sponsor company. Second, to contribute to the body of knowledge upon which this research was grounded. In the former case, our intent was to provide the sponsor company with additional knowledge and tools to assist them in overcoming the challenges associated with developing new products. Hopefully, in the latter case, we have provided the reader with exactly the same deliverables.

BIBLIOGRAPHY

[1] Abell, Thomas E., "Evaluation and Improvement of the Prototyping Process in Electronic Product Development," Leaders for Manufacturing (LFM) Program Masters Thesis, MIT, June 1994.

[2] Albrecht, Allan J. and John E. Gaffney, Jr., "Software Function, Source Lines of Code, and Development Effort Predictions: A Software Science Validation," *Institute of Electrical and Electronic Engineers (IEEE) Transactions of Software Engineering*, Vol. SE-9, pp. 639-649, November 1983.

[3] Albrecht, Allan J., "Measuring Application Development Productivity," in *Tutorial: Programming Productivity: Issues for the Eighties*, Caper Jones, Institute of Electrical and Electronic Engineers (IEEE) Computer Society Press, Los Alamitos, CA, pp. 34-43, 1981.

[4] Andersson, Johan, Jochen Pohl, and Steven D. Eppinger, "A Design Process Modeling Approach Incorporating Nonlinear Elements", *Proceedings of 1998 Design Engineering Technical Conference (DETC)*, American Society of Mechanical Engineers (ASME) Design Theory and Methodology Conference, Atlanta, GA, September 1998.

[5] Bertsekas, Dimitri and Robert Gallager, *Data Networks: 2nd Edition*, Prentice-Hall, Upper Saddle River, NJ, 1992.

[6] Cathey, Jimmie J. and Syed A. Nasar, *Theory and Problems of Basic Electrical Engineering*, Schaum's Outline Series, McGraw-Hill, New York, NY, 1984.

[7] Cookson Electronics, *Electrovert World-Wide-Web Home Page*, Universal Resource Locator (URL) http://ns.electrovert.com/products.html, Cookson Electronics, 1997.

[8] Eppinger, Steven D., "A Planning Method for Large-Scale Engineering Systems," *International Conference on Engineering Design (ICED)*, August 19-21, 1997.

[9] Eppinger, Steven D., "Model-Based Approaches to Managing Concurrent Engineering," *Journal of Engineering Design*, Vol. 2, No. 4, 1991.

[10] 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.

[11] Eppinger, Steven D., Murthy V. Nukala, and Daniel E. Whitney, "Generalized Models of Design Iteration Using Signal Flow Graphs," Research in Engineering Design, Vol. 9, pp. 112-123, 1997. Also MIT Sloan School of Management Working Paper No. 3866, revised December 1996.

[12] Garg, Vijay K., Kenneth Smolik, and Joseph E. Wilkes, *Applications of CDMA in Wireless/Personal Communications*, Prentice-Hall, Upper Saddle River, NJ, 1997.

[13] Harte, Lawrence, Steve Prokup, and Richard Levine, *Cellular and PCS: The Big Picture*, McGraw-Hill, New York, NY, 1997.

[14] Jawad, Frederick A. and Eric Johnsen, "Essential Evaluation Etiquette," Imperial College World-Wide-Web Home Page, Universal Resource Locator (URL) http://outoften.doc.ic.ac.uk/~nd/surprise_95/journal/vol4/eaj2/report.html, Imperial College, London, United Kingdom, 1995.

[15] Leon, Steven J., *Linear Algebra With Applications*, Macmillan Publishing Company, New York, NY, 1980.

[16] Mathworks, Inc., *MATLAB Reference Guide*, The Mathworks, Inc., Natick, MA, 1992.

[17] Mathworks, Inc., *MATLAB User's Guide*, The Mathworks, Inc., Natick, MA, 1992.

[18] McCord, Kent R. and Steven D. Eppinger, "Managing the Integration Problem in Concurrent Engineering," MIT Sloan School of Management Working Paper No. 3594, August 1993.

[19] Microsoft Corporation, *Microsoft Excel User's Guide*, Microsoft Corporation, Seattle, WA, 1993.

[20] Microsoft Corporation, *Microsoft Excel Visual Basic User's Guide*, Microsoft Corporation, Seattle, WA, 1993.

[21] Modell, Martin E., *A Professional's Guide to Systems Analysis: 2nd Edition*, McGraw-Hill, New York, NY, 1996.

[22] Oppenheim, Alan V. and Ronald W. Schafer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

[23] Osborne, Sean M., "Product Development Cycle Time Characterization Through Modeling of Process Iteration," Leaders for Manufacturing (LFM) Program Masters Thesis, MIT, June 1993.

[24] Pérez-Arriaga, Ignacio J., George C. Verghese, F. Luis Pagola, José Luis Sancha, and Fred C. Schweppe, "Developments in Selective Model Analysis of Small-Signal Stability in Electric Power Systems," *Automatica*, Vol. 26, No. 2, pp. 215-231, 1990.

[25] Pimmler, Thomas U. and Steven D. Eppinger, "Integration Analysis of Product Decompositions," 6th International Conference on Design Theory and Methodology, American Society of Mechanical Engineers (ASME), 1994.

[26] Putnam, Lawrence H, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," *Institute of Electrical and Electronic Engineers (IEEE) Transactions of Software Engineering*, Vol. SE-4, No. 4, pp. 345-361, July 1978.

[27] Putnam, Lawrence H. and Ware Myers, *Measures for Excellence - Reliable Software on Time, Within Budget*, Prentice-Hall, Upper Saddle River, NJ, 1992.

[28] Rogers, James L. and Christina L. Bloebaum, "Ordering Design Tasks Based on Coupling Strengths," American Institute of Aeronautics and Astronautics (AIAA) Paper No. 94-4326, 1994. Also National Aeronautics and Space Administration (NASA) Technical Memorandum No. TM-109137.

[29] Rogers, James L. and Jean-Francois M. Bathelemy, "Enhancements to the Design Manager's Aid for Intelligent Decomposition (DeMAID)," American Institute of Aeronautics and Astronautics (AIAA) Paper No. 92-4809, 1992.

[30] Rogers, James L., "DeMAID/GA - An Enhanced Design Manager's Aid for Intelligent Decomposition," American Institute of Aeronautics and Astronautics (AIAA) Paper No. 96-4157, 1992.

[31] Rogers, James L., "DeMAID/GA User's Guide - Design Manager's Aid for Intelligent Decomposition (DeMAID) with a Genetic Algorithm (GA)," National Aeronautics and Space Administration (NASA) Technical Memorandum No. TM-110241, 1996.

[32] Rogers, James L., Collin M. McCulley, and Christina L. Bloebaum, "Integrating a Genetic Algorithm into a Knowledge-Based System for Ordering Complex Design Processes," National Aeronautics and Space Administration (NASA) Technical Memorandum No. TM-110247, 1996.

[33] Rogers, James L., Design Manager's Aid for Intelligent Decomposition (DeMAID) with a Genetic Algorithm (GA) Software, *National Aeronautics and Space Administration (NASA) World-Wide-Web Home Page*, Universal Resource Locator (URL) http://www.larc.nasa.gov/LSS/ABSTRACTS/LSS-1997-0005.html, NASA, 1997.

[34] Sequeira, Michele, W., "Use of the Design Structure Matrix in the Improvement of an Automobile Development Process," Leaders for Manufacturing (LFM) Program Masters Thesis, MIT, June 1991.

[35] Smith, Robert P. and Eppinger, Steven D., "Identifying Controlling Features of Engineering Design Iteration," *Management Science*, Vol. 43, No. 3, March 1997. Also MIT Sloan School of Management Working Paper No. 3348, revised October 1995.

[36] Smith, Robert P. and Steven D. Eppinger, "A Predictive Model of Sequential Iteration in Engineering Design," *Management Science*, 1995. Also MIT Sloan School of Management Working Paper No. 3160, revised August 1995.

[37] Smith, Robert P. and Steven D. Eppinger, "Deciding between Sequential and Parallel Tasks in Engineering Design," MIT Sloan School of Management Working Paper No. 3959, October 1995.

[38] 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.

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

[40] Tausworthe, Robert C., "The Work Breakdown Structure in Software Project Management," *The Journal of Systems and Software 1*, pp. 181-186, 1980.

[41] Thomas, George B., Jr. and Ross L. Finney, *Calculus and Analytical Geometry: 7th Edition*, Addison-Wesley Publishing Company, Reading, MA, 1988.

[42] Urlich, Karl T. and Steven D. Eppinger, *Product Design and Development*, McGraw-Hill, New York, NY, 1995.

[43] Zadeh, Lofti A., "Fuzzy Logic," *Computer*, pp. 83-93, April 1988.

APPENDIX A: COUPLING ANALYSIS ALGORITHMS

FILE: coup.m

```
% FILE: coup.m

% COUPLING ANALYSIS
% MATLAB Version 4.0
%
% Randal D. Pinkett, Massachusetts Institute of Technology
% May 1998

clc;
sprintf('Performing Coupling Analysis...')

% Coupling-Related Variables
n = 6;                          % Dimension of the NDSM
m = [ 2 1 3 ];                  % Size of Groups
n2 = size(m,2);                 % Dimension of the ADSM and GDSM

% First Task of Each Grouping
group = [ 1 (m(1)+m(2)) (m(2)+m(3)) (n+1) ];

% Generate Numerical Design Structure Matrix (NDSM)
NDSM1 = [ 0 0 0 0 0 0.3 ];
NDSM2 = [ 1.3 0 0 0 2.7 0 ];
NDSM3 = [ 0 0.2 0 0 0 0 ];
NDSM4 = [ 0 2.3 0 0 0 0 ];
NDSM5 = [ 0 0 0.5 1.1 0 0 ];
NDSM6 = [ 0 0 0 0 0.7 0 ];
NDSM = [ NDSM1 ; NDSM2 ; NDSM3 ; NDSM4 ; NDSM5 ; NDSM6 ];

% Calculate NDSM Sum
NDSM_SUM = sum(sum(NDSM));

% Calculate Individual Coupling Metrics and Percentages
for j = 1:n

      % Individual Coupling Metrics
      input_measure(j) = sum(NDSM(j, :));
      output_measure(j) = sum(NDSM(:, j));
      volume_measure(j) = input_measure(j) + output_measure(j);

      % Metric Percentages
      input_percent(j) = input_measure(j) / NDSM_SUM;
      output_percent(j) = output_measure(j) / NDSM_SUM;
      volume_percent(j) = volume_measure(j) / NDSM_SUM / 2;

end
```

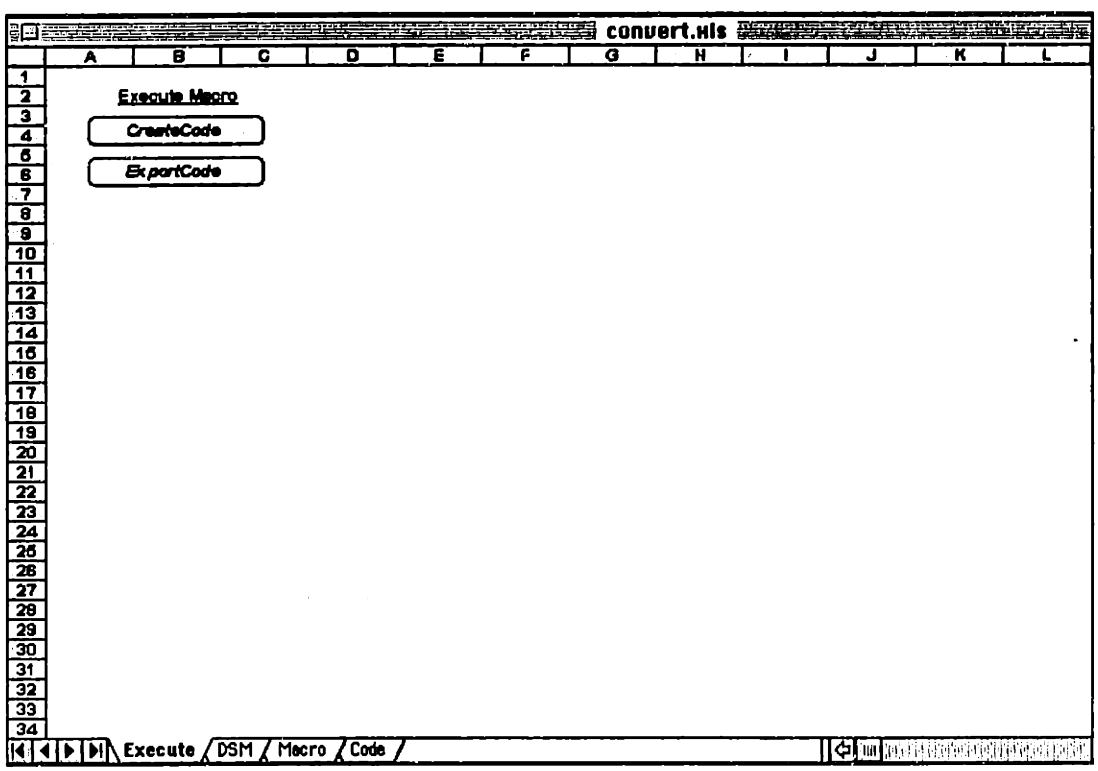
```
% FILE: coup.m (continued)

% Generate Aggregate Design Structure Matrix (ADSM)
ADSM = zeros(n2, n2);
k = 1;
for i = 1:n
      if i == group(k+1)
            k = k + 1;
      end
      l = 1;
      for j = 1:n
            if j == group(l+1)
                  l = l + 1;
            end
            ADSM(k, l) = ADSM(k, l) + NDSM(i, j);
      end
end

% Generate Group Design Structure Matrix (GDSM)
delta = max(max(NDSM));
for i = 1:n2
      for j = 1:n2
            GDSM(i, j) = ADSM(i, j) / (delta * m(i) * m(j));
      end
end

% Calculate Group Coupling Metrics and Relative Percentages
for k = 1:n2

      % Group Coupling Metrics
      group_input_measure(k) = sum(ADSM(k, :));
      group_output_measure(k) = sum(ADSM(:, k));
      group_volume_measure(k) = group_input_measure(k) +
                                group_output_measure(k) - ADSM(k, k);

      % Relative Metric Percentages
      pi = delta * n * m(k);
      po = delta * n * m(k);
      pv = delta * (2 * n * m(k) - (m(k) * m(k)));
      relative_input_percent(k) = group_input_measure(k) / pi;
      relative_output_percent(k) = group_output_measure(k) / po;
      relative_volume_percent(k) = group_volume_measure(k) / pv;

end

sprintf('Done.')
```

APPENDIX B: EXCEL/DeMAID CONVERSION ALGORITHMS

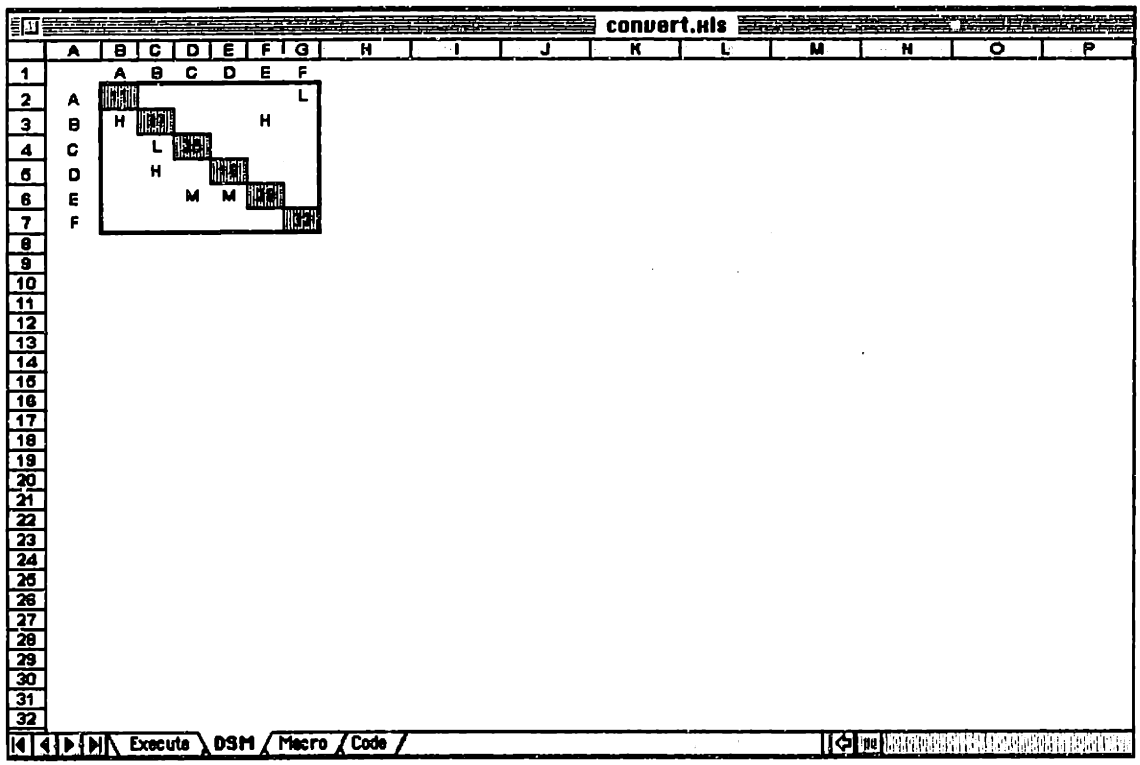
INSTRUCTIONS

1. Enter the DSM data in Excel (as shown below on the sheet "DSM").
2. Execute the Macro *CreateCode* in Excel (as shown below on the sheet "Execute").
3. Execute the Macro *ExportCode* in Excel (as shown below on the sheet "Execute") and the DeMAID/GA Code will be stored (as shown below on the sheet "Code").
4. Use the file **`dsm.npt`** as input to DeMAID/GA.

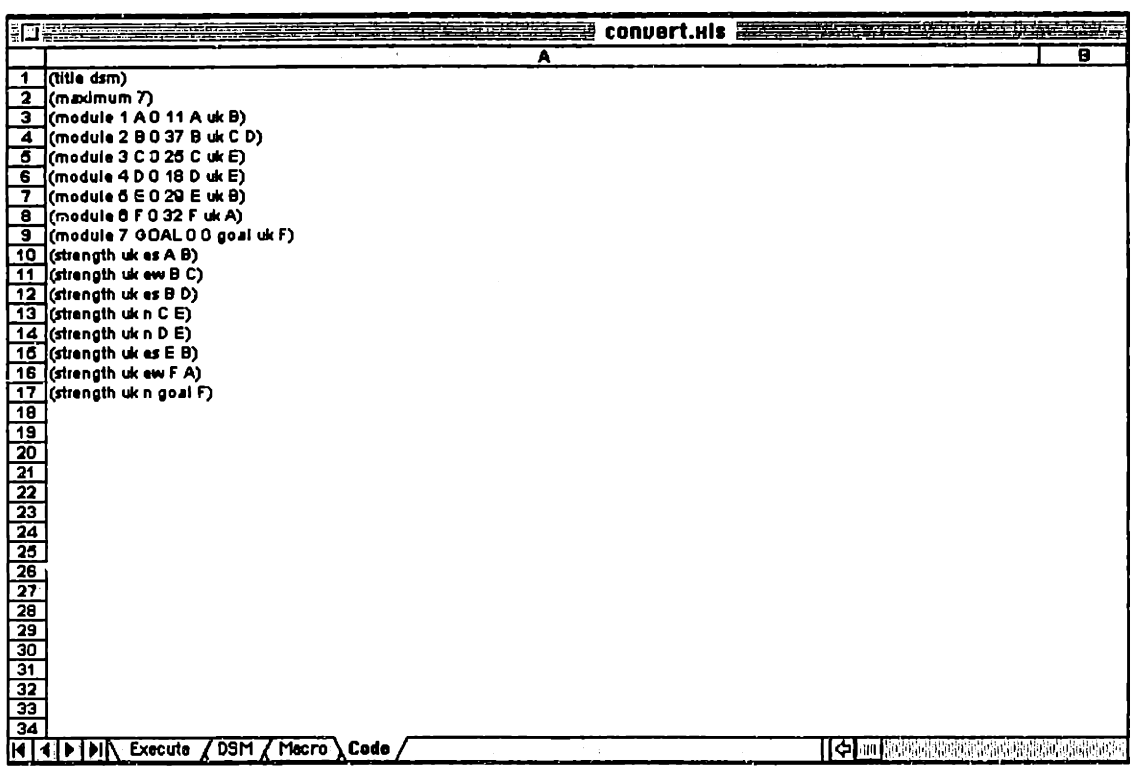
FILE: convert.xls • SHEET: "Execute"



FILE: convert.xls • SHEET: "DSM"



FILE: convert.xls • SHEET: "Code"



FILE: convert.xls • SHEET: "Macro"

```
'FILE: convert.xls - SHEET: "Macro"

'EXCEL/DeMAID CONVERSION
'Microsoft Excel Version 5.0 and DeMAID/GA Version 1.0
'Written in Visual Basic
'
'Randal D. Pinkett, Massachusetts Institute of Technology
'May 1998

'INITIALIZE GLOBAL VARIABLES

'DSM-Related Variables
Const Title = "dsm"                    'Title of the DSM (arbitrary name)
Const Dimension = 6                    'Dimension of the DSM (e.g. 10 = 10 x 10)

'Excel File-Related Variables
Const CodeFilename = "dsm.npt"         'Output file that will contain
                                       'the DeMAID/GA Code

'Excel Directory-Related Variables
Const FilePath = ""                    'Path where this file is located
Const CodePath = ""                    'Output path for the DeMAID/GA Code

'Excel Sheet-Related Variables
Const ExecuteSheet = "Execute"         'Sheet to execute the macros
Const DSMSheet = "DSM"                 'Sheet that contains the DSM
Const CodeSheet = "Code"               'Sheet that will contain the
                                       'DeMAID/GA Code

'DSM and DeMAID/GA Strength Mappings
Const H = "H"                          'DSM Symbol for High
Const M = "M"                          'DSM Symbol for Medium
Const L = "L"                          'DSM Symbol for Low
Const High = "es"                      'DeMAID/GA Mapping for High
Const Medium = "n"                     'DeMAID/GA Mapping for Medium
Const Low = "ew"                       'DeMAID/GA Mapping for Low

'CREATE DEMAID/GA CODE
Sub CreateCode()
    SetUpCodeSheet
    GenerateCode
End Sub

'EXPORT DeMAID/GA CODE
Sub ExportCode()
    ExportFile CodeSheet, CodePath, CodeFilename, xlTextMac
    MsgBox "DeMAID/GA Code Successfully Exported to File '" & _
      CodePath & CodeFilename & "'"
End Sub
```

```
'FILE: convert.xls - SHEET: "Macro" (continued)

'Clear Contents of Spreadsheet
Sub ClearContents(Sheet)
    'Clear Spreadsheet Contents
    OriginalSheetName = ActiveSheet.Name
    Sheets(Sheet).Select
    Cells.Select
    Selection.ClearContents
    Range("A1").Select
    Sheets(OriginalSheetName).Select
End Sub

'Setup DeMAID/GA Code Sheet
Sub SetUpCodeSheet()

    'Format Spreadsheet
    ClearContents CodeSheet
    With Worksheets(CodeSheet)
        .Range("A1").ColumnWidth = 100
        .Range("A1").Formula = "(title " & Title & ")"
        .Range("A2").Formula = "(maximum " & (Dimension + 1) & ")"
        .TextBoxes.Font.Name = "Arial"
        .TextBoxes.Font.Size = 10
    End With

End Sub

'Generate DeMAID/GA Code
Sub GenerateCode()

    'Initialize Variables
    ReDim Goal(Dimension) As Boolean      'Tracks modules to determine
                                          'DeMAID/GA "goal" module
    Dim NoInput As Boolean                'Tracks inputs to determine
                                          'DeMAID/GA "no-input" condition
    Number = 1    'Module Number
    CodeLine = 3  'Cell Number for DeMAID/GA Code (1 = title, 2 = maximum)

    'Evaluate Each Column to Generate Input List
    For Column = 2 To Dimension + 1

        'Evaluate Each Cell Within Each Column to Generate Input List
        Inputs = "uk"
        For Row = 2 To Dimension + 1
            Cell = Worksheets(DSMSheet).Cells(Row, Column).Value
            If (Cell = H) Or (Cell = M) Or (Cell = L) Then
                Inputs = Inputs & " " & _
                         Worksheets(DSMSheet).Cells(Row, 1).Value
                Goal(Row - 1) = True
            End If
        Next Row
```

```
'FILE: convert.xls - SHEET: "Macro" (continued)

        'Generate Input List When There Is No-Input
        If Inputs = "uk" Then
            Inputs = Inputs & " no-input"
        End If

        'Generate Input List When There Are Input(s)
        ModuleName = Worksheets(DSMSheet).Cells(1, Column).Value
        ModuleTime = " 0 " & _
            Int(Worksheets(DSMSheet).Cells(Number + 1, Number + 1).Value)
        Output = " " & ModuleName & " "
        Inputs = Inputs & ")"
        'Generate DeMAID/GA Module Code
        Code = "(module " & Number & " " & ModuleName & ModuleTime & _
                  Output & Inputs
        Worksheets(CodeSheet).Cells(CodeLine, 1).Value = Code
        Number = Number + 1
        CodeLine = CodeLine + 1
    Next Column

    'Evaluate Each Column to Generate Input List
    Inputs = "uk"
    For Column = 1 To Dimension
        If Goal(Column) = False Then
            Inputs = Inputs & " " & Worksheets(DSMSheet).Cells(1, _
                  Column + 1).Value
        End If
        Goal(Column) = False
    Next Column

    'Generate DeMAID/GA Module "goal" Code
    If Inputs <> "uk" Then
        Code = "(module " & Number & " GOAL 0 0 goal " & Inputs & ")"
        Worksheets(CodeSheet).Cells(CodeLine, 1).Value = Code
        CodeLine = CodeLine + 1
    End If

    'Evaluate Each Cell to Generate Strengths
    For Column = 2 To Dimension + 1
        NoInput = True
        For Row = 2 To Dimension + 1
            Strength = ""

            'Evaluate Strength and Assign Mapping
            Cell = Worksheets(DSMSheet).Cells(Row, Column).Value
            If Cell = H Then
                Strength = High
            ElseIf Cell = M Then
                Strength = Medium
            ElseIf Cell = L Then
                Strength = Low
            End If
```

```
'FILE: convert.xls - SHEET: "Macro" (continued)

        'Generate DeMAID/GA Strength Code
            If Strength <> "" Then
                NoInput = False
                Goal(Row - 1) = True
                Output = Worksheets(DSMSheet).Cells(1, Column).Value
                Inputs = Worksheets(DSMSheet).Cells(Row, 1).Value
                Code = "(strength uk " & Strength & " " & Output & _
                        " " & Inputs & ")"
                Worksheets(CodeSheet).Cells(CodeLine, 1).Value = Code
                CodeLine = CodeLine + 1
            End If
        Next Row

        'Generate DeMAID/GA Strength Code For No-Input
        If NoInput = True Then
            Output = Worksheets(DSMSheet).Cells(1, Column).Value
            Code = "(strength uk s " & Output & " no-input)"
            Worksheets(CodeSheet).Cells(CodeLine, 1).Value = Code
            CodeLine = CodeLine + 1
        End If
    Next Column

    'Generate DeMAID/GA Strength "goal" Code
    IsGoal = False
    For Row = 1 To Dimension
        If Goal(Row) = False Then
            Inputs = Worksheets(DSMSheet).Cells(Row + 1, 1).Value
            Code = "(strength uk n goal " & Inputs & ")"
            Worksheets(CodeSheet).Cells(CodeLine, 1).Value = Code
            CodeLine = CodeLine + 1
            IsGoal = True
        End If
    Next Row

    If IsGoal = False Then Worksheets(CodeSheet).Range("A2").Formula = _
      "(maximum " & Dimension & ")"

End Sub

'Export a File
Sub ExportFile(Sheet, ExportPath, ExportFilename, Fileformat)
    Filename = ExportPath & ExportFilename
    ActiveWorkbookName = ActiveWorkbook.Name
    ActiveSheetName = ActiveSheet.Name
    Sheets(Sheet).Select
    OriginalSheetName = ActiveSheet.Name
    ActiveWorkbook.SaveAs Filename:=Filename, _
            Fileformat:=Fileformat, CreateBackup:=False
    ActiveSheet.Name = OriginalSheetName
    Sheets(ActiveSheetName).Select
    ActiveFileName = FilePath & ActiveWorkbookName
    ActiveWorkbook.SaveAs Filename:=ActiveFileName, Fileformat:= _
            xlNormal, Password:="", WriteResPassword:="", _
            ReadOnlyRecommended:=False, CreateBackup:=False
End Sub
```

APPENDIX C: EIGENSTRUCTURE ANALYSIS ALGORITHMS

FILE: eigen_ex.m

```
% FILE: eigen_ex.m

% EIGENSTRUCTURE ANALYSIS
% MATLAB Version 4.0
%
% Randal D. Pinkett, Massachusetts Institute of Technology
% May 1998

clc;
sprintf('Performing Eigenstructure Analysis...')

% Eigenstructure-Related Variables
n_modes = 1;                    % Number of Design Modes to be Ranked
n_tasks = 4;                    % Number of Design Tasks to be Ranked
n_work = 4;                     % Number of Total Work Vector Tasks
                                % to be Ranked

% File-Related Variables
eigout = 'eigen.out';           % Filename For Eigenstructure Output Data
                                % (Tabular Format)
eigcsv = 'eigen.csv';           % Filename For Eigenstructure Output Data
                                % (CSV Format)

% Generate the Work Transformation Matrix (WTM)
WTM1 = [ 29 0.25 0.25 0 ];
WTM2 = [ 0 25 0 0.05 ];
WTM3 = [ 0 0 18 0.50 ];
WTM4 = [ 0.50 0 0 37 ];
WTM = [ WTM1 ; WTM2 ; WTM3 ; WTM4 ];

% Generate the Matrices A and W
W = diag(diag(WTM));
A = WTM - W;

% Perform Eigenstructure Analysis
[S, L, P, U, T] = eigen(A, W);

% Output Design Modes, Design Tasks, and Total Work Vector to Files (Ranked)
eigen_out(L, P, U, n_modes, n_tasks, n_work, eigout, eigcsv);

sprintf('Done.')
```

FUNCTION: eigen.m

```
% FUNCTION: [S, L, P, U, T] = eigen(A, W)

% Perform Eigenstructure Analysis

% S - Eigenvector Matrix
% L (lambda) - Eigenvalue Matrx
% P - Participation Matrix
% U - Total Work Vector
% T - Total Time Vector
% A - Coupling Strength Matrix from WTM
% W - Task Time Matrix from WTM

function [S, L, P, U, T] = eigen(A, W)

% Dimension
dim = size(A, 1);

% Identity Matrix and Initial Work Vector
I = diag(ones(dim, 1),0);
u0 = ones(dim, 1);

% Perform Eigenstructure Decomposition
[S, L] = eig(A);
eigenvalues = diag(L);
S = (-1) .* S;
INVS = inv(S);

% Calculate Contribution Matrix, Total Weight Vector, and Total Work Vector
contribution = inv(I - L);
weight = contribution * INVS * u0;
U = real(S * weight);

% Calculate Participation of Tasks to the Total Work
P = [];
for i = 1:dim
      eigenvector = (S(1:dim, i)');
      factor = ( 1 / ( 1 - eigenvalues (i) ));
      cum_sum = 0;
      for j = 1:dim
            cum_sum = cum_sum + INVS(i,j);
      end

      % Calculate Paritication Factors
      factors = (eigenvector * factor * cum_sum)';

      % Generate Participation Matrix
      P = [ P factors ];
end

% Calculate the Total Time Vector
T = W * U;
```

FUNCTION: eigen_out.m

```
% FUNCTION: eigen_out(L, P, U, n_modes, n_tasks, n_work, fileout, filecsv)

% Output Design Modes (ranked) and Total Work Vector (ranked) to Files

% L (lambda) - Eigenvalue Matrx
% P - Participation Matrix
% U - Total Work Vector
% n_modes - Number of Design Modes to be Ranked
% n_tasks - Number of Design Tasks to be Ranked
% fileout - Output Datafile in Tabular Format
% filecsv - Output Datafile in Comma Separated Version (CSV) Format

function eigen_out(L, P, U, n_modes, n_tasks, n_work, fileout, filecsv)

% Dimension
dim = size(P, 1);

% Open Output Data Files
fid1 = fopen(fileout, 'w');
fid2 = fopen(filecsv, 'w');

% Format File Header
fprintf(fid1, '\n');
for i = 1:n_modes
      fprintf(fid1, 'Design Mode #%1.0f\t', i);
end
fprintf(fid1, '\n');

for i = 1:n_modes
      fprintf(fid1, '--------------\t');
end
fprintf(fid1, '\n');

% Format File Sub-Header
for i = 1:n_modes
      fprintf(fid1, 'Task\tValue\t');
end
fprintf(fid1, '\n');

for i = 1:n_modes
      fprintf(fid1, '----\t-----\t');
end
fprintf(fid1, '\n');

% Calculate the Ranking Factors of the Design Modes
design_modes = real([ P U ]);
ranking_factor = ones(dim,1) ./ (1 - real(diag(L)));
```

```
% FUNCTION: eigen_out (continued)

% Rank the Design Modes (Total Work Vector is considered the last design mode)
previous = 0;
for i = 1:n_modes
      [current, mode(i)] = max(ranking_factor);
      ranking_factor(mode(i)) = 0;

      % Avoid Duplication of Conjugate Modes
      if previous == current
            [current, mode(i)] = max(ranking_factor);
            ranking_factor(mode(i)) = 0;
      end

      previous = current;
end
mode(n_modes+1) = dim+1;

% Rank the Design Task and Design Mode Contributions to the Total Work Vector
for i = 1:n_tasks
      for j = 1:n_modes
            [ data(i,(2*j)), k ] = max(design_modes(1:dim, mode(j)));
            data(i, (2*j-1)) = k;

            % Calculate Normalization Factor
            if i == 1
                  factor(j) = design_modes(k, mode(j));
            end
            design_modes(k, mode(j)) = -Inf;
            data1 = data(i, (2*j-1));
            data2 = data(i, (2*j));

            % Write Data to Files
            fprintf(fid1, '%1.0f\t\t%5.4f\t\t', data1, data2);
            fprintf(fid2, '%1.0f,%5.4f,', data1, data2);
      end
      fprintf(fid1, '\n');
      fprintf(fid2, '\n');
end

fprintf(fid1, '\nTotal Work Vector\n');
fprintf(fid1, '-----------------\n');
fprintf(fid1, 'Task\tValue\n');
fprintf(fid1, '----\t-----\n');
```

```
% FUNCTION: eigen_out (continued)

% Rank the Total Work Vector
for i = 1:n_work
      j = n_modes+1;
      [ data(i,(2*j)), k ] = max(design_modes(1:dim, mode(j)));
      data(i, (2*j-1)) = k;
      design_modes(k, mode(j)) = -Inf;
      data1 = data(i, (2*j-1));
      data2 = data(i, (2*j));

      % Write Data to Files
      fprintf(fid1, '%1.0f\t%5.4f\n', data1, data2);
      fprintf(fid2, '%1.0f,%5.4f\n', data1, data2);
end

fprintf(fid1, '\n');

% Close Output Data Files
fclose(fid1);
fclose(fid2);
```

FILE: eigen.out

```
Design Mode #1
--------------
Task  Value
----  -----
3     2.5569
4     2.0950
1     1.7164
2     0.2557

Total Work Vector
-----------------
Task  Value
----  -----
3     1.9396
4     1.8792
1     1.7584
2     1.0940
```

APPENDIX D: SIGNAL FLOW GRAPH ANALYSIS ALGORITHMS

FILE: sfg.m

```
% FILE: sfg.m

% SIGNAL FLOW GRAPH ANALYSIS
% MATLAB Version 4.0
%
% Randal D. Pinkett, Massachusetts Institute of Technology
% May 1998
%

clc;
sprintf('Performing Signal Flow Graph Analysis...')

% Signal Flow Graph-Related Variables
terms = 8;                      % Number of Terms for
                                % Probability Distribution Function (PDF) and
                                % Cumulative Distribution Function (CDF)
dp = 0.01;                      % Sensitivity Delta for Transition Probabilities
dt = 1;                         % Sensitivity Delta for Task Time s
                                % (must be an integer)
n_probs = 4;                    % Number of Transition Probabilities
n_tasks = 3;                    % Number of Task Times

% Data
data = [ 0.3 0.1 0.7 0.9 10 5 7 ];

% Raw Transition Probabilities and Task Times
p = data(1:n_probs);
t = data(n_probs+1:n_probs+n_tasks);

% Raw Numerator and Denominator
[numerator, denominator] = sfg_ex(0, p, t, n_probs, n_tasks);

% Calculate Probability Distribution
[distribution, expected_value] = dist(terms, numerator, denominator);

% Calculate Expected Value for Sensitivity Analysis
[expected_value, standard_deviation] = prob(numerator, denominator);

% Calculate Sensitvity Analysis
for i = 1:(n_probs+n_tasks)

% Raw Transition Probabilities and Task Times
p = data(1:n_probs);
t = data(n_probs+1:n_probs+n_tasks);
```

```
% FILE: sfg.m (continued)

% Sensitivity Transition Probabilities and Task Times
if i <= n_probs
      p(i) = p(i) * (1 + dp);
      delta = dp;
elseif i <= (n_probs+n_tasks)
      t(i-n_probs) = t(i-n_probs) * (1 + dt);
      delta = dt;
end

% Sensitiviy Numerator and Denominator
[numerator, denominator] = sfg_ex(i, p, t, n_probs, n_tasks);

% Calculate Mean and Standard Deviation
[mean, stddev] = prob(numerator, denominator);

% Peform Sensitivity Analysis
sensitivity(i) = ((mean - expected_value) / expected_value) / delta;
end

% Store Sensitivity Analysis
p_sens = sensitivity(1:n_probs);
t_sens = sensitivity(n_probs+1:n_probs+n_tasks);

% Write Probability Distribution Function
probability_sum = 0;
maximum_probability = max(distribution(1:terms, 2));
dummy_vector = [];
flag = 0;
for i = 1:terms
      time = distribution(i, 1);
      cumulative(i, 1) = time;
      probability = distribution(i, 2);
      probability_sum = probability_sum + probability;
      cumulative(i, 2) = probability_sum;

      if time < expected_value | flag == 1
            dummy_value = 0;
      else
            dummy_value = maximum_probability + 0.05;
            loc = i + 1;
            flag = 1;
      end
      dummy_vector = [ dummy_vector dummy_value ];
end

% Plot Probability Distribution Function
clg;
subplot(2,1,1), plot(distribution(1:terms,1), distribution(1:terms,2), 'y');
title('Probability Distribution Function');
xlabel('Time (Days)');
ylabel('Probability');
```

```
% FILE: sfg.m (continued)

% Plot Expected Value and Standard Deviation
xloc = distribution(loc + 1, 1);
yloc = maximum_probability + 0.05;
text1 = sprintf('Expected Value = %4.1f', expected_value);
text2 = sprintf('Standard Deviation = %4.1f', standard_deviation);
text(xloc, yloc, text1);
text(xloc, yloc - 0.10 * yloc, text2);
axis([ distribution(1,1) distribution(terms,1) 0 maximum_probability + 0.10
]);

% Plot Cumulative Distribution Function
subplot(2,1,2), plot(cumulative(1:terms,1), cumulative(1:terms,2), 'y');
title('Cumulatve Distribution Function');
xlabel('Time (Days)');
ylabel('Probability');
axis([ cumulative(1,1) cumulative(terms,1) 0 cumulative(terms, 2) + 0.10 ]);

done = sprintf('Probability sensitivity analysis stored in p_sens,\n');
done = sprintf('%sTask time sensitivity analysis stored in t_sens,', done);
done = sprintf('%s\nDone.', done)
```

FUNCTION: sfg_ex.m

```
% FUNCTION: [numerator, denominator] = sfg_ex(i, p, t, tp, n_probs, n_tasks)

% Generate Numerator and Denominator of Transfer Function H(z) for
% Signal Flow Graph Model

% numerator - Numerator
% denominator - Denominator
% i - Index i=0 for Normal Analysis
%           i>0 for Sensitivity Analysis
% p - Probabilities
% t - Task Times
% n_probs - Number of Probabilities
% n_tasks - Number of Tasks

function [numerator, denominator] = sfg_ex(i, p, t, n_probs, n_tasks)

% Transition Probability Adjustments
if i == 1
      p(3) = 1 - p(1);
elseif i == 2
      p(4) = 1 - p(2);
elseif i == 3
      p(1) = 1 - p(3);
elseif i == 4
      p(2) = 1 - p(4);
end

% Generate Numerator of Transfer Function H(z)
n1 = [ zeros(1, t(1)) 1 ];
n2 = [ zeros(1, t(2)) 1 ];
n3 = [ zeros(1, t(3)) p(3) ];
n4 = [ p(4) ];
num = conv(conv(conv(n1, n2), n3), n4);

% Generate Denominator of Transfer Function H(z)
d1 = [ 1 ];
d2 = [ zeros(1, (t(1)+t(2))) p(1) ];
d3 = [ zeros(1, (t(2)+t(3))) (p(3)*p(2)) ];
denom = sub(sub(d1, d2), d3);

% Polynomial Size Adjustment (removes unnecessary trailing zeros)
[numerator, denominator] = sfg_adj(num, denom);
```

FUNCTION: sfg_adj.m

```
% FUNCTION: [x_norm, y_norm] = sfg_adj(x, y)

% Polynomial Size Adjustment (removes unnecessary trailing zeros)

% x_norm - Adjusted Polynomial 1
% y_norm - Adjusted Polynomial 2
% x - Polynomial 1
% y - Polynomial 2

function [x_adjust, y_adjust] = norm(x, y)

size1 = size(x, 2);
size2 = size(y, 2);

if size1 > size2
      y = [ y zeros(1, size1-size2) ];
elseif size2 > size1
      x = [ x zeros(1, size2-size1) ];
end

x_adjust = x;
y_adjust = y;
```

FUNCTION: add.m

```
% FUNCTION: addition = add(x, y)

% Polynomial Addition

% addition - Polynomial Addition of x and y
% x - Polynomial 1
% y - Polynomial 2

function addition = add(x, y)

size1 = size(x, 2);
size2 = size(y, 2);

if size1 > size2
      y = [ y zeros(1, size1-size2) ];
elseif size2 > size1
      x = [ x zeros(1, size2-size1) ];
end

addition = x + y;
```

FUNCTION: sub.m

```
% FUNCTION: subtraction = sub(x, y)

% Polynomial Subtraction

% subtraction - Polynomial Subtraction of x and y
% x - Polynomial 1
% y - Polynomial 2

function subtraction = sub(x, y)

size1 = size(x, 2);
size2 = size(y, 2);

if size1 > size2
      y = [ y zeros(1, size1-size2) ];
elseif size2 > size1
      x = [ x zeros(1, size2-size1) ];
end

subtraction = x - y;
```

FUNCTION: dist.m

```
% FUNCTION: [distribution, expected] = dist(terms, numerator, denominator)

% Calculate Probability Distribution and Expected Value

% distribution - Probability Distribution
%                (Column 1 = Probability, Column 2 = Time/Days)
% terms - Number of Terms for Probability Distribution Function (PDF)
%         and Cumulative Distribution Function (CDF)
% expected - Expected Value
% numerator - Numerator of Transfer Function H(z)
% denominator - Denominator of Transfer Function H(z)

function [distribution, expected] = dist(terms, numerator, denominator)

distribution = [];

% Numerator and Denominator Powers (Order)
n_power = size(numerator, 2);
d_power = size(denominator, 2);

for i = 1:terms

      % Locate First Numerator Point
      for j = 1:n_power
            flag = 0;
            if numerator(j) ~= 0
                  flag = 1;
                  break;
            end
      end
      if flag == 0
            break;
      end
      nloc = j;
      for j = 1:d_power
            if denominator(j) ~= 0
                  break;
            end
      end
      dloc = j;

      % Quotient from Division (q)
      q_order = (nloc - 1) - (dloc - 1);
      quotient = numerator(nloc) / denominator(dloc);
      distribution(i, 1:2) = [ q_order, quotient ];

      % Numerator Order > Denominator Order
      if q_order > 0

      % Difference From Subtraction (s)
      s_order = q_order;
      subtract = [ zeros(s_order, 1)', denominator * quotient ];
```

```
% FUNCTION: dist.m (continued)

      % Numerator
      n_order = (s_order + d_power) - n_power;
      numerator = [ numerator, zeros(n_order, 1)' ] - subtract;

      % Demoninator
      d_order = s_order;
      denominator = [ denominator, zeros(d_order, 1)' ];

      % Numerator and Denominator Powers (Order)
      n_power = n_power + n_order;
      d_power = d_power + d_order;

      % Numerator Order < Denominator Order
      else

      % Difference From Subtraction (s)
      s_order = q_order * (-1);
      subtract = denominator * quotient;
      subtract = [ subtract(s_order+1:d_power), zeros(s_order, 1)' ];

      % Numerator
      numerator = numerator - subtract;

      end

      % Polynomial Size Adjustment (removes unecessary trailing zeros)
      for j = n_power:-1:1
            flag = 0;
            if numerator(j) ~= 0
                  break;
            end
      end
      nloc = j;
      for j = d_power:-1:1
            if denominator(j) ~= 0
                  break;
            end
      end
      dloc = j;
      loc = max(nloc, dloc);
      numerator = numerator(1:loc);
      denominator = denominator(1:loc);
      n_power = loc;
      d_power = loc;

end

% Expected Value
expected = sum(distribution(1:terms, 1) .* distribution(1:terms, 2));
```

FUNCTION: prob.m

```
% FUNCTION: [mean, stddev] = prob(numerator, denominator)

% Calculate Mean and Standard Deviation

% mean - Mean
% stddev - Standard Deviation
% numerator - Numerator of Transfer Function H(z)
% denominator - Denominator of Transfer Function H(z)

function [mean, stddev] = prob(numerator, denominator)

% Calculate Mean

% Perform Derivative Using Quotient Rule
[ num1, denom1 ] = deriv(numerator, denominator);

% Mean
mean = sum(num1) / sum(denom1);

% Calculate Standard Deviation

% Multiply "z" Term To Expected Value
num1 = [ zeros(1,1)', num1 ];

% Perform Derivative Using Quotient Rule
[ num2, denom2 ] = deriv(num1, denom1);

% Calculate Variance
mean_square = sum(num2) / sum(denom2);
square_mean = mean^2;
variance = abs(mean_square - square_mean);

% Calculate Standard Deviation
stddev = sqrt(variance);
```

FUNCTION: deriv.m

```
% FUNCTION: [d_numerator, d_denominator] = deriv(numerator, denominator)

% Calculate Derivative Using the Quotient Rule

% d_numerator - Numerator of the Derivative
% d_denominator - Denominator of the Derivative
% numerator - Numerator
% denominator - Denominator

function [d_numerator, d_denominator] = deriv(numerator, denominator)

% Determine the Size of the Numerator and the Denominator
n_size = size(numerator, 2);
d_size = size(denominator, 2);

u = numerator;
v = denominator;

% Calculate Derivative of the Numerator
dudx = [];
for i = 0:(n_size-1)
      dudx = [ dudx, i * u(i+1) ];
end
dudx = dudx(2:size(dudx,2));

% Calculate Derivative of the Denominator
dvdx = [];
for i = 0:(d_size-1)
      dvdx = [ dvdx, i * v(i+1) ];
end
dvdx = dvdx(2:size(dvdx,2));

% Calculate Left and Right Terms of the Numerator
vdudx = conv(v, dudx);
udvdx = conv(u, dvdx);
vdudx_size = size(vdudx, 2);
udvdx_size = size(udvdx, 2);

% Size Adjustments
if vdudx_size > udvdx_size
      udvdx = [ udvdx, zeros(vdudx_size - udvdx_size, 1)' ];
elseif udvdx_size > vdudx_size
      vdudx = [ vdudx, zeros(udvdx_size - vdudx_size, 1)' ];
end

% Calculate Numerator and Denominator of the Derivative (Using Quotient Rule)
d_numerator = vdudx - udvdx;
d_denominator = conv(v, v);
```

APPENDIX E: REWARD MARKOV CHAIN ANALYSIS ALGORITHMS

FILE: rmc.m

```
% FILE: rmc.m

% REWARD MARKOV CHAIN ANALYSIS
% MATLAB Version 4.0
%
% Randal D. Pinkett, Massachusetts Institute of Technology
% May 1998
%

clc;
sprintf('Performing Reward Chain Analysis...')

% Reward Markov Chain-Related Variables
dp = 0.01;                      % Sensitivity Delta for Transition Probabilities
dt = 1;                         % Sensitivity Delta for Task Times
                                % (must be integer)
n_probs = 4;                    % Number of Transition Probabilities
n_tasks = 3;                    % Number of Task Times

% Data
data = [ 1.0 0.3 0.7 0.1 10 5 7 ];

% Raw Transition Probabilities and Task Times
p = data(1:n_probs);
t = data(n_probs+1:n_probs+n_tasks);

% Calculate Completion Time
M = rmc_ex(0, p, t, n_probs, n_tasks);
T = markov(M);

% Calculate Sensitvity Analysis
for i = 1:(n_probs+n_tasks)

% Raw Transition Probabilities and Task Times
p = data(1:n_probs);
t = data(n_probs+1:n_probs+n_tasks);

% Sensitivity Transition Probabilities and Task Times
if i <= n_probs
      p(i) = p(i) * (1 + dp);
      delta = dp;
elseif i <= (n_probs+n_tasks)
      t(i-n_probs) = t(i-n_probs) * (1 + dt);
      delta = dt;
end

% Sensitivity Completion Time
M = rmc_ex(i, p, t, n_probs, n_tasks);
TP = markov(M);
```

```
% FILE: rmc.m (continued)

% Peform Sensitivity Analysis
sensitivity(i) = ((TP - T) / T) / delta;
end

% Store Sensitivity Analysis
p_sens = sensitivity(1:n_probs);
t_sens = sensitivity(n_probs+1:n_probs+n_tasks);

clc;
done = sprintf('Completion Time T = %6.3f\n', T);
done = sprintf('%sTransition probability sensitivity analysis stored in
p_sens,\n', done);
done = sprintf('%sTask time sensitivity analysis stored in t_sens,', done);
done = sprintf('%s\nDone.', done)
```

FUNCTION: rmc_ex.m

```
% FUNCTION: M = rmc_ex(i, p, t, n_probs, n_tasks)

% Generate Matrix for Reward Markov Chain Model

% i - Index i=0 for Normal Analysis
            i>0 for Sensitivity Analysis
% p - Transition Probabilities
% t - Task Times
% n_probs - Number of Transition Probabilities
% n_tasks - Number of Task Times
% M - Reward Markov Chain Matrix

function M = rmc_ex(i, p, t, n_probs, n_tasks)

% Transition Probability Adjustments
if i == 2
      p(3) = 1 - p(2);
elseif i == 3
      p(2) = 1 - p(3);
end

% Generate Matrix M
M = [ t(1) p(2) 0 ; p(1) t(2) p(4); 0 p(3) t(3) ];
```

FUNCTION: markov.m

```
% FUNCTION: T = markov(M)

% Calculate Total Expected Time of Reward Markov Chain
% Using Efficient Length Computation Algorithm

% M - Reward Markov Chain Matrix
% T - Total Expected Time

function T = markov(M)

% Transform Matrix into Reward Markov Chain Form
dim = size(M, 1);
P = -(M');
for i = 1:dim
      P(i,i) = 1;
end

% Place Task Times in a Column Vector
b = diag(M);

% Diagonalization Using Gaussian Elimination
[L, U] = lu(P);
D = diag(diag(U), 0);

% Use Modified Back Substitution to Get Total Time in Each Stage
X = inv(D) * inv(L) * b;

% Add Elements to Get Total Expected Time
T = sum(X);
```