MANAGING MULTIPLE INTERDEPENDENCIES IN LARGE SCALE SOFTWARE DEVELOPMENT PROJECTS

by

NANCY A. STAUDENMAYER

B.A., Mathematics/Economics
Wellesley College
(1986)

M.A., Statistics
University of California, Berkeley
(1988)

Submitted to the Alfred P. Sloan School of Management
in Partial Fulfillment of
the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1997

Signature of Author _____
MIT Sloan School of Management
June 2, 1997

Certified by _____
Michael A. Cusumano
Professor of Management
Thesis Supervisor

Accepted by _____
Birger Wernerfelt
Chairman, Ph.D. Program, Sloan School of Management

MANAGING MULTIPLE INTERDEPENDENCIES IN
LARGE SCALE SOFTWARE DEVELOPMENT PROJECTS

by

NANCY A. STAUDENMAYER

Submitted to the Alfred P. Sloan School of Management
on June 2, 1997, in partial fulfillment of the requirements for the Degree of
Doctor of Philosophy

ABSTRACT

The concept of interdependency is central to our thinking about organizations. It appears in theories of organizational design (Galbraith, 1973; Nadler & Tushman, 1983), organizational process (Van de Ven & Poole, 1989), and product development (Wheelwright & Clark, 1992; Cusumano & Selby, 1995). It is implicit in classic metaphors of organizations as "systems" (Katz & Kahn, 1966) as well as more contemporary portrayals of organizations as "networks" (Rockart & Short, 1989) or "jazz ensembles" (Eisenhardt & Tabrizi, 1995). Yet despite this centrality, interdependence itself has been under-explored both conceptually and empirically (Weick, 1974; Malone & Crowston, 1994). In particular, the structure and underlying contingencies which drive interdependency remain unclear. For example, pioneering research by Thompson (1967) defined interdependency as a varying degree of contingency among tasks inherent in the nature of the technology; more recent research has focused on the role of resource allocation (Malone & Crowston, 1994) and product strategy (Cusumano & Selby, 1995) as possible sources of interdependence. These variations suggest that there may be several types or categories of relationships among tasks, yet a comprehensive framework has yet to be developed. Furthermore, prior research largely focuses on the organizing implications of single types of interdependency in isolation from one another whereas in reality managers of organizations face a "complex web" of multiple interdependencies that they need to manage simultaneously.

This dissertation addresses the above points within the context of large scale software development projects. It uses this phenomenon to inductively derive a taxonomy of different types of interdependency and then explores the consequences of multiple interdependencies for project strategy, structure, and performance. The results suggest that interdependencies among tasks on a given project are largely driven by four sets of forces: the product technology, external product environment, internal work unit environment, and organizational structure and resource policy in the firm. Teams develop different strategies for confronting this hierarchical system of evolving task relationships. The data further suggest that the choice of these strategies has important implications for project performance.

The dissertation research is structured as three papers, each of which explores a specific component of the interdependence-structure-performance relationship. The first paper, "Interdependency: Conceptual, Empirical, and Practical Issues," surveys the literature on interdependency across three distinct theoretical paradigms, information processing theories, resource-based theories, and theories of sense-making, and develops a conceptual framework that places these competing paradigms in perspective. It also documents how researchers have measured and evaluated interdependence and notes a decided uniformity in past empirical approaches to the topic.

The second paper, "An Empirically-Derived Taxonomy of Multiple Interdependencies," attempts to open up the "black box" of task relationships. Drawing upon a large sample of examples of interdependency, it develops, via affinity diagram techniques, a taxonomy of multiple interdependencies.

Given that model, the third paper, entitled "Webs of Interdependency: Strategies for Managing Multiple Interdependencies in New Product Development," examines the strategies teams use to manage multiple interdependencies and proposes implications for project performance, which are supported by internal and external project evaluation data. The analysis revealed three basic strategies: systemic manipulation, constrained systemic manipulation, and local reaction. In this study, the highest performing projects adopted a systemic manipulation approach in which they focused on the pattern of interdependencies as a design variable. These teams made critical up front design decisions in order to simplify task coordination, prioritized interdependencies, gained leverage off of central tasks, and were aware of the path dependent nature of their choices. Results suggested that interdependency management can be viewed as a strategic process variable insofar as interdependencies are manipulatable through strategy and organization design. This flexibility is constrained, however, due to the existence of certain technical and organizational legacy factors, which tend to embed tasks and task relationships in a prior history of structure.

The research design of this study is that of multiple case studies (Yin, 1984) of six large scale software development projects at two leading companies, Microsoft and Lucent Technologies (formerly AT&T). Large scale product development projects represent an ideal setting in which to examine this concept in that interdependencies are ubiquitous and critical to project success (Clark & Fujimoto, 1991; Cusumano & Selby, 1995). Utilizing a multi-method approach, the study draws upon a combination of qualitative and quantitative data including 71 interviews with project team members, on-site observation at both firms, and various forms of project documentation such as schedules, design documents, and memos. The outcome variable (project performance) was assessed via a brief questionnaire administered to experts and upper management in each firm. Because the study was designed to be an inductive investigation, the above hypotheses await future testing.

Thesis Committee: Michael A. Cusumano (Chair)
Professor of Management
Alfred P. Sloan School of Management, MIT

Deborah Ancona
Associate Professor of Management
Alfred P. Sloan School of Management, MIT

Steven Eppinger
Associate Professor of Management Science
Alfred P. Sloan School of Management, MIT

Michael A. Tushman
Professor of Management
Columbia University

TABLE OF CONTENTS

LIST OF FIGURES*

* Note that the digit to the left of the decimal point refers to the chapter or appendix. Digits and letters to the right of the decimal consecutively number figures within a chapter or appendix (e.g., Figure 2.2 is the second figure in the second chapter).

LIST OF TABLES*

* Note that the digit to the left of the decimal point refers to the chapter or appendix. Digits and letters to the right of the decimal consecutively number tables within a chapter or appendix (e.g., Table 2.2 is the second table in the second chapter).

ACKNOWLEDGEMENTS

> Writing a [dissertation] is an adventure. To begin with, it is a toy and an amusement; then it becomes a mistress, and then it becomes a master, and then a tyrant. The last phase is that just as you are about to be reconciled to your servitude, you kill the monster, and fling him out to the public.
>
> Winston Churchill

As I approach the final stages of the doctoral program, I realize that there are really two ways of evaluating a dissertation. One is as an output, readily observed in the actual written pages. But a dissertation also represents a process-- of personal growth, learning, experiences, and interaction with other people. I will always be somewhat dissatisfied with the dissertation itself insofar as it fails to completely capture all that I set out to achieve and say. What I ultimately find far more satisfying, and am most proud of, is how my ideas and understanding evolved and matured while producing this output. Here I acknowledge the importance of some of the key people who participated in that process.

First, I would like to thank the members of my committee, Deborah Ancona, Steve Eppinger, and Michael Tushman. Deborah is responsible for my growth as an organizational scholar. She encouraged me to make the work more theoretical, and her faith in my ability to do so was a source of inspiration. Deborah also has an amazing ability to identify what is most interesting about a piece of work and translate between phenomenon and theory. Steve lent both substantive methodological ideas and key comments that always cut to the essence of what I was trying to do. While Michael Tushman joined the committee late, his insights on the interaction between technology and organizations were invaluable. Michael also served as a wonderful model of how to successfully combine excellence in both scholarship and teaching. My committee truly epitomized "interdisciplinary research," and I think our discussions were richer and more interesting as a result of that unique collective viewpoint. I hope (and suspect) that they enjoyed the interaction as much as I did.

In addition to my formal committee, I had a group of "informal" advisors whose advice and insights were critical to my work. Tom Allen was a source of both financial and moral support since my start in the MIT program. Wanda Orlikowski taught me how to "think" like a theoretical scholar; her seminar on IT and Organizations stands out as one of the best classes I have ever taken. Marcie Tyre is a continual source of advice and friendship. Working together helped balance the inevitable loneliness of doing a thesis. Ellen Langer's research and classes at Harvard opened up the world of social psychology to me. Scott Stern provided invaluable advice not only about the dissertation but also about finding a job and starting a career.

The fellow doctoral students who helped shape my experience are too numerous to list here, so I will just mention a few who deserve special acknowledgment. First is my cohort of Maurizio Sobrero, Omar Toulan, and Steven White. Omar was the best office mate one could ask for and faithfully sent me postcards from all of his exotic destinations. Maurizio and Steven helped me navigate the MTI program. The last year was particularly lonely without them, but I look forward to many trips between North Carolina, Italy, and Hong Kong in the future. Annabelle Gawer shared my passion for clothes and magazines and made me laugh when I most needed it. George Wyner and Chris Voisey were good colleagues. Finally, I would like to thank Chris Tucci for his continual friendship. Chris and I had many long, long phone conversations as we both struggled to complete the final stages of our research, and I always felt better after talking with him.

Several people outside MIT also made important contributions to this work. Larry Votta and Dewayne Perry of Lucent Technologies provided financial support, company access, and friendship. They inspired my interest in software development and patiently answered many questions about software technology and process. Much of this dissertation reflects the depth of their knowledge and experience (although any errors are my own). Dave Moore and Chris Williams of Microsoft also very generously gave their time and support. Finally, I would like to thank Lisa Pearl for her friendship.

This research would not have been possible without the participation of subjects and managers at the two firms. The MIT International Center for Research on the Management of Technology (ICRMOT) and the MIT Lemelson Foundation also provided invaluable financial support.

Finally, I would like to thank my advisor, Michael Cusumano. Quite simply, asking Michael to be my advisor was the single best decision I made during the entire five years at MIT. He was a source of support and amazing opportunities. In particular, the chance to study Microsoft and help on his book will always stand out as one of the most interesting and exciting experiences of my life. Michael also taught me about strategy and influenced my thinking about the importance of integrating strategy and structure.

But more importantly than the opportunities and resources he provided (although I greatly appreciate them), I value and respect Michael for the type of person he is. His sense of integrity and high intellectual standards are a model more scholars should try to emulate. Perhaps less visible to those who don't know him well, Michael is an incredibly caring and thoughtful person. From the beginning, he stood by and supported me during this process-- even though I tested his patience on numerous occasions. He thus played the role of advisor to the fullest extent and in the process became my friend. Michael serves as my inspiration for how to be both a scholarly-thinker and a scholarly-person.

CHAPTER I: INTRODUCTION AND OVERVIEW

This chapter serves as an introduction and overview of the dissertation. It begins by describing the overall motivation for the work and provides an organizing framework which links the next three chapters together. It also highlights and explains three important aspects of the research agenda: the distinction between the concept and the phenomenon used to explore it, the choice of an inductive analytical approach, and the unit(s) of analysis. The chapter closes with a summary figure describing the research questions, design, and deliverables, which define each of the subsequent chapters.

1.1 Research Motivation

An organization can be defined as a set of interacting elements that acquires inputs from the environment, transforms them, and discharges outputs to the external environment. The need for inputs and outputs reflects dependency on the environment. Interacting elements mean that people and departments depend upon one another and must work together (Daft, 1983b). The concept of interdependency therefore lies at the very heart of what an organization is and why organizations exist. Understanding interdependency is also crucial to addressing a myriad of problems in firms, as evident in recent work on new forms of organizing (Baker, 1992), alliances (Roth, 1995), and process engineering and improvement (Adler, Mandelbaum, Nguyen, & Schwerer, March 1995).

Yet despite this centrality, interdependence itself has been under-explored both conceptually and empirically (Malone & Crowston, March 1994; Weick, March 1976). We lack a clear, integrated understanding of the structure and underlying contingencies which drive relationships among tasks. The lack of a comprehensive framework in turn constrains our knowledge of the interplay between interdependency and organizational structure, strategy, and performance.

For example, prior research largely focuses on the implications of single types of interdependency in isolation from one another whereas in reality managers of organizations face a "complex web" of multiple interdependencies that they need to manage simultaneously. How does a team define a coherent strategy that enables it to meet its goals of customer needs and schedule while simultaneously interacting with and depending on other teams? How do managers develop an organizational structure and strategy to simultaneously coordinate multiple tasks? In other words, how do we encourage both independence of action and initiative, ownership, and flexibility (at the team and individual levels) and coordination and integration (Lawrence & Lorsch, 1967)?

The paradox is particularly evident in large scale product development efforts. Although interdependency has long been a central issue in research on new product development (Allen, 1977; Cusumano & Selby, 1995; Wheelwright & Clark, 1992), certain trends in technology, markets, and organizations are raising the level of contingency in work processes and activities. For example, the very nature of many technological products is changing. More and more products combine hardware with software or, in the case of multi-media products, technical and artistic content, implying interdependency over ever more diverse and varied disciplines. Products are also becoming increasingly complex and componentized and, therefore, beyond the capabilities of one or even a few individuals to produce. Similarly, many companies are beginning to use product portfolios or platforms to compete and adapt in fast changing environments (Nobeoka & Cusumano, November 1995), raising complicated issues of cross team coordination. Finally, we are seeing experiments with new organizational forms such as cross-functional teams (Ancona, 1990) and networked organizations (Baker, 1992).

At the same time, technical and market environments on many of these large scale development efforts demand ever quicker response time and flexibility. As a result, we have seen a profusion of research and ideas about new ways of structuring (Clark, Chew, & Fujimoto, 1987; Cusumano, 1992; Cusumano & Nobeoka, 1992; Meyer & Lehnerd, 1997; Meyer & Utterback, Spring 1993; Nonaka, 1990), leading (Ancona & Caldwell, 1987; Clark & Fujimoto, 1989; Clark & Wheelwright, 1992; Eisenhardt, 1989), strategizing (Cusumano & Nobeoka, 1992; Cusumano & Selby, 1995; Nobeoka & Cusumano, November 1995), producing (Adler, et al., March 1995; Brown & Eisenhardt, March 1995; Smith & Cusumano, September 1993), and learning (Imai, Nonaka, & Takeuchi, 1985; Sitkin, 1996). This research study argues that before proposing such new solutions, we need to develop an understanding of the underlying interdependencies, one that reflects the more complicated structure of work in organizations today.

Informal observations and conversations with managers and engineers working on large scale product development projects support the initial conjecture that interdependency is both important and not well understood. For example, people frequently cite "managing interdependencies" as one of their biggest challenges (Cusumano & Selby, 1995; Perry, Staudenmayer, & Votta, July 1994; Sabbagh, 1995).

On the Boeing 777 development effort, engineers referred to "interferences" when two pieces overlap in space:

> You have five thousand engineers designing the airplane... It's very difficult for those engineers to coordinate ... and it's very costly. You end up with an airplane that's very difficult to build. The first time that parts come together ... they don't fit. (Sabbagh, p. 52)
>
> As we go through the process of designing, day in and day out, people continue to change their designs and they will interfere with other parts....

> You may have checked against somebody else's part the day before, and when you check against it today he's changed his part and it goes right through your part. ... And designers just have to deal with the fact that parts are changing, and they have to coordinate with each other as best they can. But at the end of each of these six stages we go through what we call a freeze. I say, 'OK, no more designs. Now go work out all the remaining fit problems.' So you get a few days where nobody is designing any more; all they're doing is comparing their designs with everybody else's design to make sure they have no more interferences. (Sabbagh, pp. 56-57)

> I saw one of our senior structures people going up and down the 10-18 building the other day looking for a hydraulic guy. He wanted to put a bracket on his floor beam, and they had not come to an agreement on where the bracket was going to go and ... whether it was going to create an interference. And he stopped me in the hall and he was so mad because he couldn't find the hydraulics engineer. And he said, 'What do they look like? Do they have tubes in their pockets? Do they have tubes coming out of their heads? What do those people look like?' (Sabbagh, p. 57)

Such development efforts are obviously highly precarious. Seemingly minor schedule or design changes in one part of the project can have major and catastrophic implications for the project as a whole. Ripple effects can also start to extend across projects in a firm as products share technical components and designs. Finally, evidence in a prior study of large scale software development (Perry, et al., July 1994) and supported by other studies (Curtis, Krasner, & Iscoe, November 1988; Perlow, 1995) suggests that product engineers spent the vast majority of their time coordinating and communicating with others, in other words, managing interdependencies.

1.2 Research Questions and Organizing Framework

The specific empirical questions addressed in this research are: (1) What types of interdependency are encountered in large scale software development efforts? (2) What are the implications of multiple interdependencies for project strategy and structure? That is, how do teams approach the management of multiple interdependencies? and (3) What impact does the combination of multiple interdependencies and solution

strategies have on project performance? Figure 1 presents a framework depicting the relationships among the key constructs which emerged from the analysis. This figure also maps parts of the framework to the chapters which address them.

1.3 What This Dissertation Is (and What It Is Not)

Several factors define this dissertation. First, note that the study examines both an empirical phenomenon (large scale software product development) and a theoretical concept (interdependency). It uses the former to increase our understanding of the latter; at the same time, integrating theory with practice should yield ideas for better management of large scale product development efforts. This research is also distinguished by its inductive approach. It seeks to generate new theory and ideas about interdependency and organization design. The results of the study are, therefore, hypotheses which await future testing. Because these characteristics, as well as the units of analysis, represent potential sources of confusion or criticism, they are highlighted below.

The Concept versus The Phenomenon

Scholars who study the research process often distinguish between the theoretical and empirical planes of analysis (Bailyn, 1977; Cook & Campbell, 1979). Concepts or theoretical ideas reside in a conceptual plane and data in an empirical one. This distinction is important because it enables us to differentiate between different types of research. "Theorizing" research concentrates within the conceptual plane. At the other extreme, empirical investigations involve manipulations that stay entirely (or largely) within the empirical domain (Bailyn, 1977). This research study endeavors to work both within and across these two planes of analysis. It explores a general and important theoretical concept, interdependency, while simultaneously seeking to improve the management of large scale product development efforts.

As noted by Bailyn, attempting to exploit the obvious connection between theory and practice improves value and validity (Bailyn, 1977). Yet maintaining these dual planes of analysis also presents certain complications in terms of conducting, framing, and ultimately writing up the research. Does one lead with the phenomenon or the more general concept, and what is the appropriate balance between the two? Some readers may be less familiar with (and less interested in) the details of large scale product development. How much background on the phenomenon is necessary? How does one make use of the complexity inherent in the real world phenomenon while simultaneously achieving the simplicity and parsimony that are characteristic of good theory? How does one make it clear what results apply to the phenomenon versus those that are more generalizable? Such dilemmas are partly resolved here by breaking the research apart into three chapters.

The Analytical Approach

This research is also distinguished by its inductive approach. It represents research as a cognitive as opposed to validation process in that it involved developing conceptual understanding as the study progressed via the continuous interplay between concepts and data (Bailyn, 1977; Van Maanen, 1997). The study was guided but not governed by existing theoretical paradigms and was designed to generate new theory about interdependency, to be exploratory and theory building, not to test existing theory or refute old models. This approach was not without its costs, however. In particular, it limits the generalizability of the results and involves exploring, documenting, and positing linkages among variables as opposed to testing hypotheses.

The Unit(s) of Analysis

Finally, as a point of clarification, this research was primarily conducted at two levels of analysis. The taxonomy in chapter 3 focuses on task interdependency as the unit of analysis. The subsequent analysis in chapter 4 compares strategy, structure, and performance across projects at two firms and adopts the project as the level of analysis. The rationale and appropriateness behind this switch in level are discussed in chapter 5.

1.4 Dissertation Structure

Finally, note that the structure of this dissertation departs from the typical format of ten or so (linear) chapters. This dissertation was actually written as three papers, each of which explores a specific part of the organizing framework shown in Figure 1. The first paper (chapter 2) surveys the literature on interdependency across three distinct paradigms and develops a conceptual framework which places these competing paradigms in perspective. The survey suggests that there are significant theoretical and empirical gaps and overlaps in our understanding of this key concept and that as a community of organizational scholars we have gotten fairly sloppy in our use of the term. Building on this survey, the second paper (chapter 3) uses examples of interdependencies in six case study projects to inductively derive a taxonomy of multiple interdependencies. Given that model, a third paper (chapter 4) proposes linkages between the strategies and structures teams use to manage multiple interdependencies and project performance. Table 1 summarizes the primary research questions, design, data, analytical method, and deliverables in each of the papers (chapters).

Thus, taken together, the papers mirror the evolution of this investigation of interdependency. Although there exists some overlap, repetition is minimized such that each paper can stand on its own. Ideally, however, the papers (chapters) should be

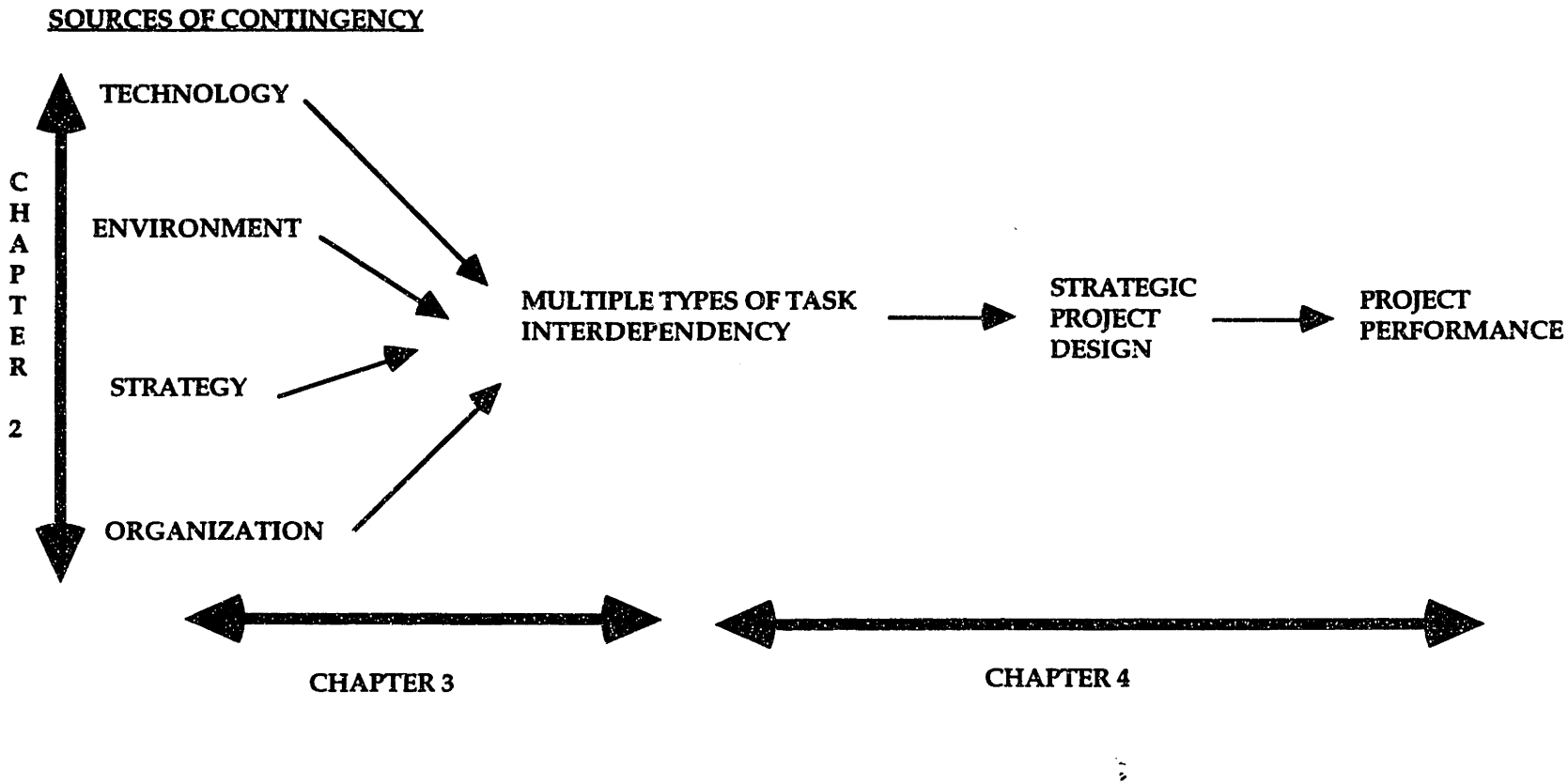
read together (in order) insofar as they tell a story about interdependency in organizations today.

TABLE 1
SUMMARY OF RESEARCH QUESTIONS & DELIVERABLES

	CHAPTER 2	CHAPTER 3	CHAPTER 4
RESEARCH QUESTION	What are the alternative ways scholars have conceptualized and empirically studied interdependency?	What types of interdependency exist in large scale software development efforts?	What are the implications of multiple kinds of interdependency for project strategy, structure and performance?
RESEARCH DELIVERABLES	An integrative model of interdependency across three theoretical paradigms; An agenda for interdependency research	A taxonomy of multiple kinds of interdependency	Identification of distinct strategies for managing multiple interdependencies and hypotheses about their implications for project performance
UNIT OF ANALYSIS	Not applicable	Interdependency, defined as a contingent relationship among tasks	Software development projects
DESIGN & DATA	Prior theoretical and empirical studies involving interdependency	A large sample of examples of interdependency derived from interviews	A comparative study of 6 case study projects at 2 firms; Interviews and project documentation
PRINCIPLE ANALYTICAL TECHNIQUE	Analytical framing and analysis	The Language Processing™ Method; KJ (Affinity) diagramming	Coding of interviews and project documents, analytical matrices and project profiling

FIGURE 1

ORGANIZING FRAMEWORK FOR THE DISSERTATION



CHAPTER II: INTERDEPENDENCY: CONCEPTUAL, EMPIRICAL, & PRACTICAL ISSUES

This chapter surveys the literature on interdependency, defined as a contingent relationship among tasks or activities. It identifies three distinct theoretical paradigms about interdependence (information processing theories, resource-based theories, and theories of sense-making) and develops a multi-dimensional conceptual framework that places these competing paradigms in perspective. It also documents how researchers have measured and evaluated interdependence. The chapter concludes with an overall assessment of the current state of interdependency theory and some suggestions for a new research agenda on this fundamental concept.

2.1 Introduction

Many scholars have identified interdependence as a critical variable for understanding organizations. Overall, this body of research has progressed from some early ideas to a vast spectrum of work by various authors. One unfortunate outcome of this rich history, however, is a confusing multitude of conceptualizations and operationalizations. Most organization theorists adopt Thompson's basic definition of interdependency as a contingent relationship among tasks or activities (Thompson, 1967). Yet March and Simon describe interdependence as a "felt need for joint decision-making" (March & Simon, 1958), while others depict it in terms of individuals (Granovetter, 1973), departments (Gresov & Stephens, 1993), and even firms (Roth, 1995). Mitchell and Silver examined the role of goal interdependence in team cooperation (Mitchell & Silver, 1990); Ancona and Caldwell focused on the importance of external environment interdependencies for successful product development (Ancona & Caldwell, 1990). Interdependence is both a process, something "that happens to X over time," but it can also be viewed as a variable (tasks can be more or less interdependent) (Selznick, 1957). Interdependence sometimes appears as a task dimension (Hrebiniak, 1974; Lynch, 1974; Van de Ven & Ferry, 1980), but other times it is a variable ("task interdependence") in its own right (Allen, November 1986; Tushman, 1976; Tushman, 1978). In fact, this variety is not that surprising given that

interdependence is a very difficult concept to define both theoretically and operationally (Pennings, September 1975). Furthermore, the very centrality of the concept implies that interdependence will appear in many different organizational models, thus accounting for some diversity of the definition.

But as scholars we are also left with a high degree of ambiguity and confounding with respect to the concept of interdependency. Often, a single label is used to refer to a broad spectrum of interaction (e.g., Thompson's definition of internal interdependence, which includes the "exchange of resources, information and products as well as contingencies of action") (Thompson, 1967). A variety of different terms also appear, even though they conceivably refer to conceptually and empirically similar concepts (e.g., component interdependency (Dellarocas, 1995), subsystem interdependence (Allen, November 1986), and technical interdependence (Eppinger, Whitney, Smith, & Gebala, 1994); coordination tasks (Hauptman, 1986) and interdepartment interdependence (Adler, March-April 1995)).

Even when authors use the same terminology, they often take very different perspectives on the concept, which reflect fundamentally different assumptions about the nature of technology, organizations, and people. For example, Thompson, March and Simon, and Weick all write about "task interdependence." But whereas Thompson views it as inherent in the technology or task (Thompson, 1967), March and Simon depict interdependence as a characteristic of the way people behave and make decisions while executing their work (March & Simon, 1958). Karl Weick rejects the very notion that task interdependencies are completely apprehensible, arguing that some relationships are too complex, ephemeral, or subtle to be completely understood via rational analysis (Weick, 1974; Weick, 1990; Weick & Roberts, 1993). Clearly, further use

of the interdependency concept would benefit from an attempt to reconcile and understand this diversity.

A primary purpose of this chapter, therefore, is to review the literature on interdependency in such a way as to place competing paradigms in perspective. The contribution here is a synthesis of theory, which reconciles existing conceptualizations and identifies some assumptions taken-for-granted as well as gaps in our understanding. Figure 2.1 outlines a metatheoretical schema for classifying the major schools of thought about interdependency. The schema is based on three analytical dimensions: (1) the primary driver of the interdependency, (2) the structure of the task relationship, and (3) the nature (content) of the tasks.

A second purpose is to suggest new questions and opportunities in the study of interdependency. The survey reveals that the information processing perspective originating with Thompson has heavily influenced our current knowledge of interdependence, a narrowness perpetrated by a decidedly limited range of empirical approaches to the topic. Furthermore, although much progress has been made in our knowledge of interdependency, this existing knowledge is currently fragmented, unreconciled, and relatively simplistic. It primarily focuses on one type of interdependency at a time, embracing single elements more than the conflicts or complements among those elements. Now, therefore, represents an opportune time for new kinds of studies of interdependence in that the real world of work and organizing is getting more complicated than our existing theories acknowledge. The second contribution of this chapter is thus a research agenda. In particular, it proposes creating more realistic views of interdependency through the application of more grounded methods, which enable us to move beyond current simplistic conceptualizations and begin to think about interdependence as a more multi-dimensional construct. Building

a firmer foundation for this central organizational variable should, in turn, lead to its better application in normative or causal modeling efforts.

The chapter begins below by outlining the overall approach to the survey, which included both a theoretical analysis and a simple quantitative analysis of citation rates (reported on in Appendix A). It then describes the principal analytical dimensions of the framework before proceeding to examine how representative authors from three different perspectives have studied interdependency. Each review concludes with a brief summary and assessment of the strengths and weaknesses of the perspective. Section 2.4 critiques the literature and offers suggestions about how we might achieve a more integrated framework. This chapter also reviews the various definitions and operationalizations of interdependence used by the scholars. The latter forms the basis for an assessment of the empirical study of interdependency in Section 2.5. The chapter concludes in Sections 2.6 and 2.7 with a summary and some suggestions for future research.

2.2 Overall Approach to the Survey

2.2.1 Constructing the Analytical Framework

This theoretical review draws upon analytical frameworks developed by (Astley & Van de Ven, 1983; Burrell & Morgan, 1979; Markus & Robey, 1988; Orlikowski & Baroudi, 1991) and follows the approach taken by (Allison, September 1969; Eisenhardt & Zbaracki, 1992; Kogut, 1988; Markus, June 1983). For example, Burrell and Morgan categorized theories as functionalist, interpretive, radical humanist, or radical structuralist according to their assumptions about the nature of science and society (Burrell & Morgan, 1979). Orlikowski and Baroudi propose alternative conceptualizations of technology as a key method of distinguishing among theories (Orlikowski & Baroudi, 1991). Kogut compares three perspectives (transaction cost

economics, strategic behavior, and organizational learning) on the motivation to joint venture (Kogut, 1988). Eisenhardt and Zbaracki compare and contrast dominate paradigms (rationality, bounded rationality, politics and power, and the garbage can model) on strategic decision-making (Eisenhardt & Zbaracki, 1992). These frameworks and analyses served as a starting point for examining different theories of interdependence. Appendix B outlines the dimensions used for this comparison. The results yielded the multi-dimensional metatheoretical schema presented in Figure 2.1.

2.3 A Framework for Understanding Interdependency

2.3.1 Three Analytical Dimensions

As indicated above, the interdependency literature is vast, ranging from case study illustrations to quantitative surveys, all of which span many types of technology, firms, and industries. Despite an initial appearance of randomness, a more thorough scrutiny of the literature suggests three main theoretical perspectives (information processing theories, resource-based theories, and sense-making theories), which vary along the following analytical dimensions: (1) the primary driver of interdependency (internal technology, internal firm environment, external firm environment), (2) the structure of task relationships (sequential, pooled, reciprocal), and (3) the nature or content of tasks (loose versus tight coupling). This review was shaped around these perspectives and dimensions for the following reasons.

Each of the three theoretical perspectives represents a relatively coherent and distinct body of research, and yet taken together they seem to capture the cumulative aspect of interdependency research to date. For instance, the notion that tasks are related for a variety of different factors is well established in the organization theory literature (Bowditch & Buono, 1985; Daft, 1983b). Likewise, scholars increasingly acknowledge the importance of considering theories of structure and content (Van de

Ven, May 1986). Moreover, these three research streams imply starkly different recommendations for the management of interdependence. Although in some sense all research on interdependency begins with Thompson (1967), resource-based views of the firm, as well as more recent research on the problems associated with sense-making under conditions of complexity, challenge some of information processing's basic assumptions.

For example, the information processing perspective portrays interdependence as clearly visible, recognizable, and stable dyadic interactions which are, therefore, amenable to integration and decision-making mechanisms that can be set up in advance. The resource-based viewpoint focuses on controlling resource allocation as the solution to issues of interdependence. The description of interdependency arising from the sense-making stream, in contrast, emphasizes our inability to comprehend and reason about the structure of certain forms of work and thus calls for more dynamic, real-time solution strategies.

Table 2.1 summarizes the three theoretical perspectives along some key dimensions. The next section reviews how they each relate to one aspect of organizational life, interdependency. Each paradigm review begins by describing the basic underlying model and then discusses its variants and implications. Each concludes with a summary and critique of research in that domain.

FIGURE 2.1
METATHEORETICAL SCHEMA

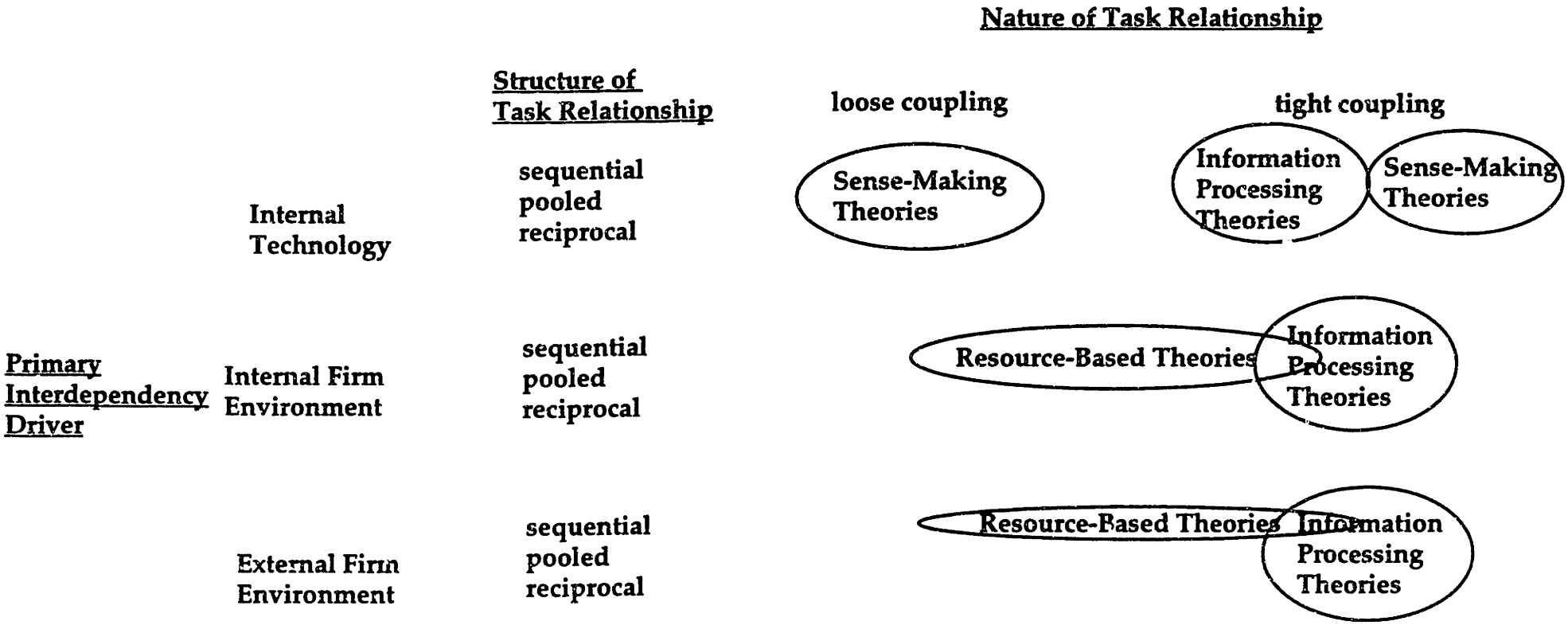


TABLE 2.1
OVERVIEW OF THREE THEORETICAL PERSPECTIVES

	INFORMATION PROCESSING THEORIES	RESOURCE-BASED THEORIES	SENSE-MAKING THEORIES
REPRESENTATIVE RESEARCH	Thompson (1967); Nadler & Tushman (1983); Tushman & Nadler (1996)	Salancik & Pfeffer (1974); Malone & Crowston (1994); Roth (1995)	Granovetter (1973); Schelling (1978); Weick (1976, 1990)
THEORETICAL STRUCTURE	realism, positivism, deterministic, nomothetic	realism, positivism, deterministic, nomothetic	nominalism, antipositivism, voluntaristic, ideographic
CONCEPT OF INTERDEPEND-ENCY	Interdependency as a (bilateral) pattern of tasks	Interdependency as a pattern of relationships within a given context	Interdependency as equivoqual systems of relationships
UNIT OF ANALYSIS	Tasks performed by work units, depts or indivs	Tasks performed by depts	Tasks performed by "things"
ANTECEDENTS OF INTERDEPEND-ENCY	Inherent in the information requirements of internal or external tasks	Uncertainty with respect to the task or environment and limited (shared) resources	Abstract nature of work; Dynamic patterns of action
CONSEQUENCES OF INTERDEPEND-ENCY	Uncertainty and information processing	Differences in power; conflict	Seemingly random and surprising events; An inability to comprehend and reason about work structure
FACTORS AFFECTING THE DIFFICULTY OF ACHIEVING INTEGRATION	Degree of contingency between tasks as represented by pattern of relationship	Criticality, value and availability of resource	Degree of coupling, timing and visibility
IMPLICIT ASSUMPTIONS	Pre-identifiable and stable tasks; clearly visible and recognizable patterns	Mutual agreement about what resources are most critical	Interdependencies are subjectively defined
PREDICTIONS DERIVED FROM THEORY	Tight couplings are most difficult to coordinate; matching coordination mechanisms to interdependency results in higher performance	Interdependency determines power relations and competitive advantage of firm	Loose couplings are most difficult to coordinate
REC'D FOR MGT	Adopt appropriate information processing mechanisms and structures	Control decisions about resource allocation or avoid interdependencies.	Impossible to design an ideal organization structure a prior; Individuals must act heedfully with respect to interdependency

2.3.2 The Information Processing Perspective

Basic Underlying Model

The information processing perspective emphasizes the uncertainty associated with performing complex tasks in a given environmental context. The most basic assumption underlying this perspective is that organizations are open systems that must process information (to accomplish internal tasks, coordinate diverse activities, and interpret an external environment) but have limited capacity to do so (Galbraith, 1973). Uncertainty is defined as "the difference between the amount of information required to perform the task and the amount of information already possessed" (Galbraith, 1973), and the technology (task) and environment are seen as the major sources of that uncertainty. Strength of interdependence between tasks, along with other task characteristics such as complexity and unpredictability, are conceived as major influences on uncertainty and hence the need for information processing. The basic underlying model of interdependency according to the information processing perspective is therefore:

Interdependency Among Internal or External Tasks ----------> Uncertainty ----------> Information Processing

Table 2.2 displays work representative of the information processing perspective

TABLE 2.2
INFORMATION PROCESSING PERSPECTIVE, SELECTED STUDIES

Author(s)	Key Word(s)	Method	Sample	Description
Thompson, 1967	internal interdependence (pooled or generalized, serial or sequential, reciprocal)	theoretical	not applicable	classifies interdependency according to its logical structure
Van de Ven, Delbecq & Koenig, 1976	task interdependency within work units	survey	197 work units in a large state employment security agency	classifies alternative mechanisms for coordinating work activities
Tushman, 1979	task interdependence within and across sub-units	survey and communicat-ion records	44 R&D projects	relates sub-unit structure and performance with work characteristics
Kmetz, 1984	maintenance work flow	descriptive case study	7 sites in U.S. Naval Air Systems	analyzes work flow problems and informal adjustments
Allen, 1986	subsystem interdependence	theoretical	not applicable	interaction of project characteristics guides decision about organizational form
Gresov, 1990	external (work unit) interdependence	survey database	230 work units in employment security offices	explores effect of external dependency on work unit design and efficiency
Eppinger, Whitney, Smith & Gebala, 1994	technical relations among design tasks (serial, coupled, parallel)	matrix analysis	2 cases from auto industry	links structure of complex products and projects with coordination and cycle time
Wagerman, 1995	outcome, goal and reward interdependence	intervention	150 teams in large corporation	examines differential effect of task design and reward system on group effectiveness

Variants

Internal Technical Interdependency

Early theorizing by Thompson is particularly influential within this research stream (Thompson, 1967). Thompson characterized an organization as "a complex set of interdependent parts which together make up a whole because each contributes

something and receives something from the whole, which in turn is interdependent with some larger environment (p. 6)." Holding the environment constant, he defined internal interdependence as "the extent to which a task requires organizational units to engage in work flow exchanges of products, information and/or resources and where actions in one unit affect the actions and work outcomes in another unit" (p. 54). When interdependence is high, frequent and unexpected adjustments are needed, and uncertainty increases in the form of "constraints and contingencies;" when interdependence is low, organizational units experience greater autonomy, stability, and certainty with respect to their coordination.

According to Thompson, internal interdependence stems from the task requirements and can be classified according to its form or logical structure. Pooled or generalized (internal) interdependence exists when "each part renders a discrete contribution to the whole and each is supported by the whole although the parts do not interact in any direct way. ... They are interdependent in the sense that unless each performs adequately, the total organization is jeopardized" (p. 54). One example provided is the relationship between corporate headquarters and each of the product divisions of a multidivisional firm, which are mutually dependent in terms of viability. Interdependence may also take a serial or sequential form in which the output of one part is the input to another. Here "direct interdependence can be pinpointed between them, the order of that interdependence can be specified... and it is not symmetrical" (p. 54). A cited example is the relationship between production departments in a manufacturing plant. A third form of internal interdependence, labeled reciprocal, refers to situations in which the outputs of each part become inputs for the other. An example is the relationship between R&D and manufacturing in product innovation.

According to Thompson, the three forms of interdependence represent different degrees of contingency, which translate into varying degrees of coordination difficulty. With pooled interdependence, contingency is non-existent or minimal. Action in each position can proceed without regard to action in the other positions as long as the overall organization remains viable. With sequential interdependence, however, there is always an element of potential contingency since each task in the set must be readjusted if another departs from expectations. With reciprocal interdependence, such contingencies are not merely potential but actual.

In summary, Thompson conceptualized interdependence as a dyad of tasks with a definite, visible, and stable structure. That pattern arises from the nature of the tasks, and the implied level of analysis is work unit relationships (i.e., tasks are defined at the level of work units). Thompson's definition of interdependency is also one predicated on literal action; unless tasks "interact in a direct way," interdependence is assumed to be minimal. Thus, mutual or pooled relationships are depicted as lowest in the hierarchy of contingency. As described below, subsequent scholars have taken issue with this viewpoint, arguing that tasks can be highly interdependent even if they involve little or no direct interaction.

Thompson's research is purely theoretical; he includes no empirical verification of his ideas other than some general illustrative examples. However, a line of subsequent empirical research and additional theorizing lends some credence to this early work. For example, Eppinger et al. focus on the form or pattern of task relationships in complex design projects, contrasting interdependent (coupled) tasks ("when task A needs information from task B, and task B also requires knowledge of A's results"), dependent (serial) tasks ("if task B simply requires the output of task A"), and independent (parallel) tasks ("if tasks A and B could be performed simultaneously with

no interaction between the designers") (Eppinger, et al., 1994). Other studies support a positive relationship between task characteristics such as difficulty, variability, and nonroutineness indicative of high uncertainty and interdependence and the amount of information processing within units (Daft & Lengel, May 1986; Gerstberger, 1971; Gerstenfeld, 1967; Tushman, 1976; Van de Ven & Delbecq, June 1974). Van de Ven et al. found that communication increased as intra-unit task interdependency among participants increased (Van de Ven, Delbecq, & Koenig, 1976). Kmetz applied an information processing perspective in a detailed qualitative study of the repair process and maintenance work flow aboard U.S. navy aircraft carriers (Kmetz, 1984).

The above group of authors and others have followed closely in the Thompson tradition, gradually translating Thompson's term 'internal interdependence' into the more general label of task interdependence or simply interdependence but making few modifications to the basic conceptual idea in Thompson's thesis. Another common term is work flow interdependence (e.g., "the extent to which individuals are dependent on other personnel in the performance of their jobs") (Van de Ven, et al., 1976). Vaughan defines it as informational interdependencies (Vaughan, 1990). Note how we are beginning to see diversity not only in the terminology used to describe interdependency but also in the level of analysis (i.e., tasks are associated with work units or individuals).

Another group of scholars, while still remaining in the Thompson tradition, portray tasks in terms of the architecture of a problem or product. For example, Allen defines subsystem interdependence as the extent to which work on one subsystem depends upon progress on another subsystem area and argues that the degree of interdependence is determined by the complexity of the interface requirements among the different areas (Allen, November 1986). Pimmler and Eppinger studied product component interdependencies varying along four different dimensions: spatial

interaction (the need for adjacency or orientation), energy interaction (the need for energy transfer), information interaction (the need for information or signal exchange), and material interaction (the need for actual physical material transfer) (Pimmler & Eppinger, 1994). Each component interaction was operationalized as a vector of four scores. Other work explores the implications of alternative product architecture on interdependency (Cusumano & Selby, 1995; Dellarocas, 1995; Henderson & Clark, 1990; Iansiti, 1994; Ulrich & Eppinger, 1995).

A search of the literature revealed a final category of information processing scholars who have sought to refine the definition of task interdependency. For example, Wagerman argues that outcome interdependency is used synonymously with task interdependency when they are in face conceptually and often empirically distinguishable (Wagerman, 1995). Outcome interdependence, which can be further differentiated into goal interdependence and reward interdependence, can exist without any interdependence in the means of accomplishing the work (e.g., a room full of telemarketers may be held accountable for a collective goal, but they complete independent tasks) (Wagerman, 1995) and vice versa (Mitchell & Silver, 1990).

Pennings proposes four distinct bases of interconnectedness: task interdependence, rooted at the task level, refers to the inter-relationship of a set of discrete operations such that each operation may have consequences for the completion of some others; role or positional interdependence is the interconnectedness of a set of role players, reflecting the position of actors engaged in a concerted action; skill or knowledge interdependence arises from the differentiated expertise of actors due to education, training, and expertise; social (goal or need) interdependence is defined in terms of the reward system and its impact on individual motivation (Pennings, 1975).

Empirical research supports making these kinds of distinction. For example, studies suggest that group members experience task and outcome interdependency differently such that changes in one form influence the experience of the other and consequentially change the way people approach their work (Berkowitz, 1957; Guzzo & Shea, 1987). Andres tested the differential effect of task and goal interdependence on software project success (Andres, August 25-27, 1995).

External Environment Interdependency

The focus on internal (task) interdependence yielded a rich stream of research but also provoked a countersurgence of studies stressing the importance of external forms of interdependency. (In fact, Thompson (1967) mentioned organization-environment interdependence but was less specific about it relative to his treatment of internal interdependencies.) Gresov points out that the bulk of the research has been directed at documenting task-design linkages to the neglect of other important context factors (Gresov, 1990). He defines external interdependence or work unit interdependence as existing "when task performance is related to (and thus dependent on) the actions and outcomes occurring outside the unit," operationalized as the extent to which resources or information from outside sources are deemed necessary inputs to a work process (Gresov, 1989). Ancona and Caldwell define a similar concept at the group level describing external interdependence as "the external activities groups undertake in dealing with others in the organization" (Ancona & Caldwell, 1990).

Gresov argues that although work unit and task interdependency are obviously related, they are distinct concepts; work unit interdependence generally arises in connection with a unit's position in the organizational work flow whereas task interdependence arises from the content of the work flow in a particular focal unit (Gresov, 1989). Empirical research tends to substantiate this claim. Van de Ven and

Ferry performed correlation and regression tests between dependency, external communication, and efficiency in a sample of employment security agencies (Van de Ven & Ferry, 1980). The results indicated that the effects of external dependency and communication varied depending on the task. In a study of R&D projects, Tushman found that both task predictors and department dependency were significantly related to structure (Tushman, 1979). Other similar work is (Adler, March-April 1995; Ito & Peterson, 1986).

Just as we saw in the case of technical (task) interdependency, scholars have sought to refine the external interdependency concept. Some authors differentiate between different types of external interdependence depending on the hierarchical relationship of the work units involved: horizontal dependency (other units inside or outside the organization) versus vertical interdependency (higher levels within the organization) (Gresov, 1990). Tushman (Tushman, 1976) and Gerstberger (Gerstberger, 1971) use a more subjective categorization, distinguishing between organizationally "close" (outside the R&D project but within the same department) and organizationally "distant" (outside the department but within the organization) relationships. Others have suggested that there exist two layers of unit context: organizational context (composed of factors that describe or affect the state of the organization as a whole and its interface with its environment) and inter-unit relations (the more immediate context of the unit) (Gresov & Stephens, 1993).

Implications of the Model

The primary goal of the information processing paradigm is to match the organization's capacity to process information with the information requirements of the task. This line of reasoning began with Galbraith who integrated work by others (Burns & Stalker, 1968; Lawrence & Lorsch, 1967; Perrow, 1967; Thompson, 1967; Woodward,

1965) to explain the variation in organizational form as stemming from the amount of information needed to reduce task-related uncertainty and attain an acceptable level of performance (Galbraith, 1973). The focus of the perspective is, therefore, on finding or "matching" information processing mechanisms, which are capable of coping with the level of task contingency.

For example, Thompson (1967) proposed parallels between his three types of internal interdependence and three types of coordination. Standardization, involving the establishment of routines or rules, which constrain the action of each unit into paths consistent with those taken by other interdependent units, is appropriate for pooled interdependence. Plans, involving the establishment of coordinated schedules, address the needs of serial interdependence. Coordination by mutual adjustment, analogous to March and Simon's term "coordination by feedback" (March & Simon, 1958) transmits new information during the process of action and is thus best suited for cases of reciprocal interdependence.

Van de Ven et al. show how impersonal, personal, and group modes of coordination vary with increased task interdependency (Van de Ven, et al., 1976). Research in the auto industry also tends to support the contingency hypothesis in that teams which adopt mechanisms of "integrated problem solving" (e.g., overlapping problem solving phases, frequent and continuous communication, cross-functional structures) show improved performance (Clark & Fujimoto, 1991; Clark & Wheelwright, 1992).

Galbraith integrated Thompson's ideas with his own and identified two general classes of solution strategies (Galbraith, 1973). Provided the interdependency is relatively simple (as is presumably the case with pooled and sequential forms), rules,

programs, hierarchy, and targets or goals are usually adequate. As complexity increases, however, (e.g., reciprocal interdependence) additional mechanisms are needed to handle the information overload. For example, one can seek to reduce the need for information processing by lowering performance standards, utilizing slack resources, or creating self contained tasks. A second category of solutions is to increase the organization's capacity to handle more information by investing in vertical information systems or lateral roles. That is, one increases the amount of information flowing across the interdependency in order to make coordination more efficient.

The large body of research on task partitioning adheres to the reductionist approach (Alexander, 1964; Eppinger, et al., 1994; Simon, December 1962; Von Hippel, 1990). These theorists propose that organizational work can be divided up (partitioned) into a number of sub-tasks in such a way as to reduce the problem solving interdependencies among them. Simon theorized that partitioning work into hierarchical systems, defined as subsystems that in turn have their own subsystems, represented one such solution:

> Hierarchies have the property of near decomposibility. Intra component linkages are generally stronger than inter component linkages. This fact has the effect of separating the high frequency dynamics of a hierarchy-- involving the internal structure of the components-- from the low frequency dynamics involving interaction among components (Simon, December 1962).

Von Hippel argues that interdependence can be reduced via alternate specifications of the task-- in other words, task partitioning is a manipulatable variable (Von Hippel, 1990). (Though he later notes that such partitioning changes will not necessarily reduce the total amount of interdependency; they simply affect how it gets distributed.)

Other researchers have taken a more model-based approach. Building on work by Steward (Steward, August 1981), Pimmler and Eppinger describe a technique called the Design Structure Matrix (DSM) for analyzing the complex interactions between components of a product design (Pimmler & Eppinger, 1994). (See also (Eppinger, et al., 1994; McCord & Eppinger, August 1993; Morelli, Eppinger, & Gulati, August 1995; Smith & Eppinger, December 1994).) Dellarocas developed a programming language, which analyzes software component interactions and semi-automatically generates "coordination software" to glue them together, an extension of object-oriented programming techniques (Dellarocas, 1995).

Examples of the capacity-increasing approach are increasing the capacity of existing channels of communication, creating new channels, introducing new decision-making mechanisms, or utilizing direct contact, liaison roles, task forces, teams, integrating roles, and matrix designs (Galbraith, 1973). For example, when task nonroutineness or interdependence is high, information processing tends to shift from impersonal rules to personal exchanges including face-to-face and group meetings such as task forces and committees (Van de Ven, et al., 1976). Daft and Macintosh found that managers favor rich communication media such as face-to-face meetings and telephone conversations for difficult and equivocal messages over other forms of communication such as impersonal memos and documents because the former allowed for greater feedback, cues, and language variety (Daft & Macintosh, 1981). Examples of full-time integrators are product and brand managers (Lawrence & Lorsch, 1967) and gatekeepers (Allen, 1977). Ancona and Caldwell identified five sets of boundary spanning activities that new product development teams rely on to interact with their environment (ambassador, task coordinator, scout, sentry, and guard) (Ancona & Caldwell, 1987; Ancona & Caldwell, 1990). Task coordinator involved coordinating

technical and design issues, scouting consisted of general scanning for useful information, and guard activities were those intended to avoid the external release of proprietary information. Adler presents a taxonomy of design-manufacturing mechanisms which distinguishes four modes of interdepartmental interaction (standards, schedules, mutual adaptation, and teams) in each of three temporal phases (pre-project, product and process design, and manufacturing) (Adler, March-April 1995).

Summary and Critique

In summary, we can identify two main branches of research stemming from the classic information processing model. One branch of sub-theories has kept relatively close to the tradition of Thompson (1967), viewing task interdependence as a relationship among internal tasks stemming from the nature of the work, and simply applied tasks to different levels of analysis (e.g., individuals, work groups, departments) or clarified the basis of connection (e.g., pure task, goal, reward, outcome, role, skill or knowledge). A second branch of research expanded the definition of task to include internal and external (environment) activities.[1] The common thread uniting this work is (1) a focus on the pattern or flow of work between entities, (2) an assumption that the nature of the task (i.e., form and content of the work flow) or the environment are the sources of that contingency pattern, and (3) the identification of the degree of direct contingency between tasks as a critical dimension affecting coordination difficulty. In particular, activities that are tightly and directly linked are viewed as being the most difficult to coordinate.

---

[1] This splintering of theory potentially stems from (and has surely contributed to) confusion over Thompson's original use of the word "task." Scholars have subsequently interpreted the label as referencing any or all of the following: a relationship between tasks, the source of the contingency, or the direct (literal) interaction of tasks.

This perspective is heavily rooted in the economic paradigm of a deterministic world and reflects the nineteenth century physical science belief that dynamics always yield unique and predictable outcomes (Orlikowski & Baroudi, 1991). The recommendations stemming from this research stream rest on the key assumptions of determinism, objectivity, and stability. Information processing theorists look at interdependence from an outsider's objective perspective; tasks are assumed "to be" reciprocal or sequential. Other research suggests that subjective judgments about interdependency may vary considerably within a firm due to the differentiation process (Lawrence & Lorsch, 1967; Lorsch, 1977; March & Simon, 1958). Because departments differ in terms of their degree of structure, leadership style, tolerance for ambiguity, etc., members of each unit will tend to see interdependencies that involve them with other units primarily from their own point of view (Lorsch, 1977). Not only does such variance in perception mean that members may perceive the same interdependency differently. They also may deem different interdependencies as being more or less important.[2]

A second major characteristic (and limitation) of research in this category is its focus on predictable, static tasks, thereby ignoring situations where tasks cannot be fully specified in advance as well as unpredictable aspects of timing. For example, one could imagine having a very routine, serial interdependency, but when and how the sequential actions take place could depend on certain unpredictable stimuli. March and Simon define contingent interdependencies as those for which timing is a major uncertainty (March & Simon, 1958).

---

[2] Some scholars get around this problem by defining interdependency as existing only when relationships are "consensually validated" (Gresov & Stephens, 1993).

2.3.3 The Resource-Based Perspective

Basic Underlying Model

Like the information processing perspective, resource-based theories start with the observation that organizations exist in an uncertain task and external environment, but this view shifts the emphasis from information processing to the resources organizations need to remain viable and compete. The resource-based perspective conceptualizes firms as unique bundles of accumulated tangible and intangible resources, defined broadly as assets, capabilities, processes, routines, and knowledge (Barney, 1991; Wernerfelt, 1984). Interdependence is the pattern of task relationships resulting from the flow and control of critical and valued resources, which reduce task or environmental uncertainty. Thus, according to this perspective, interdependencies are the outcome of trying to cope with uncertainty, not its source:

Task or Environment ----------> Flow & Control of Resources ----------> Interdependence
Uncertainty

Table 2.3 summarizes selected studies within the resource-based perspective.

TABLE 2.3
RESOURCE-BASED PERSPECTIVE, SELECTED STUDIES

Author(s)	Key Word(s)	Method	Sample	Description
Salancik & Pfeffer, 1974	dependency	ratings and rankings from interviews	29 university departments	shows how department power results from acquisition of external grants and contracts
Clark & Fujimoto, 1989; 1991	problem solving cycles among activities	questionnaire, in-depth interviews and documents	24 automobile development projects	studies effect of product and project characteristics on development lead time
Ancona & Caldwell, 1990	external interdependency	questionnaire	45 new product development teams in 3 industries	identifies four strategies teams use toward their environment
Ancona & Caldwell, 1992a	external activities	interviews, log data and questionnaire	new product team managers in high technology companies	links type of external activity and group strategy to performance
Iansiti & Clark, 1994	internal and external integration	questionnaire and case studies	29 automobile development projects, 27 computer development projects	illustrates dynamic processes used to build and integrate knowledge and solve problems
Malone & Crowston, 1994	dependencies among activities	theoretical survey	not applicable	reviews coordination process across disciplines
Roth, 1995	international interdependence	regression analysis	74 CEOs in global companies	show how influence of locus of control, information evaluation style and international expansion on firm performance vary with interdependency
Miller & Shamsie, 1996	property-based and knowledge-based activities	regression analysis	7 Hollywood film studios	relates different kinds of resource-based activities to performance in different environments

Variants

Resource Flow Interdependency

One variant of the resource-based perspective on interdependency focuses on the implications of flows of resources within and across organizational boundaries. This

variant can be further sub-divided into a stream of research emphasizing power and politics and a more recently emerging set of strategy literature.

The power and politics stream can be traced back to Crozier who, in a study of a French factory, observed that power accrued to the plant's maintenance engineers because they possessed the skill and knowledge relevant to the repair of equipment, an area of uncertainty affecting plant operations (Crozier, 1964). Based on this finding, Crozier proposed that uncertainty critical to the organization's technology determined the pattern of dependency (and power) across the organizational groups.

Salancik and Pfeffer made a substantial theoretical contribution to this line of reasoning by identifying the control of critical and valued resources as an intervening variable between uncertainty and interdependence (Salancik, 1987; Salancik & Pfeffer, 1974; Salancik & Pfeffer, 1988). Although their thesis focused on the power implications, these authors describe interdependence among and within departments as reflecting the historical flow of resources into an organization and the role those resources play in its functioning. Salancik and Pfeffer tested their theory in the context of a large university where they reasoned that ensuring an adequate flow of grant money addresses an important type of uncertainty. They found that to the extent that departments contributed more funding, they were relatively more powerful (Salancik & Pfeffer, 1974).[3]

The power view is also reflected in research on new product development projects, which emphasizes that frequent political communication (typically external)

---

[3] Other significant findings coming out of this research were that departments receiving resources are in a lower power position than those providing them, and the number and strength of dependencies are important. A department that depends on many other departments is in a low power position; a department that supplies resources to many departments is in a strong power position (Salancik & Pfeffer, 1974).

leads to higher performing development projects by increasing the resources (e.g., budget, personnel, equipment) available to the team (Ancona & Caldwell, 1992; Brown & Eisenhardt, April 1995). For example, the ambassador role in Ancona and Caldwell's typology consisted of activities such as lobbying for support and resources.

A resource-based view of the firm has also recently emerged in the strategy literature.[4] This stream articulates the relationships among firm resources, capabilities, and competitive advantage, arguing that the flow and control of valuable, costly to copy resources and capabilities represent the key sources of sustainable competitive advantage (Barney, 1991; Peteraf, 1993; Prahalad & Hamel, 1990; Rumelt, Schendel, & Teece, 1991; Wernerfelt, 1984). For example, Roth portrays international firms as a collection of interdependent resources in different locations that must be connected or integrated to some degree (Roth, 1995). See also (Clark & Fujimoto, 1991; Iansiti & Clark, 1994; Wheelwright & Clark, 1992).

Several points are worth emphasizing about the resource-power and resource-strategy lines of research. First, it is often not clear whether these theories are referring to internal or external interdependencies. Some scholars interpret this research as primarily emphasizing external interdependency (Brown & Eisenhardt, April 1995). Others argue that one of the advantages of a resource-based perspective is that it integrates false debates about the relative importance of internal versus external factors (Hart, 1995). We also see, as was the case in information processing theories, considerable variance in the level of interdependent units.

[4] The resource-power perspective articulated by Salancik & Pfeffer is sometimes referred to as strategic contingency theory.

The concept of resources also remains an amorphous one (Miller & Shamsie, 1996). For example, the early work by Crozier emphasized skill and knowledge resources, a theme carried through in more recent research (Teece, Pisano, & Shuen, 1990). Other scholars distinguish between property-based and knowledge-based resources (Miller & Shamsie, 1996). They argue that the former are likely to contribute most to performance in stable and predictable settings whereas the latter will be of the greatest utility in uncertain (changing, unpredictable) environments. Another weakness of this literature is its lack of explication of the interdependency consequences of resource flows. Most of the researchers cite interdependency as a key intervening variable but primarily focus on the power or competitive performance implications of those linkages (Hart, 1995; Salancik, 1987).

Resource Sharing Interdependency

The above research depicted interdependency as existing when materials, money, or knowledge flow between organizational units in one direction, a form of sequential interdependency (Thompson, 1967). Another variant of the resource-based perspective has explored the mutual interdependency relationships resulting from resource sharing.

This line of reasoning can be traced back to March and Simon, who refer to interdependence as a "felt need for joint decision making" and note that "the greater the mutual dependence on a limited resource, the greater the felt need to coordinate" (March & Simon, 1958). Malone and Crowston likewise propose that, whenever multiple activities share some limited resource (e.g., money, storage space or an actor's time), a resource allocation process is needed to manage the subsequent interdependencies (Malone & Crowston, March 1994). Note that in contrast to the resource flow literature, this variant of research tends to emphasize primarily internal

interdependency. The portrayal of resources is also slightly different with less of an emphasis on knowledge associated capabilities in favor of physical or capital-based assets.

Implications of the Model

The management implications coming out of the resource-based perspective are relatively straightforward. In general, they suggest that one should assess one's interdependencies and attempt to influence or even control decisions about critical resource allocation. The ultimate goal, according to this viewpoint, is to minimize interdependencies as much as possible, discretion being the ultimate and most important resource (Salancik, 1987).

For example, the resource-strategy literature emphasizes control (ownership) of rare, specific, non substitutable resources that are difficult to imitate (Teece, et al., 1990). Ancona and Caldwell propose different product development team roles for controlling resource flows (i.e., ambassador, scout) (Ancona & Caldwell, 1990). March and Simon contrast solutions involving coordination by plan (i.e., preset schedules) and coordination by feedback (i.e., mutual adjustment) and propose that the more stable and predictable the context, the greater the reliance on plans and preset schedules (March & Simon, 1958).

In a survey of coordination processes, Malone et al. identified rules such as "first come/first served," priority ordering, budgets, managerial decisions, and market-like bidding as alternative ways of managing shared resource interdependency (Malone, Crowston, Lee, & Pentland, 1993). Malone and Crowston report specifically on the use of various forms of information technology including cooperative work tools to coordinate activities (Malone & Crowston, March 1994).

Summary and Critique

In summary, we can again identify several variants of the resource-based perspective on interdependency. One stream of research depicts interdependencies as arising from the flow of resources between a supplier and consumer (activity) and focuses on the implications of that interdependency for either the distribution of power within an organization or the strategic advantage of a firm. A second branch of theories portrays interdependencies as stemming from shared access and/or use of a common stock of resources. What unites this work is an assumption that there are three necessary and sufficient conditions for the creation of interdependency: (1) resource demand, (2) limited availability, and (3) unequal allocation.

Note how this conceptualization of interdependency both resembles and differs from that in information processing. According to both viewpoints, interdependence exists among the tasks or activities in an organization and is evident in the ability (or inability) of a sub-unit to take (or not take) actions that are desired by others. Studies within each perspective are also quite inconsistent with respect to the level of task execution (i.e., individual, group, department, firm). But, in the resource-based perspective, interdependency is not rooted in the task or environment as information processing theories suggest. Rather, it is situational and varies depending upon the demand, supply, and value of a particular resource.

In other words, resource-based interdependency is an attribute of a relationship within a particular context, not a task. This is consistent with work from sociology, which defines dependency as a property of a social relationship as opposed to an attribute of a person (Emerson, 1962). Also reflecting a sociology perspective is the implicit note of conflict that runs through many of these theories but is largely absent in

the more objective information processing lens. For example, Deutsch suggests distinguishing promotive interdependence in which units depend on one another in positive and negative ways from contrient interdependence or pure conflict of interest (Deutsch, 1973). Pfeffer and Salancik make a similar distinction, labeled symbiotic versus competitive interdependency (Pfeffer & Salancik, 1978).

2.3.4 The Sense-Making Perspective

Basic Underlying Model

Like the previous two perspectives, the sense-making paradigm acknowledges the complexity and uncertainty of organizational work. But rather than examining dyadic interdependencies (information processing) or resource linkages among common units (resource-based), scholars within this line of research attempt to relate micro-level interactions to macro-level patterns and, therefore, bridge the two levels of theory. These researchers argue that one cannot simply extrapolate from the local to the aggregate because individual incentives and motives are rarely attuned to some collective accomplishment (Schelling, 1978).

A central concept in this perspective is that of equivoque; technical and organizational systems are equivocal insofar as they are amenable to several possible or plausible interpretations (Weick, 1990). Theories of sense-making suggest that the transformation and interaction of local (micro) relationships results in interdependency relationships that are so complex that people have limited and variable ability to reason about and understand the macro structure of their work. Note how, according to this model, complexity is an output of interdependency, not its source:

Micro Task <----------> Macro Level Patterns ----------> Complexity & Equivocality
Interdependency

Table 2.4 contains some of the key sense-making references.

TABLE 2.4
SENSE-MAKING PERSPECTIVE, SELECTED STUDIES

Author(s)	Key Word(s)	Method	Sample	Description
Granovetter, 1973	small scale interactions	theoretical	not applicable	proposes links between strength of dyadic ties and macros sociological theories
Weick, 1976	loose coupling	descriptive case study	educational organizations	proposes that loose couplings are ubiquitous and functional
Schelling, 1978	contingent behavior	theoretical	not applicable	explores the relationship between behavior of individuals and social aggregate
Perrow, 1984	interactive complexity	case study	nuclear power plants	proposes that coincidence of tight coupling and technical complexity create normal accident failures
Hutchins, 1990; 1991	activities	descriptive case study	navigation team	illustrates how real-time adaptations are essential to flexible system deployment
Resnick, 1992	actions and interactions	computer simulation	students	probes how people think about decentralized systems
Weick & Roberts, 1993	cognitive interdependence	descriptive case study	flight operations on aircraft carriers	suggests that continuous, high reliability situations require 'heedful inter-relating'

Variants

As this research stream is the newest and least fully developed of the three paradigms, clearly identifiable variants have yet to fully emerge. Instead, this section begins by noting some mainly descriptive studies illustrative of this perspective and then describes two of the central concepts in this literature having to do with the nature of the interdependent tie.

Descriptions of Interacting Systems of Interdependency

Schelling presents one of the best introductions to this perspective in a book entitled Micromotives and Macrobehavior (Schelling, 1978). Drawing upon a series of mundane yet compelling examples (e.g., ant colonies, people waiting in line, Christmas card exchanges, traffic jams), he explores the relationship between the behavior of individual actors who compose some social aggregate and the characteristics of the aggregate and notes how the motives of individuals can sometimes lead to striking and unexpected outcomes:

> These situations, in which people's behavior or people's choices depend on the behavior and choices of other people, are the ones that usually don't permit any simple summation or extrapolation to the aggregate. To make the connection we usually have to look at the system of interrelationships between individuals and their environments, that is, between individuals and other individuals or between individuals and the collectivity (Schelling, 1978).

Examples of other rich descriptive research are work on the processes teams use to navigate a large ship (Hutchins, 1990; Hutchins, February 1991), the interdependent know how of flight operations on aircraft carriers (Weick & Roberts, 1993), the different forms of interdependence leading to the space shuttle Challenger disaster (Vaughan, 1990), and experiments with an interpersonal computer game (Resnick, 1992).

Strong versus Weak Interdependency

In an early precursor of network theory, Granovetter considered the macro implications of one aspect of small scale interaction, the strength of dyadic ties (Granovetter, 1973). Defining the strength of a tie as a (probably linear) combination of the amount of time, emotional intensity, intimacy, and reciprocal services flowing between two points, he proposes that the stronger the tie between A and B, the larger the overlap in their friendship networks (defined as the proportion of individuals to

whom they will both be tied out of the set of people with ties to either or both A and B).[5] One possible interpretation of this is that the stronger one type of interdependency ties two units together, the greater the overlap in the set of their other interdependencies.

The flip side of strong ties are weak ties, one example of which is a bridge or line in a network, which serves as the only path between two points (Granovetter, 1973). Weak ties are, therefore, more likely to link members of different small groups than are strong ones, which tend to be concentrated within groups. Granovetter stresses the cohesive power of such weak ties by showing how information can reach a larger number of people and traverse greater social distance when passed through weak ties rather than strong. Recent research by Krackhardt has explored the strength of strong ties (Krackhardt, June, 1996).

Loose versus Tight Interdependency

Closely related to the above distinction is Weick's concept of loose versus tight coupling, which he first explored in the context of educational organizations (Weick, March 1976). Weick defines loose coupling as connoting "things that are tied together either weakly or infrequently or slowly or with minimal interdependence ... such things are somehow attached, but each retains some identity and separateness and their attachment may be circumscribed, infrequent, weak in its mutual affects, unimportant, and/or slow to respond" (Weick, March 1976). Note how, whereas Granovetter defined ties in terms of the strength of the social relationship, Weick's definition adds a temporal element (i.e., "infrequently," "slowly"). Weick goes on to suggest seven potential functions of loosely coupled systems.

---

[5] Granovetter suggests two possible factors accounting for this result: time and similarity. Stronger ties tend to involve larger time commitments, and since time and attention are limited resources, this implies less time and attention for other relationships. Empirical evidence also suggests that the stronger the tie connecting two individuals, the more similar they are in various ways.

For example, loose coupling may serve as a form of buffering by lowering the probability that the organization will have to respond to each change in its environment. Loosely coupled systems may also serve as sensitive sensing mechanisms, which localize the effects of adaptation and trouble. They potentially support a greater number of mutations and novel solutions than would be the case with a tightly coupled system. Finally, Weick proposes that a loosely coupled system should be relatively inexpensive to run because it takes time to coordinate people. The reduction in the necessity for coordination implies fewer conflicts, inconsistencies, and discrepancies.

Subsequent empirical research supports some of these propositions. For example, whereas Weick emphasizes the functions and benefits of loose coupling, Perrow stresses the disadvantages and malfunctions associated with tightly coupled relationships (Perrow, 1984). Perrow categorized organizations on the basis of their complexity (linear or complex) and coupling (loose or tight) and found that tightly coupled systems were more vulnerable to breakdown. Hutchins shows that a loosely coupled work group was remarkably adaptive in the face of a change in its informational environment (Hutchins, February 1991).

Implications of the Model

The sense-making perspective is strikingly different from the previous two paradigms both in its implications for work processes as well as its prescriptions for management. Theorists working in this stream have, in particular, emphasized the difficulty of sense-making in highly complex, interdependent situations. For example, they point out that more and more work today is occurring inside machines or minds, suggesting that many newer technologies are no longer dominated by physical or even

visible determinism (Perrow, 1984; Weick, 1990). But as tasks become more automated, abstract, continuous, flexible, and complex, they also become less analyzable via traditional (rational) means such as inference or problem solving.

Weick notes that the combination of increased cognitive demands, complexity, and dense interdependence over large areas increases the incidence of unexpected outcomes that can ramify in unexpected ways. As a result, people increasingly operate in a work environment characterized by seemingly random, unpredictable events and in which they cannot analyze interdependencies or are not even aware that they exist. He proposes that such systems make both limited sense (because so little is visible and so much is transient) and many different kinds of sense (because the dense and complex interactions they embody can be modeled in so many different ways), in other words, they are equivocal (Weick, 1990). Perrow likewise points out that under conditions of interactive complexity events are minimally buffered, and people tend to lose sight and comprehension of cause and effect relations (Perrow, 1984).

The sense-making perspective questions the notion that managers and scholars can identify interdependencies by a priori analyzing task or resource structures and, therefore, differs from the two previous paradigms in terms of its implications for organization design. While all three perspectives agree that structure matters in terms of interdependency management, they offer different opinions as to the feasibility and desirability of organization design itself.

More rational theories based on information processing or resources presume that the design of structure can and should be planned in advance. In particular, we can rationally determine the appropriate structure for an organization by looking a priori at its tasks or resources. Many (although not all) proponents of sense-making argue that

the idea of pre-planned design assumes a foreknowledge of objectives, constraints and possibilities when in fact rationality and information are often limited, goals and preferences conflicting. According to these authors, organizations are often not planned, and in fact, it may be impossible to rationally and forthrightly design structures to address certain kinds of interdependence.

This debate between organization design as a process of management reflection and intervention versus organization design as self-organization strongly resembles classic debates between design and evolution (Alexander, 1964). Design refers to a process conducted by an outsider or representative of the system, as in information processing or resource-based solutions. In evolution, the search for design is conducted by the system itself in terms of itself via a series of local adaptations. For example, Hutchins followed a work group's response to a change in its informational environment and found that the resulting reorganization of work could not be attributed to the conscious reflection of its members or an outside manager (Hutchins, February 1991). Rather, it arose through local design and adaptation by individuals to what appeared to them as local task demands. Furthermore, and surprisingly, the solution reached was the one recognized in retrospect as being the "ideal design."[6]

The sense-making perspective is not entirely lacking in recommendations, however. For example, some network theorists largely reject the notion of a self-designing organization and argue that effective structure does not occur naturally but must be designed consciously and carefully (Krackhardt, 1994; Krackhardt & Stern, 1988; White, Boorman, & Breiger, 1976). They propose using computation tools and graph theory techniques to do so. Granovetter's research suggests placing individuals

---

[6] Some sense-making theorists take the more extreme view that people often assume centralized control when none exists or impose centralized control when none is needed (Resnick, 1992).

with many weak ties such as liaisons in key information diffusion spots since many of the ties are likely to be local bridges (Granovetter, 1973). Schelling's work highlights the fact that a divergence often exists between perceived individual interest and some collective goal (Schelling, 1978). What people are individually motivated to do and what they might like to accomplish together are often not aligned, suggesting the importance of incentive mechanisms.[7]

Hutchins' research on team navigation suggests that system robustness and flexibility depend on a certain level of redundancy in the distribution of knowledge and ability (Hutchins, 1990). His work also illustrates how raising the visibility of tasks linkages, by, for example, altering the physical arrangement of tools and work stations, increases people's awareness of their interdependency and need to interact. Other researchers propose that the more 'heed' reflected in the pattern of inter-relations in a system, the greater the capability to comprehend and respond to unexpected events that evolve rapidly (Weick & Roberts, 1993). Heed refers to "a disposition to act with attentiveness, alertness and care" as opposed to being focused on local situations and events. Possible promoters of heed suggested by these authors include the use of vivid stories, common language, apprentice-mentor roles, and careful socialization of newcomers.

Summary and Critique

[7] Note the need for two forms of incentive structure. Sometimes the problem is to get people to abstain from something that imposes costs on others (i.e., incent not to do something; raise awareness of impact interdependency has on others). Other times, the problem is to get them to take the trouble to do something of no benefit to themselves but great benefit to others (i.e., incent to do something) (Schelling, 1978).

In summary, sense-making theories explore the relationship between micro-level linkages and higher-level patterns and, therefore, constitute a multi-level perspective on interdependency (Rousseau, 1985), although the precise units are often only vaguely specified. Whereas information processing and resource-based theories tend to focus on the structural or process dimensions of interdependency, sense-making emphasizes the content or nature of the relationship, in particular its strength and comprehensibility. This perspective is also decidedly less rational and more dynamic than the other two, highlighting the lack of visibility and determinism governing many task relationships. Finally, sense-making assumes that interdependencies are subjectively defined. For example, these writers portray people as "responding to an environment that consists of other people responding to their environment" (Schelling, 1978), implying that the very definition of environment and interdependency depends on a subjective experience. The management implications arising from this theory largely reflect these very different assumptions.

In many ways the sense-making viewpoint represents the most contemporary and realistic paradigm of interdependency in organizations precisely because it challenges some of our most basic and simplifying assumptions (i.e., that organizations are rational with static and uniform tasks). It suggests instead that, although parts of work may be rational or amenable to rational analysis, other parts could prove more intractable. By emphasizing the need to find ways of working, coordinating and structuring that are more dynamic and flexible, sense-making theories also converge with observations made by other scholars (Eisenhardt & Tabrizi, 1995).

One major limitation of this category, however, is its lack of empirical validation. Precisely because it drops such simplifications, sense-making theories tend to be somewhat abstract, relying on purely theoretical arguments illustrated by carefully

chosen anecdotes (Granovetter, 1973; Schelling, 1978; Weick, March 1976) or highly descriptive studies in a single setting (Hutchins, 1990; Hutchins, February 1991; Weick & Roberts, 1993).

2.4 Critique: Toward An Integrative Model of Interdependency

The previous section reviewed three ways in which interdependency has been studied in organizations. The perspectives tend to conceptualize task relationships very differently and offer complementary and sometimes overlapping insights. This raises an important question: Are these perspectives talking about the same interdependencies, and thus merely alternative conceptualizations, or do they address conceptually and empirically distinct phenomena? In other words, how can we develop a more integrated model of interdependency?

As illustrated in Figure 2.1, there appears to be some degree of overlap between information processing (external) interdependencies which involve the transfer of information and those parts of the resource-based perspective which describe flows of knowledge. Wagerman has suggested the need to distinguish cases of pure resource-based interdependency (when each member can complete his part of the whole but resources such as skills and information are distributed among members) such as exists on a design team from those of pure task-based interdependency (where task accomplishment requires collective action) as represented by a basketball team (Wagerman, 1995).

Pennings points out a similar lack of distinction between technical and environmental interdependencies due to varying (and unclear) conceptual and operational definitions, which tend to emphasize a single variable (usually uncertainty) or a cluster of conceptually similar variables (Pennings, September 1975). For example,

he identifies a large number of cases where authors equate dimensions of environment with dimensions of technology and cite references to studies of technical interdependence in support of their thesis on environmental contingencies (or vice versa). The present survey revealed a similar confusion among concepts of environment; some scholars portray environment as referencing internal firm unit relations (external to a focal unit), while others define environment as strictly outside the organization.

Finally, there is a general lack of clarity and distinction among four variables: technology, environment, internal, and external. For example, are technical interdependencies and internal interdependencies always the same thing? Some authors appear to interpret internal/external as referencing the source of contingency, while for others it refers to the structure of the organization.

On the other hand, opportunities to integrate complementary viewpoints also abound. This survey suggests three in particular. First, we need to study the relationship between interdependency process and content. Process descriptions of interdependency focus on the pattern or form of the task relationship; they emphasize how tasks are contingent, the best illustration being Thompson's (1967) categorization of pooled, sequential, and reciprocal. Content-based approaches to classification address the nature of the tasks and connection.

For example, referring again to Figure 2.1, we see that information processing theories explore many kinds of structures, while the resource-based perspective has primarily concentrated on sequential and pooled forms. We might combine Thompson's three forms with Karl Weick's notion of loose and tight coupling, and ask which forms tend to be more tightly coupled. Are there examples of reciprocal

interdependencies that are loosely coupled and others that are tight? Another possible combination of content and process would link the source or primary driver of the interdependency (technology and internal or external environment) with its structure. Do technical interdependencies tend to be more reciprocal? Are there greater lags in sequential interdependencies driven by the environment? Can we identify certain ubiquitous process types?

A second possible integration would look for evidence of expected and unexpected outcomes in both micro-level dyadic relationships and higher-level patterns. Although sense-making theorists attempt to link the two levels, their work primarily focuses on the "unexpected and surprising" macro-level outcomes. Ample and compelling evidence exists from information processing and resource-based research studies that at least some task relationships are predictable, rational, and stable. How do the latter sometimes get transformed into unexpected outcomes? Where and when do they remain stable?

Finally, we need to better integrate knowledge about interdependencies with theories of organizational design and structure. Theorists have long promoted so-called congruence or contingency models of organization behavior, which focus on the degree of fit between features of context and design and its relation to efficiency and effectiveness (Bailetti & Callahan, 1995; Burns & Stalker, 1968; Lawrence & Lorsch, 1967; Nadler & Tushman, 1983). Although most theoretical formulations include multiple features of context (Burns & Stalker, 1968; Galbraith, 1973; Lawrence & Lorsch, 1967), empirical research has usually tested single contingency factors and therefore failed to fully test the fit hypothesis.

Some theorists are beginning to question the basic congruence approach and assumptions (Andres, August 25-27, 1995; Drazin & Van de Ven, 1985; Gresov, 1989; Miller, 1981; Miller, May 1992; Scott, 1990). Among other things, these scholars argue that designing to several contingencies at once involves tradeoffs that prohibit overall fit.

This viewpoint is supported by classic theories of design. For example, in Notes on the Synthesis of Form, an extensive treatise on the process of design, Alexander writes:

> Our conviction that there is such a thing as fit to be achieved is curiously flimsy and insubstantial. We are searching for some kind of harmony between two intangibles: a form which we have not yet designed and a context which we cannot properly describe. The only reason we have for thinking that there must be some kind of fit to be achieved between them is that we can detect incongruities or negative instances of it.

He continues:

> In practice, we see good fit only from a negative point of view... Even in everyday life, the concept of good fit, though positive in meaning, seems very largely to feed on negative instances; it is the aspects of our lives which are obsolete, incongruous, or out of tune that catch our attention...Misfits are the forces which must shape [design, and there is no mistaking them. Because they are expressed in negative form they are specific and tangible enough to talk about...
>
> I should like to recommend that we should always expect to see the process of achieving good fit between two entities as a negative process of neutralizing the incongruities or irritants and forces which cause misfit (Alexander, 1964).

The focus on single dyadic task relationships in interdependency research, particularly in the case of information processing and resource-based studies, has tended to promote very localized management recommendations. There may be opportunities for synergy if we can identify certain solutions which involve managing

clusters of like interdependencies. Organizations must also perform many tasks simultaneously, suggesting the possibility of coordination conflicts (Andres, August 25-27, 1995; Pfeffer & Salancik, 1983) and coordination costs and overload (Malone, February 1988; March & Simon, 1958), prospects largely ignored in early theorizing.[8]

We need to document frequently occurring design conflicts and identify structures and processes organizations can use to surmount them. One example of this approach is Allen's work, which examined the tension between various types of organization design and the information needs in R&D organizations (Allen, November 1986). We also need to develop theories of how to manage and coordinate multiple interdependencies simultaneously and efficiently.

Another potential integration is between task interdependency and pre-existing structures. Most theoretical studies focus on logically prior decisions of how to organize and group tasks into departments (Galbraith, 1973). Empirical work, almost by definition, looks at how established departments coordinate. We need to develop a better understanding of how existing structures and processes shape and perhaps dictate relationships among tasks.

Gathering and analyzing data about interdependencies may, therefore, enable us to extend our thinking about some of the existing organizational design theories. The above comments suggest, in particular, an alternative approach to achieving congruence, one focused on identifying and minimizing instances of misfit not seeking

---

[8] Thompson does propose that different types of internal interdependence form a Guttman-type scale. For example, all organizations have pooled interdependence, more complicated organizations have sequential as well as pooled, and the most complex organizations exhibit all three types. But he never considers the implications of this heterogeneity nor how one type of interdependence relates to another.

fit. They also raise questions such as do there exist multiple kinds of fit (Drazin & Van de Ven, 1985), and which interdependency misfits are most crucial.

2.5 Measuring and Assessing Interdependency

One of the most striking revelations coming out of this survey is the narrowness of the methodological and empirical approaches taken to investigate interdependence. As indicated by a quick glance at Tables 2.2, 2.3, and 2.4 and supported by further more detailed review, the vast majority of the research is either completely conceptual, case descriptive, or replicates early operational definitions (usually some variant of (Thompson, 1967) or (Van de Ven & Ferry, 1980)). There are virtually no inductive or qualitative studies of interdependence where, for example, a researcher attempted to understand interdependence as people experience it within an organization. This method bias has no doubt contributed to our narrow understanding of the concept and its implications. In other words, it is entirely possible that our definitions and understanding of interdependence are nothing more than a testimonial to our methods, in effect an artifact of using time and context independent measures (Weick, 1974).

For example, relying on questions such as "While doing your assigned tasks, how much do you have to depend on outside departments?" or "Please indicate how much of your work flows in an independent, sequential, and reciprocal manner" (Van de Ven & Ferry, 1980) to assess aspects of interdependency assumes that people know when they come across an interdependency and can reliably assess it (Weick, 1974). Such measures, therefore, obscure important details of work which have as one of their distinguishing properties the fact that people often do not even realize that problems exist for which they need solutions. Thus, to ask people about the pattern or level of interdependency can miss or mislead us about key aspects of interdependence associated with newer, more complex work arrangements.

For instance, Weick notes that if one goes into an organization and watches which parts affect which other parts, one will predominately see the tightly coupled parts. Those parts that are interdependent slightly, infrequently or aperiodically, however, will, almost by definition, be less visible. Similarly, many organizational processes could exhibit a mixed quality of interdependence if, for example, they follow a reciprocal pattern early on but sequential pattern later. But that aspect will be altogether missed (and inappropriate coordination mechanisms invoked) if the observer extrapolates from early stages of the relationship (Weick, March 1976).

Relying on self-reports or survey responses also presents possible biases. Research in social psychology suggests that people tend to over-rationalize their activities and attribute greater meaning, predictability, and coupling among them than in fact exist (Katz & Kahn, 1966). People are also prone to report unilateral causation as opposed to mutual causation insofar as the former supports a positive self concept. The tendency to describe situations in terms of causal arcs rather than loops may also reflect high levels of mobility or a variance in perception within firms (Weick, 1974). The fact that people are highly mobile (both within and across organizations) implies that they may not stay in a situation long enough to appreciate the feedback consequences of their actions. Companies and individuals can also differ widely with respect to their perception of the time span involved (Lawrence & Lorsch, 1967). For example, a marketer who operates in the fast-paced commercial world may define reciprocal interdependence as a loop occurring within a one week time frame; someone working in a research organization, on the other hand, would probably have a longer time horizon.

Researchers also represent a potential source of bias through their modeling efforts (Athey & Stern, 1996). For example, most existing research has focused on the implications of interdependency. Yet because we lack an adequate understanding of the interdependency variable itself, we risk confounding its impact with other factors. This in turn can result in over-representing instances of mutual causation insofar as coarse effects will undoubtedly appear to feed back on earlier causes (Weick, 1974).

2.6 A New Research Agenda

The previous sections looked to the past in describing three major interdependency perspectives. These paradigms represent continued areas of active research as indicated by the large number of sub-stream theories and variants generated within each category. However, this section sketches out some ideas coming out of this survey for a bolder agenda for interdependency research.

First, the agenda should begin by closely examining the concept of interdependency itself before proceeding to investigate and measure its impact on organizational outcomes. Most authors included in this review are not really studying interdependence, but rather how interdependence impacts something else, usually performance. The fact that they fail to adequately develop and operationalize the variable is, therefore, perhaps not too surprising since it is likely to be heavily influenced by the particular objectives of their study. We need a richer, more unified understanding of the interdependence variable itself before we proceed to build such models.

Second, while it is clear that each perspective on interdependency has its shortcomings, each also offers important insights. New research should, therefore, build upon this past work, guided by a goal of increased integration. The classic

debates about determinism versus voluntarism or internal versus external are not very controversial anymore. Most scholars recognize (albeit perhaps reluctantly) that relationships are sometimes deterministic but often not and that organizations represent complex, evolving entities. It is likely that interdependencies are sometimes predictable in advance but often more emergent and that some forms of interdependency are harder to manage than others. It is equally apparent, however, that our understanding of the concept rests on isolated viewpoints, which tend to constrain the realism of interdependency research.

Third, as suggested in Section 2.5, future research on interdependency must exploit new methodological approaches. In particular, rather than starting with a pre-conceived definition (and re-operationalizing it yet again), we need to go in and observe interdependency in action. Researchers studying interdependency need to apply methods that both highlight and preserve rich details of context (Weick, March 1976) and enable us to make tractable theoretical and normative statements about variations. In fact, we may need to invent (or at least agree upon) a language or grammar about interdependence before further theoretical progress can be made (Salancik & Leblebici, 1988).

Yet certain pervasive shortcomings also become apparent as a result of this survey. In particular, although the sense-making perspective has a dynamic element to it, there is a general lack of consideration of the role of time in all of this research, undoubtedly due to the cross sectional nature of most of the work. (An exception is work by Adler (Adler, March-April 1995).) Theorists tend to portray firms as starting with a clean slate of interdependent tasks waiting to be structured and coordinated, yet we know that organizations structure and restructure themselves continually over time and that their technology and environments change very rapidly. This suggests that

interdependency patterns and designs will also evolve and perhaps be constrained by their past form.

Finally, we need to develop a clearer thesis about the relationship between tasks and organizational design or structure. This survey revealed a considerable amount of confounding surrounding these two concepts.[9] The traditional viewpoint says that, given the existence of an interdependency, we can design organizational solutions to manage it. Recently, however, scholars have begun to modify this simple determinate model by depicting organizational structure as an intervening variable between tasks and interdependency (Eccles & Nohria, 1992). That is, interdependency is not inherent in the task (or environment) but rather an outcome of the organizational design and differentiation processes.

For example, Wagerman defines task interdependence as existing when "each member must take action for any other member to do any part of their work," but locates the source of such interdependence not in the task itself but rather in the "organizational structure, work instructions and materials" (Wagerman, 1995). According to this interpretation, the same tasks could conceivably exhibit different types or degrees of interdependence in different organizational structures.

2.7 Conclusion

This chapter builds upon a long history of research on interdependency. In particular, it presents a conceptual framework, which integrates interdependency research from three separate streams of theorizing in an attempt to develop a clearer

---

[9] One possible source of this confusion, revealed in this survey, is the considerable variation across studies in all three paradigms as to the level of task execution. For example, we saw numerous cases where one author's 'interdependency' was conceptualized as a form of structure in other studies. Some micro-level researchers view group-level interdependencies as a structural variable whereas group scholars portray inter-group relations as a type of interdependency.

understanding of both the similarities and important distinctions in theorists' approach to the concept.

The survey suggests that our understanding of interdependency is actually quite limited and dated. We are still primarily drawing upon the Thompsonian notions of pooled, sequential, and reciprocal forms of contingency. While that work contains some fundamental insights, it ignores some of the more recent approaches to theorizing about the nature of work in organizations and largely fails to reflect the complexity of work and work relationships in organizations today. On the one hand, we need to broaden our view of interdependence beyond that of simply a structured pattern of task relationships. More content-based viewpoints likewise need to incorporate the importance of structure and pattern in their models. Although much additional theoretical and empirical work remain to be done, this survey represents an important first step in opening up a dialogue about a fundamental concept that has not been adequately developed in recent years.

CHAPTER III: AN EMPIRICALLY-DERIVED TAXONOMY OF MULTIPLE INTERDEPENDENCIES

This chapter seeks to refine our understanding of how and why tasks are related in organizations. Analysis of interdependency in six personal computer and telecommunications software development projects suggested that product development teams face four types of interdependency that need to be managed simultaneously: Product Definition and Architecture Interdependencies, System of Use Interdependencies, Technology Sharing Interdependencies, and Resource Sharing Interdependencies. These interdependency types can be traced back to the product technology, external product environment, internal work unit environment, and organizational structures and resource policies in each firm and are further shown to vary along two dimensions, task structure and predictability. Some implications of a multiple interdependency perspective for strategy and organization design are also discussed.

3.1 Introduction

The concept of interdependency is central to our thinking about organizations. It appears in theories of organizational design (Galbraith, 1973; Nadler & Tushman, 1983), organizational process (Van de Ven & Poole, 1989), and product development (Cusumano & Selby, 1995; Wheelwright & Clark, 1992). It is implicit in classic metaphors of organizations as "systems" (Katz & Kahn, 1966) as well as more contemporary portrayals of organizations as networks (Rockart & Short, 1989) or "jazz ensembles" (Eisenhardt & Tabrizi, 1995).

Yet despite this centrality, the structure and underlying contingencies which drive interdependency remain unclear. For example, pioneering work by Thompson defined interdependency as a varying degree of contingency among tasks inherent in the nature of technology (Thompson, 1967); more recent research has focused on the role of a firm's external environment (Ancona & Caldwell, 1990; Christensen & Rosenbloom, 1995; Gresov, 1990) and its strategy (Cusumano & Selby, 1995; Iansiti & Clark, 1994) as possible sources of interdependence.

These variations suggest that there may be several types or categories of interdependence, yet a comprehensive framework has yet to be developed. A major initial research question is therefore, simply, what does a map of interdependencies within an organization look like (Weick, March 1976)? Such a mapping, in which we identify and systematically analyze a wide variety of interdependency relations and their associated management implications, would contribute greatly to our knowledge and understanding of organizations (Malone & Crowston, March 1994).

A multiple interdependency perspective also raises new research questions in terms of strategic organization design and potentially lends insight into some core organizational problems. For example, what are the challenges associated with trying to manage multiple interdependencies simultaneously? Is it possible to design organizational solutions to meet the needs of multiple interdependencies, and if not, how do managers choose which interdependencies to concentrate on? Are multiple interdependencies one potential source of the organizational design "dilemmas" cited by Pfeffer and Salancik (Pfeffer & Salancik, 1983), and if so, can we document their existence and design new coordination solutions to overcome them?

Developing a multiple interdependency paradigm which addresses such issues will take time. The process is begun here by focusing on interdependency in large scale software development projects. The next section describes the overall empirical approach of the study, and the sample, data, and analytical technique used to derive a taxonomy of different interdependency types. Section 3.3 presents the qualitative results in terms of the types and their variations and discusses how they are linked to current literature. Section 3.4 contains a quantitative analysis of the distribution of types across projects, firms, and functions. The final two sections summarize the study

and discuss how it both builds upon and enriches our current understanding of the relationship between interdependency and organizational strategy and structure.

3.2 Identifying and Classifying Interdependencies

This study is distinct from most research on interdependency in its inductive, qualitative analytical approach. In particular, it draws upon descriptions and examples from interviews and project documentation to record, often in team member's own words, what interdependencies exist and the problems they present on six case study software development projects. These examples are then analyzed using language processing techniques (described in Appendix C), yielding a taxonomy of multiple interdependency types. The result is a rich, grounded view of interdependency in new product development efforts.

3.2.1 Overview of the Analytical Approach: Classification

Classification schemes have been highly influential in developing analytical frameworks to understand how firms operate and compete (Burns & Stalker, 1968; Lawrence & Lorsch, 1967; Thompson, 1967; Woodward, 1965). There are two general approaches to classification: conceptually-derived typologies and empirically-based taxonomies (Bailey, 1994). That is, one can impose a pre-existing framework on empirical data (typology) or allow patterns in the data to emerge from an examination and comparison of cases (taxonomy).

The emergent approach examines the raw data and relies on intuition and (increasingly) specialized analytical techniques to organize and interpret the resulting patterns. One problem associated with the taxonomy approach is that it can become unwieldy when large databases must be assessed; the mass of detail makes it difficult to tease out patterns in the data that represent true empirical variations as opposed to

random fluctuations. As a result, researchers are often forced to examine a limited number of cases, which makes the generalizability of their conclusions suspect. Taxonomies are also criticized for their lack of theoretical significance, arbitrary nature, and unreliable results (McKelvey, September 1978). Typologies, on the other hand, because they involve imposing a framework to organize the data, create an artificial sense of orderliness in what may actually be a highly chaotic process. Compared to typologies, therefore, taxonomies tend to be more firmly based on facts and can disclose important empirical regularities.[10]

A second distinction that can be made is between special and general classification schema (McKelvey, September 1978). Special typologies are based on one or two organizational attributes, such as those proposed by Burns and Stalker (Burns & Stalker, 1968), Perrow (Perrow, 1967), Thompson (Thompson, 1967), and Weber (Weber, 1978). While the above are exemplary, many special classifications tend to be "thin and arbitrary" insofar as they pertain to only limited aspects of organizational life (Miller, 1996). In consequence, they have high predictive validity but within a limited slice of the total behavior of an organization. A general classification, in contrast, groups objects together according to all (or most of) their attributes, although some attributes may be weighted more than others. Since many attributes are taken into consideration, a general classification allows one to make broad predictions about the form and total behavior of members of a given class (McKelvey, September 1978).

The present analysis can be summarized as a general taxonomy. By adopting an emergent approach it avoids the disadvantage of imposing prior theoretical biases on the data and increases the chances of discovering the unanticipatable. Rather than

---

[10] Both approaches suffer from a lack of testing. Many typologies are never tested empirically; those that are often fail to be validated (Miller, 1996). Taxonomies usually yield interesting and evocative empirical hypotheses, which the authors all too often subsequently fail to develop into coherent integrated theories.

identifying one or two types of interdependency, it is a multidimensional classification, reflective of the systemic nature of task relationships.

3.2.2 The Research Domain and Sample

The complexity of both the technology and process in large scale product development suggested that it would be an ideal and exciting phenomenon within which to explore interdependency and begin to construct a taxonomy. Building and maintaining software systems represents an extremely complex activity (Brooks, 1975). This complexity arises not only from the inherent complexities of the technology, but also due to the difficulties in managing the software development process.

Software systems typically consist of millions of lines of code grouped into hundreds or thousands of files; those lines execute hundreds of different functions. For example, the first version of Microsoft Windows NT consisted of 5.6 million lines of code organized into 40,000 files; at the peak of the development cycle, approximately 200 people were on the team (Zachary, 1994). One successful real-time telecommunications switching system currently in its eighth version of release is approximately 10 million lines of non-commented code divided into 41 different subsystems; 3000 software engineers contribute to its development and maintenance (Sumner Jr., April 20, 1993).[11]

Because the products themselves are so large and complex and yet must be produced very rapidly, these development efforts are usually highly distributed and interdependent. Literally hundreds or even thousands of individuals whose activities are highly related contribute aspects of the product in parallel (Tichy, 1992).

---

[11] These figures include product, not support or test code, and technical employees, not staff or contingent workers, respectively.

One way of conceptualizing this process is as a series of implementation steps with iterative relationships between successive phases: system requirements, software requirements, analysis, program design, coding, testing, operations, and maintenance (Royce, August 1970). This waterfall-like approach, which tries to simplify the process by "freezing" a product specification early and then integrating and testing the system at the end, was common in the 1970's and 1980's and remains popular in many industries (Cusumano & Selby, 1995).

Recently, some researchers have begun to explore alternative development models such as "iterative enhancement" (Boehm & Papaccio, October 1988), "spiral" (Boehm, January 1984), "concurrent development" (Pimmler & Eppinger, 1994), "synch and stabilize"(Cusumano & Selby, 1995), and "interpretive" (Piore, Lester, & Malek, March 25, 1997). These authors argue that in many industries user needs and desires are so difficult to understand and evolve so rapidly that tasks are much more overlapping and inter-related than commonly supposed. As a result, it is impossible and unwise to design the system completely in advance. Instead, projects should "iterate" as well as concurrently manage as many activities as possible while they move forward to completing the project (Cusumano & Selby, 1995). Iteration has the added benefit of reducing the likelihood of unpleasant surprises as the product is assembled where, for instance, different physical parts overlap or fail to work together (Sabbagh, 1995).

An alternative conceptualization of product development is, therefore, repeated occurrences of a sequence consisting of a development phase and a coordination phase (Cusumano & Selby, 1995; Hauptman, 1986; Koushik & Mookerjee, 1995). According to this interpretation, coordination of tasks is a key requirement for project success, and

managers must decide how best to coordinate different activities while still meeting the project goals, schedule, and budget.

One component of coordination is the continuous day-to-day integration of individual work with the activities of other team members. For example, people working on parts in the same subsystem need to coordinate with respect to their design and testing. They also need to interact with people working on other subsystems that interface or interact with some of these components, with representatives from other functional areas such as marketing or customer support, and potentially with people on other projects within the firm (Koushik & Mookerjee, 1995).

The two participating organizations in this study were Microsoft, a leading producer of personal computer (PC) software, and Lucent Technologies (formerly part of AT&T), a maker of telecommunications switching system software and hardware products. Although in different industries, these companies exhibit high levels of product and process complexity. Both firms primarily produce large scale software products in competitive, rapidly evolving markets.[12] They also perform a variety of tasks in the production process.

Key informants helped select three projects to study at each firm with the goal of capturing many different types of interdependency. Specific project selection factors were based on past research (Boehm, January 1984; Cusumano & Kemerer, November 1990; Cusumano & Selby, 1995; Kemerer, 1997) and discussions with experienced managers in the field. Actual project names were disguised.

---

[12] One significant difference between the two firms is that Microsoft develops products for a mass market, while Lucent's products usually represent more customized forms of development.

For example, the projects were large in terms of both the number of lines of code being developed and team size, ranging from 50,000 to 5 million lines of code and 75 to 200 people. The projects within each firm varied in terms of the level of innovation required; three were first-time releases, while the others represented updates of a past product.

Tables 3.1 and 3.2 summarize the companies and projects, respectively.

TABLE 3.1
COMPANY COMPARISON

	LUCENT	MICROSOFT
CORPORATE HISTORY	110 years old 1885-1995: regulated monopoly 1997: broken apart from AT&T	20 years old
POPUL AND CULTURE	120,000 total employees primarily engineers avg. age: 45 years avg. experience: 15 years	17,000 total employees primarily comp scientists avg. age: 30 years avg. experience: 5 years
PRODUCTS	real-time switching systems that route calls through a telecom network; features that perform specific call processing operations	operating systems that control the primary operations of a computer; applications that perform specific word processing and graphical operations
Flagship Product	ESS Switch - 5-10 M NCSL; 40 subsystems - avg. downtime: 1 min/yr	Windows Operating System - 10 M NCSL
PRODUCT MARKET		
Customers	Local and long-distance telecoms, gov't agencies in US and abroad	Individual and corporate customers in US and abroad
Requirements	High reliability	High compatibility
Delivery	Annual -- > Incremental	Incremental --> Annual

TABLE 3.2
SAMPLE PROJECTS

FIRM	PRODUCT	DESC	PRODUCT SCALE	TEAM SCALE	ARCH LAYER	VERS NO.
LUCENT	"Handphone"	an application providing wireless communication in the Far East	50,000 NCSL on a code base of 10 million	~ 100 people	app	1.0
	"Tollphone"	an application providing operator services, toll and directory assistance	1-2 M NCSL on a code base of 10 million	~ 100 people	app	7.0
	"Autophone"	a switching system for mobile phones	1-2 M NCSL	~ 200 people	app/OS	1.0/2.0
MICROSOFT	"Desk"	an integrated application suite for PCs	~ 2 M NCSL	~ 200 people	app	5.0
	"Data"	an entry level database application for PCs	~ 1 M NCSL	~ 75 people	app	1.0
	"Network"	a network operating system for PCs	~ 5.6 M NCSL	~ 250 people	OS	3.0

The data consisted of a total of 71 interviews with team members across the two firms. The interviews were balanced across the six projects with the exception of one case at Lucent (Autophone), which was under-represented. The interview subjects varied in terms of their functional area and level of experience. For example, the researcher spoke with designers, system engineers, developers, testers, documenters, and upper-level managers. Due to time constraints, marketing and field representatives were not included in the interview sample.

The interviews began by asking the subject for facts, opinions and insights regarding interdependency in their work as well as basic background about their project and job. A key aspect of the interview technique was to ask people for stories or specific examples of the interdependencies they faced on the project. This approach is similar to the technique of collecting data that tell stories about organizational processes and then looking for patterns in that data (Daft, 1983a). In line with Glaser and Strauss' suggestion, interviewing ended when the results (stories and examples of interdependency) became highly consistent (Glaser & Strauss, 1967). All of the interviews were recorded on audio tape and transcribed.

The companies also granted access to various forms of documentation and archival records such as design documents, product schematics, and organizational charts. As in Eisenhardt and Tabrizi's research, attempts were made to reconcile interview data with other sources of information such as documents and reports (Eisenhardt & Tabrizi, 1995). For example, if a subject described an interdependency which involved a technical relationship between two product subsystems, it was verified via architectural design documents.

3.2.3 Specific Analytical Technique: Language Processing Methods & Affinity Diagramming

The methodology employed here to collect and classify interdependencies is adapted from a technique known as Affinity Diagramming or the KJ Method and widely used in marketing research (Griffin & Hauser, Winter 1993; Juguilon, 1996; Kawakita, 1977; Kawakita, 1991a; Kawakita, 1991b; Shiba, Graham, & Walden, 1993; Ulrich & Eppinger, 1995).[13] This technique emphasizes collecting and analyzing real cases and personal experiences within their context of use, reflecting the premise that those with direct experience (in using a product or coordinating tasks, for example) are uniquely positioned to provide information about such. The method is well suited to the present inductive analysis in that it (1) begins with a broad exploration of a situation that attempts to suppress preconceived ideas and hypotheses and (2) employs tools to gradually structure and focus the data toward specific patterns without being overly rigid (Shiba, et al., 1993).

Appendix C outlines the steps used to cluster the interdependency examples and construct affinity diagrams.

3.3 Results

3.3.1 An Overview of The Taxonomy

Table 3.3 presents an overview of the three dimensional taxonomy. The data revealed four archetypal categories of interdependency commonly encountered in large scale product development projects, each of which can be further broken down into secondary and tertiary categories: Product Definition and Architecture Interdependencies, System of Use Interdependencies, Technology Sharing Interdependencies, and Resource Sharing Interdependencies. These clusters sort

[13] KJ stands for Jiro Kawakita who originated the methodology during the 1950's.

interdependencies on the basis of the content or primary driver of the task relationship (product technology, external product environment, work unit environment, and organizational structures and policies, respectively). Figure 3.1 is the resulting affinity diagram; only the cluster titles are shown for ease of presentation.

The analysis also suggested that interdependencies can be further categorized along two additional (orthogonal) dimensions. The first is that of the structure of the task relationship. Certain binary task structures were clearly evident in the data (e.g., sequential, pooled, reciprocal), but the examples also suggested two possible additions to this basic categorization scheme. First, it is sometimes useful to refine these relationships (e.g., sequential can be differentiated into sequential interdependencies between a base and higher level component and sequential interdependencies between two components interfacing at the same level).

Even more strongly supported by the data in this study, however, is the need to go beyond binary classes of task structure and look for frequently occurring patterns of task relationships. Possible patterns suggested here are cascade (core), chain, and indirect. Cascade patterns exist when one central or core task is related to many other tasks in a fan-like manner (alternatively, it can be viewed as many sequential interdependency relations emanating from a core task). Chain interdependencies reflect the fact that some tasks require the performance of sequences of tasks in rapid or immediate succession. Indirect interdependence is a structure in which one bilateral relationship indirectly impacts other tasks.

TABLE 3.3
KEY DIMENSIONS OF THE TAXONOMY

Classification Dimension #2: Task Structure

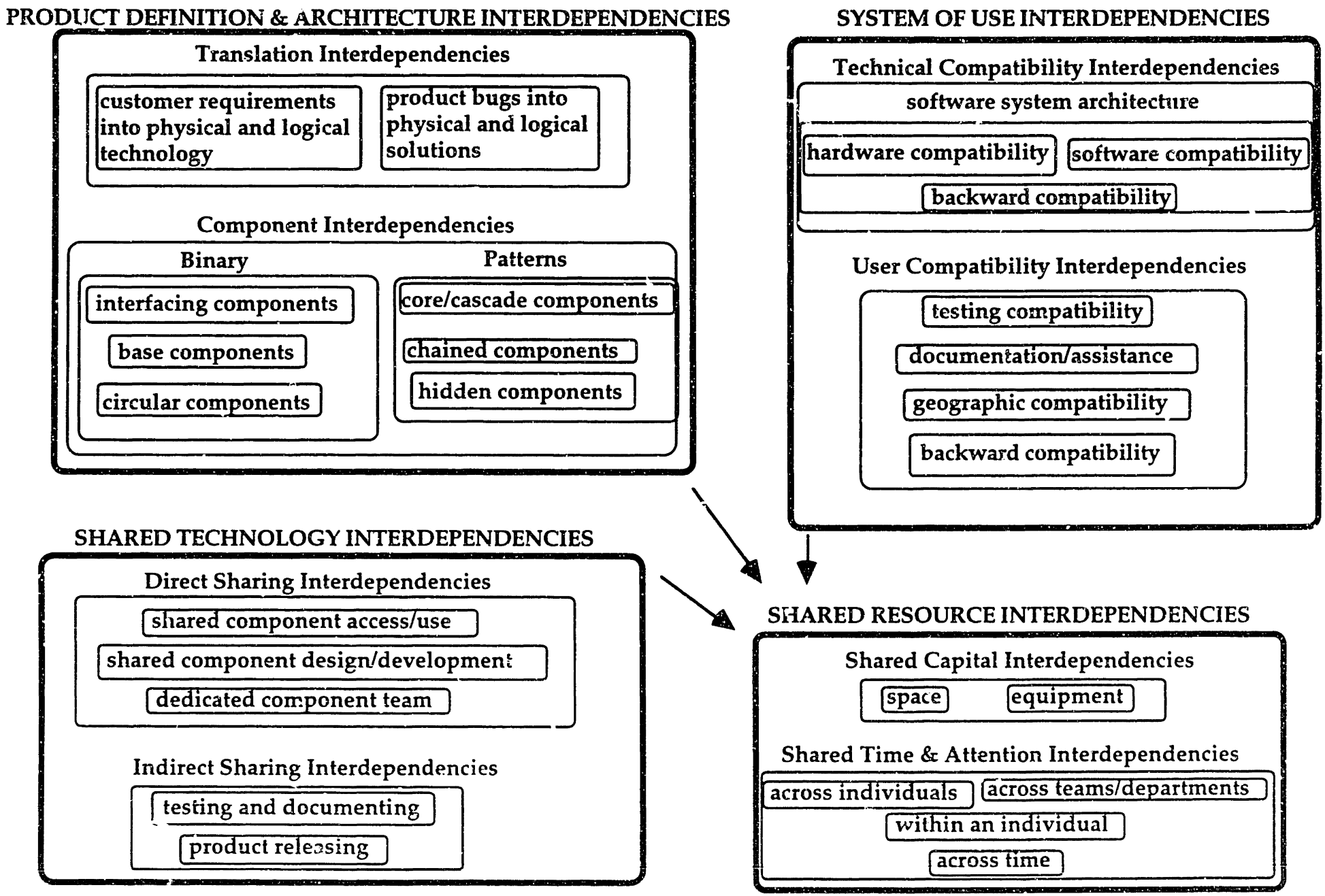
Bilateral Structures **Patterned Structures**

	Sequential	Sequential	Pooled (Mutual)	Reciprocal	Cascade (Core)	Chain	Indirect
	Base	Interface					
Product Technology							
External Product Environment				predictable hidden			
Work Unit Environment & Strategy		predictable hidden				predictable hidden	
Organizational Structure & Resource Policies							

Classification Dimension #3: Task Predictability

FIGURE 3.1: KJ DIAGRAM OF INTERDEPENDENCIES

What types of interdependencies exist in large scale product development projects?



The third dimension shown in the table is that of the predictability of the interdependency. Some task relationships are relatively predictable in advance (anticipatable, stable, visualizable) while others are more hidden and thus emergent over time.

The next section describes each primary category in more detail as well as its linkage to existing literature. References are also made to supporting KJ diagrams for each type (Figures 3.2a - 3.2d), which display both the cluster titles and representative examples from the data, and to Table 3.4, which shows the distribution of the examples across the four major categories.

3.3.2 Interdependency Types Discovered

Product Definition and Architecture Interdependencies

Overview

Technical products can be thought of in terms of both functional and physical elements (Ulrich & Eppinger, 1995). Functional elements are the operations and transformations that contribute to the overall definition and performance of the product. They correspond, at a high level, to customer needs and requirements-- what users want a product to actually do. Examples include drawing a box, calculating a mean, or distributing memory in a PC product, or making and billing different kinds of phone calls in a telecommunications product.

After gathering and defining the customer's functional requirements, the process of engineering design involves first mapping the functional elements to physical implementation-level entities and then clustering the physical elements into some sort of hierarchy or grouping. Physical elements are the parts that ultimately implement such features and functions. In software products, physical elements usually take the

form of files, modules or, more generally, components and subsystems. This process of translating customer requirements into a technical product consisting of physical parts and logical parts and connections and then breaking the physical parts down into components and subsystems creates a set of interdependent tasks labeled here, Product Definition and Architecture Interdependencies. Thirty-three percent (33%) of the examples fell into this primary category (Table 3.4).

Each of the four primary interdependency types could be further characterized into a set of secondary types. In some cases, the secondary types are further broken down into clusters of tertiary types as well. Product Definition and Architecture Interdependencies had two distinct secondary types: translation interdependencies and component interdependencies. These clusters and their corresponding sub-clusters are depicted in a detailed KJ diagram (Figure 3.2a) and described below.

Secondary and Tertiary Types

Translation Interdependencies. Product Definition and Architecture Interdependencies were labeled translation when they involved converting customer requirements in the form of statements of need into the logic of software. For example, at Lucent customer representatives and systems engineers work with a telecommunications service provider (the customer) to identify what kind of switching product the customer needs. Systems engineers then take that information and, as one team member described it, "translate it into engineering and [telecommunications software] terms" in the form of a functional specification document (FSD). The purpose of the FSD is to "understand and document what we are building ... to reach an understanding with the customer." The specification then forms the basis for a set of designs, which lay out the implementation solutions not only for the product but also

for requirements tests, which verify that the product operates as the customer requires it to. Finally, each of the designs is implemented in the form of a software product or test.

Team members at Microsoft follow a similar translation process, although in this setting the specification task is much more fluid and iterative, and detailed design stages are often skipped entirely. Nor is the translation process limited to the beginning of the development cycle. Engineers also need to translate problems into coded solutions, as indicated in the following statement by a tester at Microsoft:

> Failure conditions often occur far away from where the real problem actually is in the code. It can take months to pin down.

The examples revealed several significant characteristics of translation interdependencies. First, the output of each task is usually some form of documentation in either a written format (design documents, which are written in English or pseudo code), diagram format (product schematics), or a product artifact (software code or tests), and these outputs form the basis for subsequent tasks. Yet the relationships are not strict hand-offs; one task does not necessarily end before another starts.

For example, at Microsoft, writing the functional specification depends on information about the customer in the product vision, "which is a variable over time." Similarly, as things progress during coding, developers often discover many issues not considered in the original design. Designs, therefore, continue to evolve with the actual coded product. As a Lucent developer described it:

> We need to define the components and interfaces in the design documents. Defining things clearly doesn't mean we are not going to change it, because you always change it. But it has to be clear now so we know it changes.

FIGURE 3.2a

DETAILED KJ DIAGRAM: Product Definition & Architecture Interdependencies

Translation Interdependencies

customer requirements into physical and logical technology

Developers depend on marketing for the product vision [in order to write the spec]

Developers depend on system engineers for customer requirements. We also provide feedback on their technical feasibility with respect to cost and maintenance

We take requirements and specs and start to identify component and subsystem areas in high level design ... Then we translate that high level design into the actual C logic and functions [in low level design]

Testing depends on development to tell them how code should behave ... so they can specify and write requirements tests

problems into solutions

Failures often occur far away from where the problem actually is in the code

Developing fixes depends on first testing the code to identify problems and bugs

Component Interdependencies

Binary

interfacing components

Components communicate through formalized interfaces. There's a give and take with modification going in both directions because bugs often occur at the interaction point and because interfaces grow and change over time

The terminating component interfaces with the originating component due to the natural progress of call sequencing

base components

There is an innate dependency between the diagram engine and the schema and query designers because the former is the base

There are [component] tools for checking the integrity of the product that must be done early and reliably

circular components

Printing and setup have a circular dependency. Printing can't be done until setup is completed, but the whole point of setup is to have a final version of all the files

The X component depends on functionality in the Y component and vice versa. They use each other's technology

Patterns

core components

Changing one line of code in the cache manager impacts all parts of the system

Certain pieces are used by everyone such as authentification of the handset. If that piece changes, [every piece] will be impacted

chained components

One developer who writes sloppy code can break the product and impact the rest of the team... Nobody can get any work done

My software depends on another piece of software. That code may be dependent on yet another piece, and that chain goes on and on

hidden components

The real problems occur with the hidden interdependencies- the ones that no one thought about that pop up at the last minute, like C2 certification

Thus, interdependent translation tasks overlap and continue in parallel, to some degree, and documents and diagrams are as much outputs of the overall translation process as inputs to subsequent tasks.

A second point revealed in the examples is that translation tasks often depend not just on previous activities but also on subsequent ones. For example, a Lucent team member described how "design and development depend upon system engineers for the customer requirements, but developers [who perform design and development] also provide feedback on the technical feasibility of requirements with respect to cost and maintenance." Here, writing requirements is not only an input to subsequent design and development tasks but also depends on the knowledge of people performing those tasks.

Finally, depending on the complexity of the technology, translation tasks may be refined into interdependent sub-tasks. For example, at Lucent the design task is broken down into two sub-tasks, called high-level design and low-level design.

Component Interdependencies. As noted earlier, complex technical products exhibit what is known as a hierarchical or "parts within parts" structure (Simon, December 1962). The product is composed of inter-related components and subsystems, each of which is in turn hierarchical down to some lowest level of elementarity.[14] Because the product functionality and features are allocated across these pieces, the end result is a large number of technical parts that can interact in often non-trivial ways. Dividing the product up into multiple components also necessitates

[14] Note that where we leave off partitioning and what constitutes an elementary element are both somewhat arbitrary. For example, if we were to apply a telescope to a particular component it would reveal yet more levels of complexity. In other words, each component of a subsystem can also be viewed as a system, which comprises sub components whose relation to each other is defined by a system-like architecture.

new tasks, namely the components need to be re-integrated into one working product. Interdependencies created by allocating product functionality among components were classified as component interdependencies.

As shown in Figure 3.2a, dividing functionality among components creates six tertiary types of component relationships. Some components simply share an interface, meaning that one piece is adjacent to another. A variation of this type is a base interdependency where one component forms the base of the product and must be developed first.[15] In the case of software products, software tools (pieces of software needed to write software) fall in the base category. Circular component relationships (also referred to as two-way or cross interdependencies by the interview subjects) arise when two or more components share functionality in a reciprocal fashion.

For example, the setup component performs the installation of all the files in a product. It determines what files get requested and how and where they end up and is a vehicle for pulling different components together. As numerous people on the Desk team described it, "Setup is one of the biggest interdependencies for other components. No one can test without setup or integrate without setup, yet it is very difficult to develop setup when the [other] underlying files and feature sets are changing." Data team members described a similar type of "chicken-and-egg" relationship between the shell component (a harness other components plug into and gain value from; it provides certain unifying technology such as menus, toolbars, common error messaging, etc.) and other feature components:

---

[15] Interfacing and base component dependencies are both forms of sequential interdependence (Thompson, 1967). I make this connection later on but argue that it is useful to distinguish the two insofar as base dependencies may imply a level of reliability not (necessarily) needed in the other case. Furthermore, interfacing components can often be developed in parallel whereas base components must be completed in sequence.

> The components depend on the shell component so the shell should come first. But there is calling back and forth between the shell and components, and we need to get new features into the shell, so the shell should also come last.

The data also suggested that component interdependencies are not merely binary relationships; some of the most complicated scenarios arose when technical interdependencies existed across three or more components. For example, on Network, three components shared a set of circular interdependencies that "ran very deep," according to the interview subjects. One component supplied a structured storage space for another; it in turn depended on the other for some of its interfaces, as did a third. As a result of this complicated relationship, people working on these pieces met twice a week to go over the interdependencies and resolve blocking issues.

Another multiple component pattern observed in the data involved a core component, such as the cache manager, that interacted with many other parts of the product. Changing the design or code in that core piece, therefore, set off a "ripple" or "cascade" of changes elsewhere.[16] Developers also talked about how certain compiling and testing tasks depended upon chains of integrated components being functional.

Not all component interdependencies were easily identifiable by looking at the product architecture diagram, however. A final class of component relationships, labeled "hidden" by the interview subjects, referred to interdependencies that cut across the physical structure and/or were more emergent as designs and technical relationships changed during implementation.

---

[15] Note that this interdependency can be viewed as a collection of sequential relationships. Later I discuss why it is useful and important to sometimes take a more collective viewpoint.

In summary, Product Definition and Architecture Interdependencies can be broken down into two sub-types: translation interdependencies and component interdependencies. Translation interdependencies arise due to the need to translate between the world of the customer and the world of software engineering. Breaking the product down into a set of components necessitates a refinement of certain translation tasks (e.g., 'develop product code' becomes 'develop code for component one,' 'develop code for component two,' etc.) as well as new tasks (e.g., component integration), each of which must be re-integrated with the existing task set. These types were further shown to vary in terms of their structure (one-way, two-way, and patterned relations) and predictability.

Linkage to the Literature

Product Definition and Architecture Interdependencies most resemble the types of task relationships described in information processing and resource-based theories of product development. Adherents to the information processing viewpoint argue that the nature of the technology (task)[17] dictates different degrees of contingency among tasks and hence creates the need for information processing (Allen, 1977; Galbraith, 1973; Thompson, 1967; Tushman, 1978). Resource-based interpretations focus on the flows of skill and knowledge necessary to perform complex activities (Iansiti & Clark, 1994; Piore, et al., March 25, 1997; Wheelwright & Clark, 1992).

For example, Woodward focused on technical complexity, defined as the extent of mechanization and predictability in a manufacturing process, and found that mass production companies tended to be highly differentiated, with extensive formalization and little delegation of authority, whereas small batch and continuous process firms

[17] Defined broadly as the actions, knowledge, and techniques organizations employ to produce their products and services (Galbraith, 1973).

were structured more loosely (Woodward, 1965). Perrow examined technology along two dimensions, task variability and problem solving analyzability, and argued that organizational control and coordination efforts should be based on the degree of routineness created by the interaction of these two variables (Perrow, 1967). Thompson developed a typology based on the type of tasks performed in an organization (mediating, long-linking, and intensive) and proposed that each technology creates a type of interdependency (pooled, sequential, reciprocal), which in turn requires a certain kind of coordination and structural arrangement (Thompson, 1967).

New product development research builds upon this early work. For example, Dougherty found that one of the biggest barriers to making commercially viable new products is the linking or integrating of technical and market knowledge across disparate thought worlds (Dougherty, May 1992). Distributed information and knowledge therefore create interdependencies based on the need for people to communicate during product design (Smith & Eppinger, December 1994; Wagerman, 1995). Visual representations such as design sketches and product schematics (which played a prominent role in many of the interdependency examples described under the translation heading) act as a means of organizing the design process by facilitating discussion and consultation among people at various stages (Henderson, Autumn 1991).

Communication and interaction patterns also arise from the product architecture (Allen, November 1986; Henderson & Clark, 1990). Complex interfaces among the different product parts necessitate continuous communication among developers because interfaces can evolve during implementation of the design (Cusumano & Selby, 1995). Ulrich and Eppinger distinguish between those interactions that correspond to lines connecting physical parts on a product schematic and those that arise incidentally

due to a particular physical allocation and structure, concepts similar to the hidden tertiary cluster uncovered in this analysis (Ulrich & Eppinger, 1995).

Breaking the product into components also leads to interdependencies of actions such that team members must perform certain acts in order for other members to do their work (Wagerman, 1995). For example, changing functionality in one component often requires making corresponding changes in other parts of the system; certain tests cannot be run until all of the components are integrated together.

System of Use Interdependencies

Overview

Products must eventually operate in a heterogeneous real world environment, which consists of both other technologies and product users with distinct preferences and habitual ways of behaving. For example, in order to function effectively, software products need to be compatible with other hardware and software layered above and below them as well as with peripheral hardware such as printers, device drivers, phones, and circuit boards. Such relationships are partly due to the larger technical system architecture in which products reside but can also be an important source of strategic advantage due to network externalities (Christensen & Rosenbloom, 1995; Cusumano, Mylonadis, & Rosenbloom, 1997).

System of Use Interdependencies arise in the form of more refined testing tasks such as stress, user interface, and performance testing and in the need to develop product documentation and customized products. The fact that products exist in technical and user environments which evolve over time also creates a set of interdependencies with past product versions and necessitates tasks such as maintenance and configuration management. Figure 3.2b is a detailed KJ diagram depicting examples of the secondary

and tertiary types uncovered within this category. System of Use Interdependencies accounted for 20% of the examples (Table 3.4)

Secondary and Tertiary Types

Technical Compatibility Interdependencies. System of Use Interdependencies created because the software product exists in a larger system architecture of other hardware and software products were labeled technical compatibility. Figures 3.3 and 3.4 show the technical architecture of a PC software and telecommunications system, respectively. Note the inherent "nested" nature of such systems (Christensen & Rosenbloom, 1995). Products that were previously shown to have a complex internal definition and structure also act like components in systems at higher and lower levels. Thus, an operating system like Network must be compatible with the hardware platform below it and with PC applications like Desk or Data, which call functions in the operating system layer. Telecommunications products, likewise, exist within a layered architecture of call processing software, operating software, and hardware. These technical relationships have implications at the task level.

FIGURE 3.3
PC SOFTWARE SYSTEM ARCHITECTURE

FEATURE, FUNCTION, & CAPABILITY LAYER (software used to perform particular tasks or activities)	**Application products**
SYSTEM OPERATIONS LAYER (software that controls operations of the technical system)	**Operating System products**
KERNEL OR BASE LAYER (software that interfaces between other software and hardware)	**Operating System and Networking products**
HARDWARE LAYER	**Intel, MIPS, Alpha, and Power PC Chips Printers and Peripheral Devices**

FIGURE 3.4
TELECOMMUNICATIONS SYSTEM ARCHITECTURE

FEATURE, FUNCTION, & CAPABILITY LAYER (software used to perform particular tasks or activities)	**Call Processing Products**
SYSTEM OPERATIONS LAYER (software that controls operations of the technical system)	**System Maintenance and Integrity Products**
KERNEL OR BASE LAYER (software that interfaces between other software and hardware)	**Operating System for Distributed Switching**
HARDWARE LAYER	**3B20, 3B21 processors Wireless phones and periphals**

For example, Network runs on four hardware platforms (Intel, Mips, Alpha and Power PC) and supports hundreds of applications. This creates the need to test multiple interfaces:

> The testing matrix is just horrendous. To do a full test run of Tier 1 applications on one platform alone can take two weeks. And each platform may have its own issues and require its own changes.

One Network team member described how a single code change could easily expand into many tasks due to these interdependencies:

> Changing one line of code may require hundreds of tests on different machines and applications. Maybe 100 applications across 4 platforms. So delivery is gated by the need to test and be compatible with [all those combinations].

FIGURE 3.2b

DETAILED KJ DIAGRAM: System of Use Interdependencies

Technical Compatibility Interdependencies

software system architecture

software compatibility

Desk has a dependency on the base operating system. As the operating system changes, it can introduce new bugs or change the status of fixed bugs. That adds cycles to our fix and regress processes

A key to Network's success in the market is getting applications to run well on Network. That means that we need to tell apps how to test on Network and work with them to resolve issues

The software program continually accesses the database to see what it should be doing... Changes in code lead to changes in data, sort of like a tail

hardware compatibility

Network runs on 4 hardware platforms, so the testing matrix is just horrendous. Changing one line of code may require hundreds of tests on different on different machines and applications. To do a full test run of Tier 1 apps on one platform can take 2 weeks

Testing the code depends on having access to the actual hardware mobile phone unit

backward compatibility

The Backward Compatibility Rule says that who ever is out the door first gets supported by followers. So if we find a bug in a product that has already shipped, we have to work around it. That can cause some hair raising issues!

User Compatibility Interdependencies

documentation/assistance

Development of a document is a very linear process. There are a series of specialized tasks like write, edit, proof etc. that are performed by individuals who all work on the same piece along the way to its finished state

When developers change code it means that changes need to be made in all forms of the documentation. But the CD can usually accept changes after the book goes to print

When the development schedule slips or they make schedule swaps ... certain help topics may not get done

Creating documentation depends on the development of certain authoring and database tools

Documentation depends on knowing the product spec

testing compatibility

User interface testing is responsible for the look and feel of the product. It depends on knowing the model or spec in which the code will be used. If someone changes the spec at the last minute, we need to know about it

Customers require the phone system to be extremely fault tolerant, so we hae to perform stability tests which are basically 12-15 hours of testing under heavy calling conditions

geographic compatibility

One set of files may get distributed to [20 translators in different countries], so you can get cascading changes. Each change is multiplied by 20 and all will have unique problems

backward compatibility

If we add functionality in patches we end up with 'this only runs on release 3.0 with patch #7'

Similarly, telecommunications software needs to operate with certain hardware peripherals. In the case of Handphone, a key piece of interfacing hardware was the mobile phone:

> Development and testing depend on having access to the mobile phone. Writing code is not the big deal. [Coordinating development and] testing with equipment is the gating factor.... It takes a long time to run through all the cases... and you can end up taking multiple steps backward ... when code changes.

Code changes can also break functionality in other related products due to calling relationships between system layers. For example, two Desk team members recalled the impact of a Network code change:

> One directory in the Network registry that stored a lot of stuff wasn't protected. When Network protected it, the applications broke because we were assuming we had access to it.
>
> It introduced new bugs and changed the status of fixed bugs, which added cycles to the bug fixing and regression testing processes.

Although product teams usually try to specify such interfaces in advance, the implementation process inevitable reveals errors of both commission and omission, as noted by one developer at Microsoft:

> [Interfaces with applications] are sometimes not discovered until during implementation. For example, the developer working on backup will realize that in order to backup structured storage in a linear format, he has to provide an interface.

Two interesting aspects of technical compatibility interdependencies are evident in many of the examples. First, interdependency among products (and therefore tasks) was not strictly driven by the existence of a technical interface; product strategy also

played a role. For example, Network has a technical interface with hardware below it, but the decision to be compatible with four different hardware platforms was a strategic choice made by the team leaders. Similarly, Network can operate without Desk, and bugs in Desk code will (usually) not break Network functionality, yet Network is strategically dependent on Desk, as explained by one team member:

> Strategically, it is really important for Network to support Desk. A key to Network's success in the market is getting applications to run well on Network. That means that [we] need to tell [people in Desk] how to test on Network, motivate them to do so, and work with them to resolve issues.

Second, there is a strong temporal theme running through many of the examples, suggesting that technical compatibility interdependencies evolve over time. Earlier we saw instances in which mid-cycle implementation changes altered the system of use environment. Products also enter the market at different times, creating precedents that extend longitudinally:

> The 'backward compatibility rule' says that who ever is out the door first gets supported by followers. So if Desk finds a bug in Network after Network has shipped, we have to work around it. That can create some hair-raising issues!

User Compatibility Interdependencies. Products also need to be compatible with particular aspects of their use environment. For example, customers or users may require a certain quality or performance level or a particular user interface. This necessitates performing more refined testing tasks such as stress and user interface testing, as in the following example:

> One feature of Desk is an etcher that provides the means to draw free forms on the screen. User interface testing depends on the specification because we are responsible for the look and feel of that feature. User

> interface testers also don't write code, so we depend on other testers [to write the tests].

Most customers also require some level of assistance or documentation about how to use a product.[18] In PC software products, documentation takes the form of printed or compact disc (CD) manuals or on-line help features. Thus, an additional set of tasks arises related to developing those documentation and assistance artifacts. Many of these tasks are themselves interdependent, and each must be integrated with existing tasks such as specification, development, and testing.

Producing documentation can be broken down into a series of sub-tasks which are sequentially interdependent. For example, one starts by sketching out a design of the document (what types of information will be provided and how it will be organized into chapters and sections, what figures and diagrams will be used, the font, etc.), followed by the actual writing, editing, and proofing tasks. Eventually, the document is printed as a paper manual or copied onto CDs.

Yet because the documentation artifacts must reflect the software code, there is also an interdependency between the specification and writing of code and the specification and writing of the documentation:

> Documentation depends on program managers, developers, and testers for change reporting of [the specification and] software files.
>
> Development is very problematic for us. When their schedule slips, we are usually not taken into consideration. When they make schedule swaps, it can affect us. For example, certain help topics may not get done.

---

[18] Customer service and support activities such as field representatives and telephone help lines might also be included in the user compatibility category. They do not play a prominent role here primarily because the interview sample did not include people performing such tasks.

Writing documentation also requires the development of certain authoring and database tools, pieces of software that are usually written by developers.

The examples suggested several reasons why coordinating the development-documentation interdependency is often problematic. First, as was also the case in hardware testing, each code change necessitates making many changes to different forms of documentation (e.g., manuals, CDs, on-line features). This can lead to inconsistencies, as in the following example:

> The Desk manual had already gone to print. We had to pull a topic from the manual because a feature didn't work the way it was described, but the people working on the CD version were able to respond and correct the change.

Second, documentation tasks usually face certain constraints imposed by downstream activities that are less of a factor in code development. For example, it is usually necessary to fix certain things in the documentation like index formatting, page numbers, and page breaks relatively early in the process in order to coordinate with printing and manufacturing (shrink wrapping for software products). Developers face no such constraint, however, and often make code changes which require adjusting these elements. Thus "a lot of changes that are significant from a documentation perspective are considered so trivial that they don't even show up to developers."

Finally, software code and documentation are literally written in different languages (programming languages such as COBOL or C++ and the language of the customer, respectively). As a result, developers and documenters tend to partition their tasks differently, rendering their subsequent integration difficult:

> Documenters write about features from a user's perspective, how a user sees it. Developers work on features in terms of pieces of a code module. So a developer may partition the coding of a toolbar across several milestones, meaning that it can't be documented until the very end of the cycle.

A final tertiary-level cluster under the user compatibility heading reflects the need to tailor the software product for different customer or geographic markets. Like integrating development and documentation, coordinating different versions of a software product is complicated due to expanding (and inconsistent) change implications across different versions, unequal downstream constraints, and different partitioning schemes.

For example, Microsoft translates the English version of its Desk product into approximately twenty foreign languages, a process referred to within the company as localization. Each change to the English software code or document, therefore, explodes into a set of changes across multiple product versions:

> During localization, one set of files may get distributed to twenty people [in different countries.] So you can get cascading changes. Each change is multiplied by twenty, and all twenty will have unique problems with those files and changes.

The Tollphone project at Lucent likewise produces software products for different geographic markets. Because telecommunications products are usually customized to specific user requirements and settings, different versions of the product can end up having slightly different features and fixes, which complicates the performance of testing tasks:

> There are different streams [versions of the software for particular customers and countries]. Each potentially has its own unique set of

> software. So you can go into the lab and not be able to find a bug or not get the reaction you expect because you're running the wrong software.

Finally, like the cluster of technical compatibility interdependencies, user compatibility interdependencies exist within a temporal frame. Because customer needs evolve over time, most companies produce a series of product version releases over time and may need (or choose) to maintain forward compatibility (in which past features and functions are migrated or replicated in current product versions) or backward compatibility (in which new features must be consistent with past functionality and user routines). This is reflected at the task level in, among other things, the addition of new tasks such as maintenance of products in the field, configuration management, and mapping of changes forward and backward in time (for examples of the latter from the Autophone project, see Figure 3.2b).

In summary, System of Use Interdependencies can be broken down into two major sub-types, technical compatibility interdependencies and user compatibility interdependencies, corresponding to the external technical and user environment, respectively. Certain themes carry through from the results reported under the Product Definition and Architecture heading. In particular, the examples illustrated how the addition of new types of interdependency both refined existing tasks (e.g., testing broke down into stress, user interface, performance, etc.) and necessitated new ones (e.g., documentation, maintenance, configuration management). We also saw evidence of similar interdependency task structures (e.g., sequential structures in documentation, cascade patterns set off by changing a core component or product version) and a strong dynamic (temporal) effect.

Linkage to the Literature

The idea that products are systems comprised of components which relate to each other in a designed architecture, is an established concept in studies of innovation (Alexander, 1964; Simon, December 1962). More recently, some scholars have pointed out that an end-product can also be viewed as a component, which relates to other products within an architecture defined by users and competitors. Christensen and Rosenbloom label such nested commercial systems a value network and provide illustrations within the context of management information systems (Christensen & Rosenbloom, 1995). Cusumano and Selby discuss the implications of similar networks of products for competition and strategy in the PC software industry (Cusumano & Selby, 1995).

These authors and others point out that such networks are not static structures but can in fact be highly dynamic. For example, the work on technological history and the interpretive analyses of Rosenberg (Rosenberg, 1982), Clark (Clark, 1985) and Cusumano et al. (Cusumano, et al., 1997) illustrate how technical designs tend to build upon one another and influence the direction of industry evolution. The present analysis complements this work by describing the impact of design hierarchies and path dependency on task coordination for a given project.

Research demonstrating the importance of achieving customer compatibility, and the role of customers and users in the innovation process, likewise has a long history. Some authors discuss the importance of effective strategic management of a core product and its derivative products (Clark, Chew, & Fujimoto, 1987; Clark & Fujimoto, 1991; Wheelwright & Clark, 1992). Other work focuses on the interface between development and marketing (Griffin & Hauser, Winter 1993) or manufacturing (Adler,

March-April 1995) functions. Coordinating development with documentation has received less attention, but studies do exist (Sitkin, 1996).

Finally, the special requirements of interdependency management in high risk industries such as nuclear power (Perrow, 1984) and navigation (Hutchins, 1990; Perrow, 1984; Weick & Roberts, 1993) is receiving a great deal of attention.

Technology Sharing Interdependencies

Overview

More and more firms today are grappling with the challenge of managing concurrent, overlapping, or consecutive project efforts in which the products constitute some type of product platform or family (Meyer & Lehnerd, 1997; Meyer & Utterback, Spring 1993; Nobeoka & Cusumano, November 1995). One result of this trend is that product teams face a dual challenge. Not only must they coordinate their own project's activities; often those activities need to collaborate and coordinate with bundles of semi-autonomous but interdependent activities in other projects within the firm.

Technology Sharing Interdependencies arise when two or more projects are trying to design, develop, or use common features, components or subsystems (20% of the examples, see Table 3.4). Such sharing creates a complicated set of task relationships such that the specification, design, development, testing, and documenting of the shared piece for one project become interdependent with the same tasks on another project. Allocating components across people and projects, therefore, adds a work flow dimension to previous task relationships and necessitates new activities such as project management and scheduling. As shown in the detailed KJ diagram presented in Figure 3.2c, sharing technologies yielded two secondary clusters of interdependency, labeled direct and indirect, and discussed below.

FIGURE 3.2c

DETAILED KJ DIAGRAM: Technology Sharing Interdependencies

Direct Sharing Interdependencies

shared component access/use

One type of dependency is on a stream of software code you must integrate with. For example, projects copy and share code in libraries, common header files are shared among projects

Another project put a brand new chunk of software in that changed a very central file everyone depended on. It caused our product to break disastrously

dedicated shared component team

Some components ship with a lot of products. For example, about 20 applications in Microsoft use OLE. The customer or end user needs are not consistent. That produces a layer of friction, and [groups need to work together to resolve conflicts about interfaces or bug fixing priority]

The X component hosts into several different products including Data97, [and 3 others] Products using X include Data97, Desk 95 and 96 and Y. They have conflicting needs and schedules and a different set of interfaces

shared component design/development

[Two operating systems] used to have separate help engines. Then one team developed a help engine for both products, but the quality and feature considerations for the two platforms are totally different, so [Nework] has to modify it slightly

Indirect Sharing Interdependencies

product releasing

At one point, [two products] shared the same ship date. One team was also responsible for delivering and testing OLE in both products. If you have to ship two key products with the same technology the same week, it's going to get a little hairy

It was a morale hit for the groups that depended on X when they announced the schedule slip. They worked very hard and did their part, but the outcome was affected by third parties

Say you have a situation where there's a component with several arrows pointing into it. If that component slips, it's not just a question of doing without one thing. Now we may have to do without 5 things. It means you take less risk in the feature set [and schedule

testing and documenting

Contention between [applications] affects documentation. We wait due to the uncertainty or end up doing needless work because inconsistencies show up in the documents

Shared features create dual and overlapping testing responsibilities because bugs often exist at the intersection of several pieces of code. Is the bug in OLE code [or one of the applications]? Probably somewhere in the middle

Secondary and Tertiary Types

Direct Technology Sharing Interdependencies. In firms which produce multiple products, different software projects usually share access to certain common modules or files such as libraries or header records. As noted by a Network developer, "This creates a direct dependency from a coding sense. People depend on header files or libraries in one project in order to complete a compile. Or one project needs a library built on another project."

This type of "common file touching" is extremely high in some projects at Lucent, where the code has not been completely modularized into independent objects. As a result, seemingly minor code changes on one project routinely break functionality elsewhere, even on seemingly unrelated projects:

> Another project put a brand new chunk of software in that changed a very central file everyone was dependent on. It caused Handphone to break disastrously and took us one week to recover.
>
> The first time we submitted code, it immediately broke several other ... projects. In fact, five Handphone developers all broke other projects in different places. It had a global impact and stopped everything.
>
> A recent Tollphone product was an enhancement for automatic collect calling ... It was originally thought to be a relatively simple feature with a narrow interface ... Then we discovered that we had to add data to a file. The file the developer touched was sequential, not modular, and a developer working on an Australian wireless product touched the same file ... We had a whole bunch of experts trying to break the dependency, but we were unable to do it. That was one line of changed code, but we had to pull in all the Australian development for the Tollphone integration. What should have been a five subsystem product became twenty nine subsystems.

Another cluster of examples involved two or more projects collaborating over the design and/or development of a common feature. Sometimes the shared feature

existed previously in one or both of the products; other times it represented a brand new feature. For example, Network and another major operating system at Microsoft share certain features such as the Software Development Kit, Win Help, TCP/IP (a telecommunications protocol), and the printing interface. One team is usually primarily responsible for specifying and designing the feature, but they incorporate comments and feedback from developers and testers on the other team. The sharing is complicated, however, because the quality and feature considerations for the two products are completely different. This creates conflicts between the teams and often adds cycles to existing development tasks, as shown in the following example:

> Network is porting the [other operating system's] printing interface as a baseline, but we will add some advanced capabilities to it in order to support extensions that Network customers demand.

Several examples showed how the existence of separate products with different historical code bases and (perceived) users complicates the coordination of tasks (for developing both code and documentation):

> [Application 1] and [application 2] have different historical code bases, which complicates feature sharing. For example, ghosting is a feature that takes the user to a place in the product when they perform a search. If that place is not on the screen, [the applications] implement different solutions. ... Whose version should be the shared one?
>
> The old Desk document consisted of 4 or 5 books [one for each application]. The new model is one [shared] manual. The document groups therefore had to agree on goals, processes and tools. Data resisted the less information model because their customers are power users who can't get enough information. [Another application team] argued that a big window with lots of information covered up the toolbar, which their customers use a lot.

Sharing features can also create dual or overlapping testing responsibilities because bugs often exist at the intersection of two or more pieces of code. In the case of

Desk, this led to "bugs shuttling back and forth across the groups ... A bug originally reported by [one application] was assigned to [another] and then passed back to [the first]." One Desk tester labeled that bouncing "a symptom of our interdependence." At Lucent, a Handphone tester described the testing implications of shared subsystems in a similar manner:

> At the interface, it is very hard to see whose problem it is. I do as much as I can on my side to identify if it is my problem, but then I need the other side to do things. We work back and forth until we narrow it down and decide who should fix it.

Teams sharing components also need to coordinate regarding the prioritization of bug fixing.

Components can also be shared across versions of the same product. In this case, the users tend to be the same, but the scheduling of tasks can be problematic:

> [One Desk application] currently has a feature which enables users to type words within a box. Now [we are] replacing that drawn object code with [new] code. The next version of [the application] will include a new feature to link boxes. The developer working on the linking code needs to know 'what can I touch'?
>
> A new feature is going into the next release of Network. It depends on functionality that was originally not scheduled until very late in the cycle of the current release.

The schedule coordination task at Microsoft was usually assumed by a lead program manager. This person was also responsible for assuring design consistency across the components being supplied by different groups:

> As program manager, I need to coordinate the actual delivery of all the tools and features or files that go in the Desk box. We need certain pieces by a certain date ... and they all need to get to the same place at the same

> time.... There are a whole set of design interdependencies in Desk. We need to achieve consistency, a single vision or user interface across all of these components and groups. ... [Application 1] might have [component] version 1.0 but [application 2] uses 2.0. This creates a complication because if [application 2] is installed after [application 1], [application 1] might not work.

One solution companies frequently adopt for managing widely shared technologies is to assign a dedicated component team.[19] For example, about twenty application products in Microsoft depend on a dedicated team for the one component. At Lucent, certain key subsystems such as Recent Change/Data and Peripheral Control are produced in departments that support multiple projects simultaneously. Under these circumstances, shared technical interdependencies involve not just two teams but many, and the structure of the tasks is mutual or pooled, not reciprocal.

For example, Data faced the difficult situation of being both a heavy component user and a component supplier; it depended on four external teams for critical component technology and was a component supplier to Desk because it shipped as part of the Desk box. The multiple task interdependencies across components and teams rendered integrating the product a major challenge:

> The X component depends on the Y component for some things and vice versa. They use each others technology, so they need to synchronize [integration] to get a running and testable exe. Add five components, two shells, and six teams and watch the debates about how to make that happen.
>
> The Y component hosts into several products including Data, Desk, [and two other products]. They all have conflicting client needs and schedules. Each has a different set of interfaces and tends to apply different requirements to the component because they have different end users. Or

[19] This sub-type is also discussed under the Shared Resource heading because it yields task interdependencies created by limited time and attention on the component team.

> one wants something now but we can't deliver because of [other] demands to be met.

On the Desk project at Microsoft, which had about forty components being developed elsewhere in the company, one developer defined a component supplier as "a dependency created because some line of code is being written by someone not sitting in this hall." An example was the setup component: "Every day, one guy in [application 1] runs the setup software, notes the problems and talks in detail with the developer supplying it."

Because components are arranged hierarchically, developers and testers working on one component can find themselves needing to interact with their counterparts working on other components only indirectly related to their own, as evidenced in the following example:

> [Application 1] is dependent on X. X depends on [another component]. Therefore [application 1] is dependent on [the component]. We don't even know these people but we need their help in finding bugs.

Indirect Technology Sharing Interdependencies. The above section described how the desire to share certain technologies can create linkages among the design, development, testing, and documentation tasks across different projects in a company. Recall that the code design and development tasks and the documentation tasks are also interdependent (System of Use). Sharing technologies, therefore, also has indirect implications for the interdependence between development and documentation as well as between development and product release, both of which are described below.

Contention about the design and development of shared features can result in increased blocking and waiting times for certain documentation tasks. Writing about a

particular feature may be delayed while designers from different projects try to resolve their disagreements. This was most notable on the Desk project, as described by one documenter:

> Contention between [two applications] affects [us]. We wait due to the uncertainty or end up doing needless work.

This "needless work" appeared in a variety of ways. For example, documenters described how their tasks tended to be characterized by high degrees of rework because feature sharing decisions were often reversed or altered mid-cycle. In other examples, documenters faced the problem of how to write about a feature that worked differently across the application products. Inconsistent functionality also showed up in testing as bugs where code failed to perform as expected.

The examples also illustrated how sharing the design and development of components can have significant implications for the shipment of the product. As multiple products depend on one another (either directly or through their mutual dependence on a common component team), product gridlock often occurs. The probability of missing a shipping deadline therefore tends to increase with the level of sharing, according to the team members interviewed. This is best illustrated by an example from Data:

> One reason Data96 doesn't exist is due to interdependencies. [What is today Data97] was originally scheduled to ship in September 1996, but no one had any confidence this would happen because of all the components and shared dependencies ... Some of the teams they were relying on for components were doing work for other projects and slipping. So we moved Data to 1997.

In summary, Technology Sharing Interdependencies fall into two major secondary categories. First, sharing components across projects creates sequential, reciprocal, or mutual (in the case of a dedicated component team) interdependency among tasks across projects in the firm. These interdependencies in turn alter the relationship with certain existing interdependencies and downstream activities, a second category labeled indirect interdependencies here.

Linkage to the Literature

Although studies of new product development have traditionally focused on how to achieve speed and productivity on an individual project, the perspective of multi-project management is receiving increased attention from both managers and scholars. Researchers point out that firms gain certain strategic and productivity advantages from having consistency across products (Meyer & Lehnerd, 1997; Meyer & Utterback, Spring 1993). In particular, user interface commonality and consistency can be a source of strategic market advantage insofar as they increase the probability of "locking in" customers to a particular standard of use (Cusumano & Selby, 1995) Common or shared development can also be more efficient than having the same tasks replicated across the company and often permits task specialization (Hayes & Clark, 1985).

This literature tends to highlight two major multi-project strategies (Nobeoka & Cusumano, November 1995). First, firms can use technologies from an existing product with some modification or enhancement in a redesign of the same product. Firms can also share the technology across product lines that target different market segments, commonly referred to as an economy of scope or leverage strategy (Clark & Fujimoto, 1989).

Empirical research that systematically explores how firms can actually achieve such sharing is only now emerging. For example, Nobeoka and Cusumano discuss ways of managing evolutionary linkages among products (Nobeoka & Cusumano, November 1995). Their survey results indicated that projects using a rapid design transfer strategy are the most efficient in terms of engineering hours. Earlier, Cusumano argued that reusing existing designs in new software development without appropriate planning may have a negative impact on development productivity and quality (Cusumano, 1991). Sanderson demonstrated how Sony used product families to manage change across product generations (Sanderson & Uzumeri, 1990). Brown and Eisenhardt suggest some remedies for leverage "traps" that projects commonly fall into (Brown & Eisenhardt, May 1995).

March and Simon observed that interdependence is one of the conditions necessary for inter group conflict (March & Simon, 1958).

Resource Sharing Interdependencies

Overview

The final type of interdependency identified in the examples reflects the fact that tasks exist within a context of limited resources. The execution of various tasks can, therefore, become mutually interdependent as people vie for access to or use of physical resources like special equipment, laboratory testing facilities, or meeting rooms. Similarly, human beings have a limited resource of time and attention to devote to different tasks.

Resource Sharing Interdependencies clustered into two secondary types, capital resource sharing and time and attention resource sharing (Figure 3.2d), and accounted for slightly more than one quarter (26%) of the examples (Table 3.4).

FIGURE 3.2d

DETAILED KJ DIAGRAM: Resource Sharing Interdependencies

Shared Capital Interdependencies

equipment

The integration process is very hardware intensive, very dependent on machine time

Say I change one line of code. Then I need to compile that change. Fifteen people may be making changes and compiling in parallel. The system starts to slow down

space

We had to slip the release date because [another product] was going through manufacturing at the same time

There's contention for lab time, and it can be difficult to get the lab in the right state because so many people are changing it every day

Reviews and code inspection meetings require scheduling a room in advance. We end up meeting in the cafeteria because all the rooms are booked

Shared Time & Attention Interdependencies

across individuals

Developers depend on other developers to review code. Everyone needs the same people so authors and reviews become scarce resources

across teams/departments

A component is in the hook to deliver to Desk and Data. They are very Desk focused right now, so it's hard to get their attention for Data

One setup team was responsible for supplying Desk4, 95 and 96. We were constantly in pressure mode

The tool organization is very lean due to cutbacks. They serve a lot of products so we have to beat on them in order to get our tools

within an individual

Program managers are responsible for design issues and scheduling. We realized that the scheduling task was overwhelming their productive ability

The Desk feature leads need to coordinate with five teams. Time is a real problem

As a lead I am responsible for things like reviews and one on ones as well as answering design and technical questions and advising people about how to handle non-technical issues

Every developer and tester has to know how to bring up the lab. You can spend two hours just getting the lab in the right state and have no time to actually run the test

The integration process is very manual and requires a certain amount of close attention from the builders. But builders are also responsible for handling unanticipated problems and requests from developers

across time

Developers are sometimes pulled off current development to help with maintenance on a previous release

An Desk developer programmed the main engine for [a feature] Later there were bugs in it, but the developer had started working on 96 design. This created a lag of 1.5 - 2 months

It took forever to get version 1.0 out. When it finally shipped everyone was tired and burned out. We needed to spend time rebuilding the team

Day by day I watch them and realize that every day they delay 95 is one day less for 96

Secondary and Tertiary Types

Shared Capital Interdependencies. Shared capital interdependencies arise when two or more tasks (on the same or different projects and performed by one or more people) depend on access to a capital-intensive resource such as equipment or space. This type of interdependency arises because firms have a limited supply of capital, which must be allocated across multiple projects and needs. Thus, the more capital spent on special equipment for one project, the less money exists for equipment elsewhere in the firm; money spent on labs is not available for other types of space or equipment.

The implications of this type of sharing were particularly prominent at Lucent, which has experienced several rounds of down sizing and cost cutting over the last several years. As a result, the performance of certain key tasks was often significantly delayed or blocked on the projects.

For example, large software projects usually require special laboratory testing facilities, which simulate the actual working environment of the software product. Because these labs are typically very expensive (approximately $2-5 million each at Lucent), multiple projects share access to them. There are six testing labs at Lucent's Naperville location (the site of this study), that are widely shared across the firm by international and domestic projects as well as field support organizations. According to the company's official development process, all regression (feature) testing and some requirements testing must be performed in a lab. Testers schedule their sessions in four to six hour blocks (twenty four hours a day, seven days a week) but are often blocked from access by other projects or higher priority field bug "crises."

Gaining access is not the only problem resulting from sharing, however. In particular, different products require different lab "setups." Testers, therefore, spend a large part of their valuable session time removing previous test environments and installing the environment required for their product. If those installation procedures fail to work (a common occurrence), testing is delayed. The lab facilities also tend to be a disorganized and distracting work environment, perhaps an inevitable consequence of their widespread use. These factors, plus the existence of faulty, unreliable equipment (another consequence of sharing is that equipment tends to wear out quickly), further complicate and often delay the performance of the testing tasks, as revealed in Figure 3.2d and the following example:

> Stability testing needs a very clean system lab, a clean test bed. But it's a lot harder to have a clean test bed than a clean [real world] system because you've got people in the lab constantly changing things every day.

A more prosaic but nevertheless significant set of examples concerned gaining access to shared meeting rooms. As task interdependency increases and extends across people and projects, more meetings are inevitably required in order to coordinate the work. Meetings require rooms, however, and if the room is not available, the completion of tasks is delayed. This was a problem at both firms but particularly at Lucent where the official development process called for extensive design and code inspection reviews.[20]

Finally, the integration of large software products consisting of multiple components requires very fast (and expensive) computers. If those machines are not available or are shared by too many projects, the integration task can be blocked or significantly slowed down.

---

[20] Many of these meetings end up being held in the cafeteria.

Shared Time & Attention Interdependencies. Another very important limited resource in companies is human time and attention. People can typically only perform one task at a time and have a limited cognitive capacity to attend to multiple issues and problems (Bluedorn & Denhardt, 1988; McGrath, 1990; McGrath, Kelly, & Machatka, 1984; Zerubavel, 1981). Like some physical resources, time and attention also tend to deteriorate over time if they are not occasionally replenished or renewed. The result can be various forms of task interference, where the performance of one task occurs at the expense or to the detriment of others.

For example, more and more firms are attempting to shorten development cycles. But the more intensely people work on the final tasks for one product, the less energy (immediately) available to start working on start-up tasks for the next release:

> User interface testers often need API testers to stay on and find/fix bugs in the test application code. But the Desk API testing team is usually ramping up at that point [for the next version of Desk].
>
> A Desk developer programmed the main engine for [a feature] in Desk95. Later, there were bugs in it, but the developer had started working on 1996 designs. This created a lag of 1.5 - 2 months for documentation.

Similarly, developers who are responsible for both current implementation and maintenance can find themselves unable to focus on current tasks because of past project demands.

The Technology Sharing section described how one popular approach to sharing components assigns development of such to a dedicated component team. Multiple projects then depend on the limited time and attention of these team members,

however, as evidenced in Figure 3.2d. At Lucent, a major inter project blocking factor was common dependence on a tool support organization:

> The tool organization is very lean due to cutbacks. They serve a lot of products, so we have to beat on them in order to get our tools.

Other frequent examples in the data occurred when a single individual was assigned responsibility for more than one task. For example, on the Network project, people working on product integration were responsible for both the daily integration of files and components and the solving of unanticipated problems and requests from developers. The integration lead on the team described the effect of trying to perform these two tasks simultaneously:

> The [integration] process is very manual and hardware intensive. It requires a certain amount of time and close attention from the [people performing it]. Developers also depend on integrators to handle unanticipated requests and problems like accepting bug fixes.... We found that the [integrators] would come in at 9AM and just go crazy all day long. We used a white board and then a pad of paper to track the backlog of unanticipated requests, but they could never catch up ... and were burned out and stressed. So they never had time to actually integrate the product [or were careless and distracted when doing it].

Program managers were similarly overloaded; the task of scheduling multiple components often overwhelmed their ability to perform certain design tasks. Team leads at both companies, who must typically juggle mentoring, advising, and reviewing responsibilities, also supplied a large number of the examples under this category.

The examples suggest several interesting characteristics of Shared Resource Interdependencies. First, like Shared Technology relationships, sharing resources has important affective and work satisfaction implications. In the case of Technology

Sharing, the result was inter-group conflict. For Resource Sharing, it took the form of frustration and apathy.

For example, highly experienced engineers and designers at Lucent often spent the vast majority of their day trying to arrange rooms for meetings or negotiate access to a computer queue or laboratory. At Microsoft, as the level of component and design sharing increased, leads and program managers found themselves devoting more and more of their time to meetings and communication with their counterparts across the firm. Many of the people interviewed resented the amount of time spent coordinating with others or negotiating access to limited resources. They believed it wasn't "productive work" like writing code or documents and felt ineffective when it interfered with their ability to accomplish those tasks.

The second interesting aspect of Shared Resource Interdependencies was the fact that they often arose as almost an unintended consequence of incremental decisions made at a firm-level. A series of decisions aimed at cost cutting at Lucent resulted in minimum support staff and decentralization of tasks such as reserving rooms or scheduling meetings. At Microsoft, experienced project members described how their responsibilities had gradually expanded as the firm grew rapidly in size and altered its strategy from single products to integrated ones.

Linkage to the Literature

A large body of studies adhering to the resource dependence view indicate that frequent external communication leads to higher performing development projects by increasing the resources (e.g., budget, personnel, equipment) available to the team (Brown & Eisenhardt, April 1995). Ancona and Caldwell identified a special boundary

spanning role, ambassador, which consisted of political activities such as lobbying for support and resources (Ancona & Caldwell, 1990).

Salancik and Pfeffer distinguish between tangible resources (money, space) and intangible resources (time, social currency) and discuss the implications of resource control and interdependency for power and conflict within an organization (Salancik & Pfeffer, 1974). More recent work by Malone and Crowston builds upon the tangible/intangible distinction (Malone & Crowston, March 1994).

Perlow specifically addresses the work and individual effects of the "time famine" in product development and organizations more generally (Perlow, 1995).

3.4 Quantitative Analysis of Interdependency Types

Table 3.4 shows how the interdependency examples were distributed across the four primary categories. Most examples (33%) fell into the Product Definition and Architecture cluster, probably a reflection of the large number of developers and designers in the interview sample. Somewhat more surprising was the frequency of Shared Resource Interdependency examples (26%). Virtually every person interviewed supplied examples under this heading, although the distribution of the secondary types differed across the firms. Lucent projects tended to cite more shared capital examples. These results suggested analyzing the distribution of types along three additional dimensions: firm, project, and functional area of the interview subject.

TABLE 3.4
DISTRIBUTION OF EXAMPLES BY PRIMARY INTERDEPENDENCY TYPE

	Frequency Counts	Percentage of Total
Product Definition & Architecture Interdependencies	175	33%
System of Use Interdependencies	103	20%
Technology Sharing Interdependencies	110	21%
Resource Sharing Interdependencies	139	26%
TOTAL	527	100%

Table 3.5 presents the distribution of types by company. As revealed in the second and fourth lines of the table, the percentage of examples in the System of Use and Resource Sharing categories are approximately the same at the two sites, at about 20% and 25%-28%, respectively. In contrast, the Product Definition and Architecture and Technology Sharing categories diverge sharply. Almost one-half of the examples from projects at Lucent were Product Definition and Architecture (48%), while only 5% related to Technology Sharing. The two categories were more evenly divided at Microsoft, and the Technology Sharing category led with 34% of the examples

Several possible factors may account for this difference in results. First, the two categories may be confounded due to an inability to operationally distinguish them. This explanation does not account for the different direction of the results, however.[21] An alternative explanation reflects the firms' histories. At Lucent, products were

[21] In order to get this pattern of results, the analyst would have to confound the two categories inconsistently across the two firms.

originally designed to be sold as part of a telecommunications system "black box." Shared components and subsystems are thus almost an inevitable and implicit part of such an integral architecture. At Microsoft, the situation is reversed. Formerly independent products are now becoming more system-like by sharing components and features. Note that if we sum examples across Product Definition and Architecture and Technology Sharing, the percentages at each firm are nearly the same (55% and 53%, respectively).

TABLE 3.5
DISTRIBUTION OF INTERDEPENDENCY TYPES BY COMPANY

	Microsoft	**Lucent Technologies**
Product Definition & Architecture Interdependencies	61 (21%)	114 (48%)
System of Use Interdependencies	57 (20%)	46 (19%)
Technology Sharing Interdependencies	97 (34%)	13 (5%)
Resource Sharing Interdependencies	72 (25%)	67 (28%)
TOTAL	287 (100%)	240 (100%)

Table 3.6 shows the same distributions by project. First, note that on Network (column 2), Product Definition and Architecture account for the greatest percentage of the examples (34%). This result supports the explanation given above about the influence of a prior architecture. Network is a system-like product and thus many of the Technology Sharing relations will be implicit in its architecture. There are still a sizable number of examples in the Technology Sharing category (25%) because Network was sharing components with another operating system during the time of this study.

As indicated by a star in columns 2 and 3 of the table, the most frequently cited examples in Desk and Data fell into the Technology Sharing category. On Desk, five applications teams and a Desk team were attempting to share components and features. The level of feature and component sharing was even higher on Data and is reflected in a correspondingly higher percentage (47%).

Several cells of Table 3.6 also stand out for the surprisingly low incidence of examples (e.g., 9% System of Use on Data, 12% System of Use on Handphone and 0% Technology Sharing on Autophone). These results are most likely an artifact of the interview sample and process. For example, testers and documenters (who are most likely to cite System of Use Interdependencies, as shown below) constituted a small fraction of the interview sample on Data. The interviews were also of limited duration, and people tended to begin with and spend the most time talking about the types of interdependencies that were most prominent in their minds.

Table 3.7 shows the top citation category for five different functional areas. Developers tended to focus on the implications of Product Definition and Architecture Interdependencies. Program managers and designers, who are usually responsible for coordinating tasks, schedules, and designs across projects, cited more examples of Technology Sharing Interdependencies than any other category. Testers and documenters, whose work is closest to the boundary between the product and the external environment, yielded numerous examples of System of Use interdependence.

TABLE 3.6
DISTRIBUTION OF INTERDEPENDENCY TYPES BY PROJECT

	Network	Desk	Data	Handphone	Tollphone	Autophone
Product Definition & Architecture Interdependencies	27 (34%)*	25 (17%)	9 (16%)	48 (51%)*	29 (39%)*	37 (53%)*
System of Use Interdependencies	14 (18%)	38 (25%)	5 (9%)	11 (12%)	16 (21%)	19 (27%)
Technology Sharing Interdependencies	20 (25%)	50 (33%)*	27 (47%)*	5 (5%)	8 (11%)	0 (0%)
Resource Sharing Interdependencies	18 (23%)	38 (25%)	16 (28%)	31 (32%)	22 (29%)	14 (20%)
TOTAL	79 (100%)	151 (100%)	57 (100%)	95 (100%)	75 (100%)	70 (100%)

* Indicates top citation category in the project (column).

TABLE 3.7
TOP CITATION CATEGORIES BY FUNCTIONAL AREA OF THE INTERVIEW SUBJECT

Function	Top Citation Category
Designers	Shared Technology Interdependencies
Developers	Product Definition & Architecture Interdependencies
Program Managers	Shared Technology Interdependencies
Testers	System of Use Interdependencies
Documenters	System of Use Interdependencies

3.5 Discussion

The previous two sections contained a fairly detailed analysis of different interdependency types within the context of a particular phenomenon, large scale software development. This section steps back from that level of detail and makes some observations about what the results of this analysis tell us more generally about tasks, interdependency, and organizational structure. It begins, however, by highlighting some of the limitations of this research.

3.5.1 Limitations of the Study

It is important to stress the limitations of the approach used in this study. Because the analysis relies on self-reported data, the interdependency examples are not randomly selected, but rather reflective of the projects and people interviewed. This was particularly evident in Table 3.7, which showed how the distribution varied by functional area of the speaker. The reliance on self-reports also suggests possible biases due to human cognitive limitations. For example, it is well known that more recent events tend to be highly salient (Cook & Campbell, 1979). This may account for the

preponderance of examples having to do with shipping and releasing a product (most of the projects were at or near the end of their development cycle when the data was collected).

The classification proposed here is not intended to be exhaustive but merely illustrative of important types of interdependency. The frequencies with which different types were observed (Tables 3.4 - 3.7) may not reflect their actual representation in the companies or projects.[22] Nor are they any indication of the relative importance of different types. Certainly there are many viable schema but by identifying some common types and then exploring their implications, it is possible to begin to go beyond the previous approach of "one interdependency at a time."

The fact that we see similar types across projects and companies, and that the percentages are relatively stable once we account for differences in firms, strengthens the results and supports their internal validity. Subsequent interviews with experienced product development managers further suggested that the results have strong face validity (Judd, Smith, & Kidder, 1991). The extent to which the taxonomy is externally valid (generalizable to other companies, projects, and outside the product development domain) awaits further investigation.

Finally, although the KJ Method is an attempt to introduce some structure to qualitative analysis, it is not a science. Throughout the extensive analysis period, effort was made to continually challenge the results and verify that they were consistent with the knowledge and intuition developed through many hours of interaction with team members and study of the data. Intra rater and inter rater reliability tests support the

[22] For example, representatives from marketing and sales were under-represented in the subject pool. Although interdependencies involving marketing tasks do appear in the examples, there are undoubtedly others not captured here.

stability of the resulting taxonomy categories, but there remain questions of how reliable and repeatable the methodology is.[23]

This research, however, adds to the store of knowledge about the role of interdependency in organizations. The findings bear on several issues, including the relevant dimensions upon which we might characterize interdependencies, the implications of multiple interdependencies for organizing, and the very relationship between interdependency and organizational structure. Each of these is discussed below.

3.5.2 Additions to Theory

Characterizing Interdependency

This research sought to inductively derive a taxonomy of different types of interdependency. The analysis suggested that interdependency can be characterized along three dimensions: the primary decision driver, task structure, and predictability.

Product Definition and Architecture Interdependencies arise due to the need to translate customer needs into product functions and features in the form of components and subsystems and can be traced back to the product technology. System of Use Interdependencies are primarily driven by the external environment; the product needs to be compatible with its technical and use environments, necessitating tasks and interdependencies which span the firm's boundaries. Technology Sharing Interdependencies reflect a firm's multi-project strategy and structure, in this setting how project teams share certain components, and create work unit interdependencies within the firm. Finally, Resource Sharing Interdependencies are both an outgrowth of

[23] An interesting extension of the process would be to have team members themselves perform the clustering exercise and compare their results to the ones obtained here.

the structures and processes used to coordinate work and influenced by firm-level physical and human resource policies.[24]

This multidimensional interdependency taxonomy builds upon and integrates prior thinking about task relationships. For example, previous researchers have focused on the influence of technology (Henderson & Clark, 1990), work unit relations (Gresov, 1990), and the environment (Ancona & Caldwell, 1990) on tasks. The present framework provides a more global and unified approach to interdependency management because it explicitly considers how these different domains interact at the task level. The analysis demonstrated how the very addition of more (different) interdependencies alters the definition of the task set (both refining existing tasks and necessitating new ones), in effect shifting the basic set of things that need to be performed and coordinated, and often necessitating a reallocation of tasks. By imposing successive types of interdependency and then examining or mapping how they get translated and refined, we can begin to examine how tasks and task relationships grow and get more complicated.

This analysis also integrates other conceptualizations of interdependency and verifies some untested assumptions made by previous scholars. For example, the most ubiquitous existing interdependency framework is probably that of Thompson (Thompson, 1967).[25] Thompson explicitly wrote that the sequential, pooled, and reciprocal interdependencies he was describing were driven by the internal technology (Thompson, 1967). Since then scholars have applied the framework to describe task

---

[24] Of course, in reality these distinctions are rarely so clear and simple. For example, we saw how strategy also influenced certain System of Use relationships and how the Product Definition and Architecture and Technology Sharing categories are often very difficult to distinguish.

[25] Relative to Thompson's typology, the present framework can be criticized for its complexity. One could argue, however, that it is difficult if not impossible to make a parsimonious addition to Thompson's framework in that he definitively described the ways in which two things can be logically related.

relationships driven by the environment or strategy, but never questioned the appropriateness of such extensions. This analysis verified that Thompson's framework generalizes outside the technology domain but also suggested several possible future research questions. For example, Thompson proposed that cases of mutual interdependency are easy to coordinate (relative to more sequential and reciprocal forms) via standardization because there exists very little contingency, an assumption scholars following in this tradition rarely question. The results of this study suggest otherwise, especially when mutual dependence involves work units.

The predictability dimension integrates Thompson's work with past and more recent thinking by other authors. Thompson implicitly assumed that interdependencies are anticipatable in advance. For example, he defined sequential interdependency as existing "when direct interdependence can be pinpointed between [tasks], the order of that interdependence can be specified... and it is not symmetrical" (Thompson, 1967). March and Simon, in contrast, observed that:

> Interdependency does not by itself cause difficulty if the pattern of interdependencies is stable and fixed. In this case, each sub program can be designed to take account of all the sub programs with which it interacts. Difficulties arise only if program execution rests on contingencies that cannot be predicted perfectly in advance... We need a framework that recognizes that the set of activities to be performed is not [always] given in advance... that one of the very important processes in organizations is the elaboration of this set (March & Simon, 1958).

More recently, Karl Weick also noted that people increasingly work in a context of "randomly occurring and unpredictable" events (Weick, 1990).

A multidimensional taxonomy allows for more fine-grained consideration of appropriate coordination solutions. For example, are sequential interdependencies

driven by technology "the same as" those that are environmentally imposed? There may be greater lag effects or a higher proportion of the unpredictable sub-class in the environment domain due to greater uncertainty and more distributed tasks. Where are there opportunities (and need) to design new solutions (e.g., hidden interdependencies) or coordinate work more efficiently (e.g., patterns of similar bilateral relations)? Why and how do subjective judgments about interdependency itself vary, as suggested by the analysis in Section 3.4 as well as previous scholars (March & Simon, 1958)?

The multiple interdependency framework and perspective position us to start addressing such questions as well as the existence of essential organization design dilemmas within firms.

Multiple Interdependency Challenges

Under a multiple interdependency perspective, organization design becomes a process filled with contradictions and dilemmas (Pfeffer & Salancik, 1983). Three important dilemmas captured in this analysis are balancing the need for (1) part-whole coordination to avoid organization design conflicts, (2) planned and unplanned structure, and (3) time to coordinate and time to produce.

Looking at interdependencies unidimensionally or in isolation from one another as previous scholars have done brings with it the risks of mis-applying coordination solutions or failing to identify more efficient ways to arrange work activities. For example, researchers in new product development routinely advocate the use of multi-functional teams and "frequent and intensive communication and feedback" as a means of coordinating reciprocal interdependencies (Imai, Nonaka, & Takeuchi, 1985; Weick, 1990; Wheelwright & Clark, 1992). Although such approaches may appear adequate in terms of addressing the needs of a single type of interdependency, they appear less

ideal as the number of such interdependencies increases, and team members find themselves devoting more and more time to coordinating and communicating and less to production activities.

Managers need to impose organizational structures and processes in advance but also be responsive to changing contexts and more emergent interdependencies. The existence of "hidden" interdependencies was a constant theme in all four primary categories.

Finally, multiple interdependencies create coordination overload and inefficiencies. As interdependencies grow and spread throughout an organization, the task set expands as more tasks are added and existing tasks become more refined. More and more people (with different perspectives, priorities, and constraints) and work units become involved, and the associated coordination becomes more complex. The result can be a vicious spiral in which "the cost of increased coordination is yet further coordination" (Katz & Kahn, 1966). Or, as one interview subject observed, "the more interdependencies, the more needs to be done in person. It creates a double bind."

Decisions, Tasks, Interdependency, and Structure

Perhaps the most interesting and provocative result of this study was the questions it raises and insights it offers about the relationship between and among decisions, tasks, interdependency, and organizational structure. These concepts are both frequently confounded (e.g., equating interdependency of components with interdependency of tasks) or presented as artificially distinct (e.g., we apply structures to coordinate tasks) in the existing literature. The present analysis suggested that the relationships are in fact much more complicated.

For example, contingencies between product components may or may not be reflected in similar interdependent relationships at the task level (e.g., sequentially interfacing components lead to both sequential development tasks and reciprocal testing tasks). Assigning the components to different developers or repartitioning the task itself can also transform one type of interdependence into another.

Traditional design literature has largely depicted structure as something imposed on tasks. That is, given a certain task set, we can design an organization structure to coordinate work (Galbraith, 1973). Emerging theory suggests that the two are hard if not impossible to disentangle because the coordination solutions we impose often create tasks and interdependencies (Ancona, Kochan, Scully, Van Maanen, & Westney, 1996). The present analysis lent strong empirical support to that claim.

How do we then distinguish a task from the structure used to design and coordinate work? Indeed can we? Figure 3.5 presents a possible model for making sense of these relations. It suggests that decisions in different areas (product technology, product environment, work unit environment and strategy, and organization) result in task interdependencies. In the presence of resource scarcity, organization design conflicts arise. Projects which recognize such conflicts and employ strategic organization design principles to resolve them perform better.

3.6 Conclusion

The objective of this study was to break down the "web" of relationships among tasks in new product development projects into different types or categories of

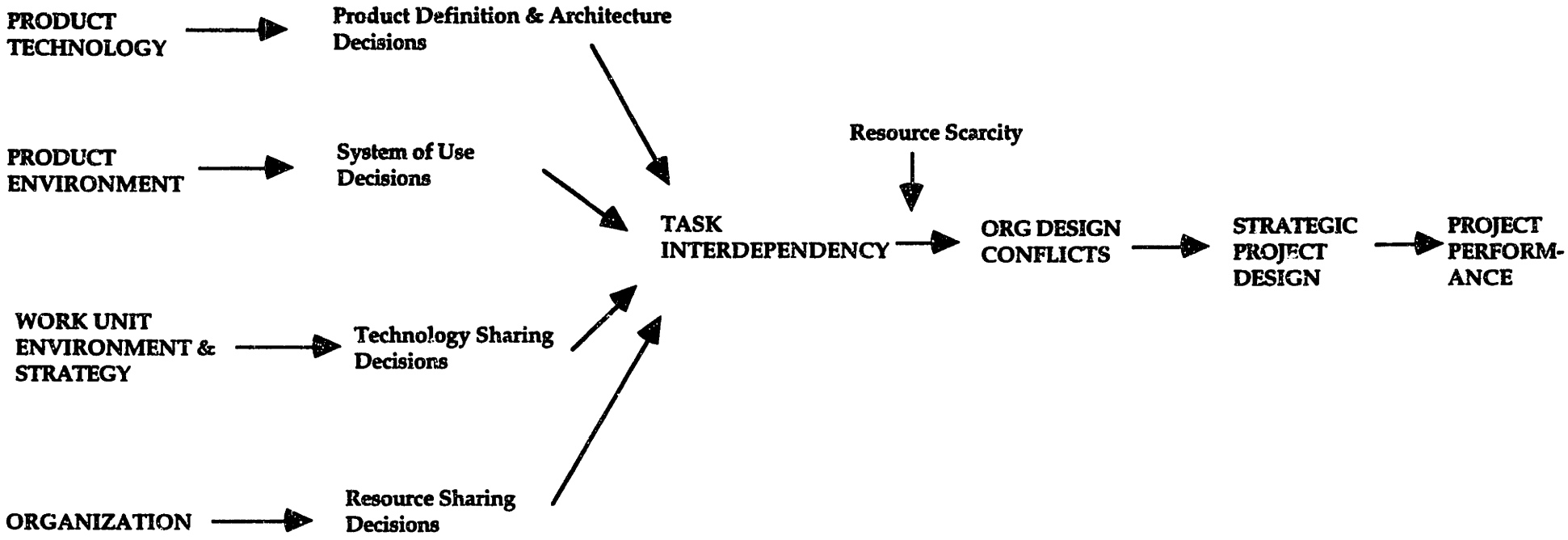
interdependency.[26] Note the inherent contradiction in that statement. Characterizing the relationship among tasks as "web-like" suggests a convoluted, inter-woven structure; the notion of there being distinct, isolatable types is in many ways the direct antithesis of a web.

Yet conceptually, this juxtaposition is quite significant. On the one hand, researchers need to come up with more sophisticated ways of depicting task relationships that reflect the level of complexity found in work such as large scale product development. We also need to understand some of the larger implications of managing multiple interdependencies simultaneously, in particular the ways in which the overall work process becomes more chaotic and unpredictable and the relationship with firm strategy and structuring over time. Yet in order to design solutions to coordinate, it is necessary to somehow break apart that complexity. At its heart, this study is about trying to understand and tease apart that duality.

[26] The metaphor of a web is particularly appropriate and illustrative of some of the gaps in our current theoretical models. For example, a web connotes the image of a spiral whose ultimate pattern no one envisioned at the start; i.e., its structure and pattern are evolving and somewhat indeterminate. Yet most theories of interdependency and organizational design are predicated on the notion that interdependence is stable and can be identified in advance. Likewise, ecologists who describe rain forests as ecological webs note that 'we can plant trees but not forests. They make themselves.' Managers and design theorists plant tasks and structures, but interdependencies are largely an outgrowth of such decisions.

FIGURE 3.5

A MODEL OF THE ORGANIZATIONAL DESIGN PROCESS



CHAPTER IV: WEBS OF INTERDEPENDENCY: STRATEGIES FOR MANAGING MULTIPLE INTERDEPENDENCIES IN NEW PRODUCT DEVELOPMENT

Using a multiple interdependency perspective as a research lens, this chapter examines interdependency management in six large scale software development product teams in two firms. Drawing upon interviews and product and process documentation, it identifies three strategies for managing multiple interdependencies: systemic manipulation, constrained systemic manipulation, and local reaction. In this study, the highest performing projects adopted a systemic manipulation approach in which they focused on the pattern of interdependencies as a design variable. These teams made critical up front design decisions (about product technology, product strategy, and organizational structure) in order to simplify task coordination, prioritized interdependencies, gained leverage by efficiently performing central tasks, and were aware of the path dependent nature of interdependency. The results suggest that interdependency management can be viewed as a strategic process variable insofar as interdependencies are manipulatable through strategy and organizational design. This flexibility is constrained, however, due to the existence of certain technical and organizational legacy factors, which tend to embed tasks and task relationships in a prior history of structure.

4.1 Introduction

Research in strategic design constitutes one of the foundations of management theory (Ancona, Kochan, Scully, Van Maanen, & Westney, 1996; Burns & Stalker, 1968; Galbraith, 1973; Lawrence & Lorsch, 1967; Lorsch, 1977; Nadler & Tushman, 1983). Originally labeled simply "organizational design," this body of work is increasingly incorporated under the strategic heading as more and more scholars recognize the importance of linking theories of strategy (what to do) and organization (how to accomplish it). This theoretical transformation has been accompanied and perhaps motivated by an increased emphasis within firms on issues associated with execution and implementation. As a result, both researchers and managers are moving away from traditional notions of design as simply a form or type of structure and instead adopting the viewpoint that structure or "structuring" (Eccles & Nohria, 1992) can be a powerful and malleable tool for achieving strategic advantage.

Despite this increased emphasis, one of the key concepts underlying theories of strategic design, interdependency, has attracted relatively little recent empirical or theoretical attention. As a result, our understanding of design is actually quite limited and dated. For example, we are still primarily drawing upon the notions of sequential, pooled, and reciprocal interdependency (Thompson, 1967) and thus basing our ideas about how to organize and coordinate work on concepts that are thirty years old. While that early research contains some fundamental insights, our theories need to be updated to reflect the complexity that characterizes work and organizations today.

Furthermore, prior research on the relationship between interdependency and organizational design largely addresses the effects of single contingency factors (or mutually consistent ones) on organizational structure and thus fails to capture the realities and complexities associated with organizational efforts over time. Most organization theorists today acknowledge that there does not exist one ideal form of organizational structure (Bowditch & Buono, 1985; Daft, 1983). Rather, work needs to be structured or designed such that it "fits" or matches the particular context (Galbraith, 1973).[27] Yet while theorists acknowledge heterogeneity across organizations, they have largely failed to consider heterogeneity of context/structure within a given organization.[28] We know that work should be structured differently in different contexts; we know little if anything about how a given organization structures itself to deal with multiple contexts.

---

[27] Context is usually interpreted here as including technology and environment (inside and outside the firm).

[28] Read closely, Thompson's typology does in fact imply intra-firm heterogeneity. He proposes that all firms have pooled interdependency, some have pooled and sequential, and the most complex organizations have all three. However, Thompson never considered the implications of this variance, and subsequent authors have largely presented each type as a firm-level variable.

The existence of multiple interdependencies raises a series of central theoretical and managerial challenges. Gresov notes that work units must be organized to respond not simply to the content of their work (technology) or to their position within a work flow (environment) but to these two (and possibly other) contingencies in combination (Gresov, 1989; Gresov, 1990). Given a development project in which team members must coordinate their activities to produce a complex technology, respond and adapt to an external environment, and possibly interact with other project teams in the firm, how do team members plan to meet those demands? What interdependencies do they concentrate on, and how do they decide which coordination solutions to put in place? In other words, how is the work structured for multiple interdependencies? There is a consequent need for a research approach that examines the multiple forms of interdependency product teams face and considers their implications for structures and processes. While individual scholars have studied specific kinds of interdependency and their coordination solutions, no multiple interdependency paradigm has emerged.

This research study is part of an attempt to address that gap. A prior chapter addressed the question of why and how tasks are related in new product development projects and yielded a taxonomy of different types. This chapter explores the strategic and organizational structure implications of that taxonomy. After presenting an overview of the multiple interdependency perspective taken here, it analyzes six new product development teams. The focus is on the teams' approaches to different types of interdependencies, how they dealt with the conflicts associated with trying to coordinate multiple interdependencies simultaneously, and the outcomes associated with their divergent approaches. The epistemological perspective adopted is that of induction and grounded theory building (Glaser & Strauss, 1967).

4.2 Theory

4.2.1 An Overview of the Multiple Interdependency Perspective

Historically, scholars have focused on three main factors which influence decisions about organizational structure: the organization's technology, environment, and size (Bowditch & Buono, 1985; Daft, 1983b). For example, we know a great deal about how to structure and coordinate work for a particular type of technology (Allen, November 1986; Henderson & Clark, 1990; Perrow, 1967; Pimmler & Eppinger, 1994; Thompson, 1967; Woodward, 1965) and how certain environmental conditions necessitate particular designs (Ancona & Caldwell, 1990; Burns & Stalker, 1968; Gresov, 1990; Lawrence & Lorsch, 1967; Wheelwright & Clark, 1992).

We are far from understanding, however, what it takes for a company to effectively design and implement a comprehensive (multiple) interdependency management strategy (Miller, 1996; Nobeoka & Cusumano, November 1995). Adopting a multiple interdependency perspective shifts the focus from the organizing implications of single contextual factors to one of designing organizational solutions to a pattern of relations. It promotes a more global or unified view of organizational design and interdependency management, which integrates and reconciles previous understanding.

A multiple interdependency perspective also raises new research questions by highlighting the conflict and inadequacy in current design approaches. For example, most prior research found that lower organizational performance is related to a lack of fit between context and organization design, but no research has addressed the question of why such misfits occur or under what circumstances a lack of fit is likely. Notably, most authors simply attribute a lack of fit to misperception or ignorance of context,

incompetence, or other "departures from the norms of rationality" on the part of managers (Thompson, 1967) but never explicitly examine these assumptions.

Yet designing an organization (or project) to handle several contingencies at once may involve tradeoffs that make achieving an optimal design "fit" more difficult or even impossible (Andres, August 25-27, 1995). Recent research suggests that misfits may in fact occur as a functional response to the multiple contingencies faced by a work unit. For example, scholars have documented the problems product development teams encounter due to the need for both structure (to develop a complex technology) and flexibility (to respond to unforeseen competitive events in their environment) (Allen, November 1986; Brown & Eisenhardt, March 1995; Cusumano & Selby, 1995; Iansiti, 1994; Imai, Nonaka, & Takeuchi, 1985). Miller found that internal and external requirements are often incompatible, resulting in a mis-alignment of structural and process variables (Miller, May 1992). Other researchers have explored the circumstances under which internal and external couplings are inversely related (Weick, March 1976).

Coordinating multiple interdependencies may also overwhelm or interfere with a team's productive capacity, raising questions of how teams balance time to coordinate with time to produce (Malone, February 1988; Perlow, 1995). Theories and empirical data tend to support the hypothesis that this conflict exists. For example, the amount of coordination on a team depends on the number of communication links (interdependencies) that need to be established and increases non-linearly with team size (Allen, 1977; Brooks, 1975). Interviews with programmers and system engineers working on software development projects suggest that as the size of the team increases, communication overhead in the project can quickly get out of hand (Curtis, Krasner, & Iscoe, November 1988; Perlow, 1995; Perry, Staudenmayer, & Votta, July

1994). McCabe's data indicate that 50% of a typical programmer's time is spent interacting with other team members (of the remaining time, 30% is spent working alone and 20% is devoted to administration and travel) (McCabe, 1976). Observational studies at one large firm revealed that people spent one-half of their time in meetings, and developers attended one meeting, on average, for every line of code they wrote (Sumner Jr., April 20, 1993).

Another telling indicator of coordination cost in product development is data on design changes. For example, during the construction of the Boeing 777 airplane, the team working on a twenty piece wing flap found 251 interferences where parts occupied the same coordinates in space. All design activity on the project had to be suspended every few weeks during the main design phase while team members looked for problems that had arisen because of the interferences between one subsystem or set of parts and another (Sabbagh, 1995).

Design changes in software development tend to be even more frequent. On the first version of Microsoft Windows NT, there were 150-200 component changes per day in the weeks leading up to release (Zachary, 1994). A major telecommunications subsystem experienced approximately 132,000 changes over its 12 year history (averaging approximately 30 per day); as many as 35-40 different team members "touched" parts of the subsystem on any given day (Graves, 1996).

Clearly, finding ways to coordinate more efficiently and developing a conceptual framework to understand the multiple forms of coordination would make a substantial contribution to both theory and practice. Although myriad research questions exist, this study addressed three questions applied to six software development teams in two firms. The questions were: (1) What strategies, if any, did these teams use to meet the

demands of multiple interdependencies? (2) What approaches (techniques, structures, processes) differentiated these strategies?[29] and (3) What impact did these multiple interdependency strategies and approaches have on subsequent product team performance?

4.3 Methods

4.3.1 The Approach of the Study

Since traditional models do not attend to the issues identified above, this study used a comparative case study design (Yin, 1984) to explore the management of interdependencies in an analytically inductive fashion. The sample consisted of six case study large scale software development projects in two companies. The data consisted of 71 interviews, extended field observations at both sites, a questionnaire, and product, project, and process documentation. In forming hypotheses, I drew upon independent assessments of what was going on from multiple individuals and sources, using multiple methods, including interviews and documents. I also looked for consistent patterns in the data before drawing conclusions and routinely challenged the hypotheses as they emerged with evidence of potential threats to validity, but the results are nevertheless subject to limitations and biases of judgment.

Yin and others point out that there often exists confusion between the case study as a research strategy and specific methods of data collection (Eisenhardt, 1989; Yin, 1984). People tend to equate case studies with ethnography or participant observation, which are perhaps better thought of as particular methods for data collection. In order to keep this confusion to a minimum, the research strategy used here is broken down into four major components. Below I provide background about the sample-- the

[29] Consistent with the exploratory nature of this study, I took a very liberal definition of approach and included both formal and informal structures (e.g., group arrangements) as well as roles, processes, technologies, and tools.

companies, Lucent Technologies and Microsoft, as well as the primary task, internal team processes, and interdependency challenges in each of the case study projects. Data collection, analytical approach, and specific analytical techniques are described in detail in Appendix D.

4.3.2 Design and Sampling

The present study centers on six large scale software development projects in two firms and thus adheres to the "comparative case study" or "multiple case-embedded" design (Yin, 1984). There are multiple units of analysis (six case study projects), each of which is examined in terms of its interdependency profile, strategy, structure, and performance. Comparative studies have been shown to be particularly useful in helping to establish the generality of a fact or the structural boundaries of a phenomenon (Glaser & Strauss, 1967).

Yin suggests that one should consider multiple cases as one would consider multiple experiments and thus follow a cross-experiment or replication logic in both sampling and analysis. In terms of sampling, this translates into selecting each case so that it either (a) predicts similar results (a literal replication) or (b) produces contrary results but for predictable reasons (a theoretical replication) (Yin, 1984). The companies and projects for this study were chosen with both of these criteria in mind.

The Companies

Despite their respective dominate positions, Lucent Technologies and Microsoft currently find themselves in a very competitive environment undergoing significant technical and market transition. In terms of their product development efforts, this translates into a need to develop very large, complex software products that are reliable, competitive in terms of their feature set, and attractive to customers. Competitive

pressures further dictate that this be done quickly and efficiently. Although these firms each face a very similar technical and market situation, they do so from very different historical legacies of success.

For example, Microsoft has traditionally operated by instituting per product loyalty and focus. This approach is being increasingly challenged, however, as its products increase in size and become more integrated and system-like (Cusumano & Selby, 1995). Lucent is transitioning from being a producer of bundled system products to largely unbundled features. Thus, although the issues concerning product development at the two sites are quite similar, each firm's response strategy in terms of how they design and manage their teams should be quite different. This heterogeneity ensures a wide range of management outcomes consistent with the goals of the study. Finally, both firms had previously identified the management of interdependency as crucial to their future success (Cusumano & Selby, 1995; Perry, et al., July 1994), which greatly facilitated access and cooperation and also substantiates the importance of this research topic for managers. Below I review the two firms in more detail.

Lucent Technologies

Lucent Technologies, formerly part of AT&T, produces telecommunications switching system software and hardware products.[30] The company's four primary lines

[30] In September 1995, AT&T, one of the largest companies in the world, announced that it would voluntarily split itself into three new companies: a service provider (AT&T), an equipment provider (Lucent), and a computer maker (NCR). This unprecedented act was viewed as both a tacit acknowledgment that the large integrated company had become too unfocused and unwieldy to manage and a response to recent and impending changes in technology and government regulation (Landler, (September 21, 1995). In particular, AT&T's dual role as a major equipment supplier and a giant telephone company was creating excessive coordination costs and conflicts. AT&T's progeny, the seven "Baby Bells," are prime customers for its digital switches and other networking equipment. But as AT&T's service division began to compete in more and more local markets, the Bells became increasingly reluctant to buy equipment from AT&T, viewing each new order as effectively an assault on their markets. AT&T also ran into resistance from government owned telecommunications companies abroad, which were buyers of AT&T equipment but competitors in the global service market. These tensions eroded AT&T's share of the market for giant network switching systems in both the U.S. and abroad quite

of business are systems for network operations, micro-electronics, business communication services, and consumer products. In 1996, Lucent's annual revenue was $23.8 billion with profits of $1.1 billion. R&D investment was $2.55 billion, 10.7% of revenue. Network systems, the focus of this study, which produces switching systems that connect telephone calls, had sales of about $3.5 billion in a global market of $27 billion in 1996. Successive rounds of down sizing and cost cutting over the last several years have resulted in a fairly resource constrained environment within the firm, although Lucent appears to be rebounding recently (Schiesel, (April 13, 1996).

A telecommunications network can be viewed as a collection of interconnected terminal equipment that provides end users (businesses, residences, and public agencies) with different forms of telecommunications service. The basic functions performed by switching systems are the local switching of calls and toll connections between switches (Sneed, Huensch, Kulzer, & Moerkerken, November/December 1994). The real source of revenue, however, resides in innovative optional features, which overlay the core infrastructure. These include things such as credit card dialing, message waiting and notification, conference calling, and automatic number identification, all of which are software based. Product development life cycles at Lucent average 2-5 years. Product size can vary from 10,000 to several millions lines of code (Factor & Smith, July/August 1988).

The importance of software development is evident in certain key statistics on the company. Prior to the 1995 break-up, AT&T spent about $3 billion a year on software development, and annual production was estimated to be 50 million lines of

code in new product releases to customers.[31] Top managers have also stated that "software is the single most important capability" the company has (Sumner Jr., April 20, 1993). This technological emphasis is further reflected in recent demographic and managerial changes in the firm. In 1988, approximately 40% of the R&D community at AT&T devoted most of their time to the design, coding, and testing of software (Factor & Smith, July/August 1988). By 1995, that figure had grown to over 90%.

Approximately 12,000 of Lucent's 121,000 total employees formerly worked at AT&T Bell Laboratories, the R&D unit of the organization and now referred to in Lucent as Bell Laboratories. The average employee is about 40 years old. Most people have worked for AT&T in some capacity for more than ten years. Engineers primarily have masters degrees in electrical or mechanical engineering or computer science.

Two factors characterize the culture at Lucent. First, the company has more than one hundred years of experience installing and developing telecommunications equipment. People, therefore, take great pride in the technical excellence of the products, particular their reliability and robustness. Second, the culture of Lucent is rooted in a long tradition of monopoly. Before the 1995 trivestiture, the primary customer for products and services was AT&T itself. Although things are changing, most Lucent engineers have little contact with or knowledge of the customers for the products they work on. For example, few members of one case study team could name the Far East customer for their product or provide details about a competitive product already in the market.

---

dramatically and foreshadowed limited opportunities for future growth and success. Today, Lucent Technologies is run as an independent company and is free to sell its products to a variety of customers. AT&T accounts for approximately 10% of Lucent's sales.

[31] Current figures for Lucent are not available, but we can assume they are comparable since most software development went to Lucent.

Microsoft

Microsoft produces software applications and operating systems for the personal computer (PC) market and, increasingly, multimedia, consumer, and Internet applications (Cusumano & Selby, 1995). In 1996, Microsoft's revenues were $8.67 billion with net income of $2.2 billion. R&D investment was approximately $1.43 million (16.5% of revenue). In contrast to Lucent, Microsoft operated in more of a growth mode during the period of this study with extensive hiring and construction of new facilities.

A PC system consists of several types of software products layered on a hardware base. Operating systems are special programs that control the primary operations of the computer. Application products represent the functions and services users want to perform, examples being word processing and spreadsheet analysis. Network operating systems enable distributed computing as well as multi-user communications and applications (Cusumano & Selby, 1995). Microsoft competes in all of these arenas. Product development cycles at the firm vary, ranging from several years for large system products to one year or less for some consumer applications.

Several important trends characterize the PC software industry (Cusumano & Selby, 1995). First, functions are increasingly migrating across layers, resulting in a blurring of distinction between types of products. Second, products are getting larger and more complex as companies vie to distinguish their offerings from those of competitors by adding new features and functions. Third, customers are increasingly demanding commonality and consistency across products. These trends represent important challenges in how companies develop their products and compete. In particular, they imply greater product complexity and inter project sharing and coordination.

Microsoft had approximately 17,000 employees in 1995, a rapid increase from its base of 1,000 only 10 years ago. Its culture has been described as fiercely competitive, individualistic, entrepreneurial, and dominated by the CEO, Bill Gates (Zachary, 1994), although these characteristics are changing as the company evolves from a start-up to a mature firm (Cusumano & Selby, 1995). Early on, small groups of developers had ultimate control for what and how they developed products. As products grew in size and complexity and became more central in the business operations of customers, Microsoft incorporated practices such as more formal testing and design and planning methods and hired experienced people from other companies, both of which modified the culture. Nevertheless, the firm still derives a large part of its strength from hiring smart, motivated people and encouraging them to care deeply about customers and competitors in a given product space (Cusumano & Selby, 1995).

The Projects

Within each company I selected software development projects of roughly comparable size and complexity in terms of the amount of code and the number of components in the product as well as team size. The selected projects varied, however, in terms of their degree of innovativeness and uncertainty (represented by product version number) and position in the system architecture. The choice of selection factors was based on past research (Boehm, January 1984; Cusumano & Kemerer, November 1990; Cusumano & Selby, 1995; Kemerer, 1997) and discussions with experts in the field. Actual project names are disguised.

Microsoft "Network"

Task

Microsoft Network is a network operating system for PCs. This product is in its third version of release, with approximately 5.6 million non-commentary source lines of

code (NCSL). At the peak of production, about 250 people worked on the product. Typical customers include major corporations, banks, and scientific institutions, necessitating high levels of reliability and robustness.

Internal Team Processes

The Network project was characterized by smooth internal processes with little conflict or politics. Leadership on the team was exceptionally strong; numerous team members described key leaders such as the head of development as being highly technically experienced and "hands on":

> [The head of development] writes a lot of code, watches check-ins, and knows what is going on in the system. He's a very active manager, very experienced. He can set the direction for the whole team in a 9:00 am meeting that day.

Interview comments also suggested that people identified with the team and product as opposed to particular functional tasks or components associated with their particular work assignment.

Interdependency Challenges

Network represents an extremely complicated technology, with hundreds of interacting functions. Most pieces of Network have traditionally been designed, developed, and tested internally (to Network), but the number of externally supplied components is rising steadily over time. For example, components such as MSN client support, TAPI fax support, MAPI, Exchange, multimedia, and system applets are now supplied externally. Network also shares technology (e.g., RPC, OLE, and TCP/IP) with another major operating system product.

Microsoft "Desk"

Task

Microsoft Desk is an integrated application "suite." It first came out in 1990 and is currently in its fifth version of release, with about 2 million NCSL and 200 contributors. The standard version consists of four applications; a professional version, targeted to business users, also includes "Data." Desk is produced in multiple languages and sold around the world. The Desk use environment necessitates the production of various forms of documentation to assist customers in the product's use (e.g., manuals, CDs, on-line help).

Internal Team Processes

Leadership in Desk was strong, but varied. The people interviewed cited several key people who played important roles and enjoyed high respect for their technical skills, but no one or two individuals appeared to stand out and serve as a rallying cry for the entire project. The functional groups also enjoyed markedly different status levels. Developers were generally perceived (by themselves and others) to be the "kings" who "make it happen." One of the applications was also seen has higher status.

There was a fairly high amount of conflict on the Desk project, although this was tempered by high respect resulting from past working relationships.[32] For example, application team members argued over the standardization of work processes like component integration and the very definition of key words like code complete.

Many individuals admitted experiencing a conflict between loyalty to their primary work assignment (an application product) and Desk. This was largely a result of Microsoft's prior application focus, which encouraged "tribal loyalty" and was further

[32] In particular, most of the people in the Desk unit had previously worked in one of the applications.

exacerbated by different practices within the groups. Several people described an apparently common fear that applications might "lose their identity" and "get taken over by Desk":

> [The applications] development are traditionally very proud and independent teams. There is some resentment about the role of Desk and loss of autonomy. For example, [applications] people complain that 'we spend all of our time supporting Desk features, so there's no time left in our schedule to do [our] features.' There's a Desk tax.
>
> There exist different histories and cultural practices and priorities in each group. Different strategies and levels of rigor. Now they need to buy into one set of criteria and get to the same place at the same time.
>
> It's very difficult to gain agreement on one [integration] process among so many people. It's a constant source of friction between [the applications] and Desk.

Interdependency Challenges

Prior to 1993, Desk existed as a bundle of discounted applications with minimal integration and sharing of components. The applications were separate products and projects with little technology or process integration. In 1993, partly in response to customer demands for more commonality and consistency of features, Microsoft created the Desk product unit with responsibility for authoring, integrating, and testing the Desk suite components (applications), although there was little formal coordination in terms of schedule or design at this point. Desk shipped after each of the stand-alone applications was released, and the Desk team was responsible for resolving inconsistencies in product design across the applications.

Microsoft began to recognize some of the inefficiencies associated with the above approach and radically changed its strategy accordingly in 1995. For example, the uncoordinated application schedules meant that Desk often shipped multiple times a

year as each application released a new version. Furthermore, since Desk was still a mere bundle of applications, there existed considerable variation of functionality and user interfaces across the applications at a time when customers were demanding consistency. This inconsistency varied from the relatively trivial (e.g., should the box be green or yellow) to major and complicated technical conflicts (e.g., single document interface and command ids).

The new strategy is considerably more Desk-focused, reflecting the suite's success in the market.[33] Going forward, Desk will be released on a 12/24 month cycle, alternating minor and major functionality additions. More importantly, Desk will ship first with the stand-alone applications released approximately one month later and all at the same time. The Desk unit is also responsible for developing certain shared features used by all of the applications as well as some stand-alone technology. For example, features like the toolbar, menus, and status and title bars are now shared. Other components such as OLE and MS Query are supplied by an external team but used by some or all of the application products.

The result of the above changes has been a dramatic rise in interdependency among the applications. At the level of product code, the applications are now directly dependent on Desk.[34] The high degree of sharing also necessitated more interaction among program managers and developers on various teams. The prior product and organizational histories of the applications complicated the sharing and integration process.

---

[33] By 1995, 65-80% of application sales came through the sale of Desk.

[34] For example, everything that was required to execute an application used to be contained in one file (the exe). Now certain key pieces of functionality have been moved into the Desk Dynamic Link Library (DLL); when you launch an application, it makes a call into that code. Thus, applications effectively no longer exist on their own-- the DLL is required even to run a stand-alone version of the application.

A primary area of concern on the team was the increased schedule risk associated with extensive technology sharing. The applications are now directly dependent on the ship vehicle (Desk) and indirectly dependent upon the other applications in it. For example, if one of the applications should miss the target ship date, it threatens not only the release of Desk but also the subsequent release of four other stand-alone products. As the lead program manager noted:

> As you introduce more components, there's more randomness. Take five products and a lot of little components. Do we really think that all of these people are going to finish on the same day? It means you have to take less risk.

Microsoft "Data"[35]

Task

Microsoft Data, code named 'Sterling,' is an integrated database application. Originally released in 1985, it currently stands at about 1 million NCSL and approximately 75 contributors. Customers of Data are so-called "power users" and tend to be attracted by very innovative, technical features.

Internal Team Processes

There was little evidence of either formal or informal leadership, and little sense of team or product unity on the Data project. Instead, people described their primary affiliation as a particular component. Authority was likewise centered in the components and functions, resulting in high degrees of conflict. The levels of stress and frustration on the project were extremely high. People described feeling overburdened by the amount of work and number of relationships they were expected to manage:

---

[35] This version of Data was not the version of Data appearing in the Desk case study.

> I feel like Sherlock Holmes tracking this stuff down. Have I talked to the right people? Do they understand my priorities? It's non-stop.

Interdependency Challenges

Data underwent a major architectural change in 1995, which significantly increased its level of external interdependency. Earlier versions of the product were primarily built on the version 1.0 code base; beginning in 1995, Data had four external teams supplying key functionality previously developed internally. These dependencies were in addition to Data's dependence on Desk because it ships as part of the professional suite.

Like the Desk project, increased schedule risk was a big concern. The fact that Data was both a heavy component user and a provider of functionality to the Desk suite resulted in a particularly tenuous situation:

> There's a burden caused by code sharing and componentization. If one component is late, everything [goes late]. And there are gaps in the release cycle because you can't slip continuously... Sterling could slip either zero or twelve months, in one year increments. If you don't make the [Desk shipping] window, you have to be picked up next year. It's an incredibly constrained process.

Lucent "Handphone"

Task

The Handphone product, positioned for a young urban market, provides basic code control for wireless hand-held communication in Japan. The wireless market is extremely competitive and growing rapidly. The customer demanded that the product be developed very quickly and closely tracked its progress. Handphone represented both new functionality and the recombination of existing technology in new ways. For example, the platform overlays cellular network communication on an established

ISDN architecture. A significant feature of the product is its low power usage, which enables longer talk time between recharging (40 hours versus 10 for comparable products). This version 1.0 product had approximately 50,000 NCSL (on a base of 10 million) and 100 team contributors.

Internal Team Processes

The Handphone project was characterized by smooth internal processes and remarkably high levels of cooperation:

> When you asked for something from somebody, no matter how busy they were, they would make the time for you.
>
> Issues always had a sense of urgency and people responded. Versus on other projects where you go with a problem and people say 'Yeah, I'll get to it,' and two days later, nothing has happened.

Like Network, leadership was exceptionally strong and visible:

> The feature engineer is extremely good-- very strong technically, and he knows this area very well.

Handphone was unique at Lucent in its use of real-time rewards to motivate people and keep them convinced that things were moving along. The typical practice at Lucent is to recognize and reward people formally after the project is completed:

> The feature manager gave out little trinkets to everyone at key milestones. It made for great lunches, and I actually kind of liked it. Certain things like that kind of help things along. There's a certain amount of psychological tension that you need to deal with, working so much.

Interdependency Challenges

The Handphone project had a number of things going against it from the start. Lucent customer representatives in Japan set the deadline before the requirements were even gathered. The project was also severely understaffed for the first six months.

Many of the developers and testers were transitioning off of two recent unsuccessful projects, creating low morale conditions. The primary challenge Handphone faced was how to respond to customer needs and resolve technical uncertainty, given limited resources.

Lucent "Tollphone"

Task

Tollphone is a 5E application that provides operator-assisted, toll (collect and credit card), and pay phone services around the world. First introduced in 1987 in Panama City, Florida, it has been under continuous development since 1984. Some examples of typical features developed in the past are 1-800-OPERATOR and out-of-band switching. The product is 1-2 million NCSL and in its seventh version of release. Approximately 100 people contribute to its production.

An international version of Tollphone was split off of the U.S. product in 1988/1989, and the two products have since evolved independently. Today, they are incompatible with very little common code. The U.S. and international products are also produced by different groups within the Tollphone department.[36]

Internal Team Processes

The Tollphone project exhibited a mixed leadership pattern with strong informal leaders (highly experienced technical people who have been working on the product since its inception) and a few strong formal leaders (department and division managers). The team exhibited relatively little conflict; most of the people had been working together for several years and appeared to get along well.

---

[36] International Tollphone development is further divided into one sub team responsible for India and on-going support efforts and a second sub team responsible for Japan and "the rest of the world."

Interdependency Challenges

The Tollphone case is interesting for several reasons. First, up until 1995, the department was organized as a more cross-functional team in terms of both the key product subsystems and functional disciplines. Gradually, some of these dependencies were centralized into other departments as the company moved towards a more functional organization. For example, development of the Recent Change and Audit subsystems was centralized into another department, as were component integration and field support. As we shall see later, the remaining team members had some strong opinions about the effect this structural change had on their relationships with those dependencies and ultimately their ability to get work done.

Like Desk, the Tollphone case also illustrates the considerable challenge facing a company when technology and customer needs change in ways that contradict its existing technical and organizational infrastructure. The product was originally designed as a system product with annual or biannual delivery, which usually involved incremental changes in functionality. Today, the nature of telecommunications products and services has changed such that customers are demanding more frequent delivery of smaller products.

Lucent "Autophone"

Task

Autophone is a brand new infrastructure for automobile phone customers in the U.S. The product is approximately 1-2 million NCSL and about 200 people contributed to its production. The project was geographically distributed between Illinois and New Jersey; about two thirds of the technical work was done in Illinois, with project management almost entirely in New Jersey.

The cellular market constitutes one of the most tumultuous arenas in telecommunications today. Competition is fierce, demand is high and growing exponentially, and yet customer needs are ill-defined and nascent. The technology is also not well understood, with a variety of protocols and standards in simultaneous existence. Autophone was one of two major technology standards vying for market dominance.

The Autophone project reflected this general high level of environmental instability. Due to competitive pressures and rapid technical advances, the team actually had three versions of the product in various stages of development during this investigation. Autophone v1.0 was in field testing, v2.0 was mid-cycle, and v3.0 was just entering the coding stage.[37]

Internal Team Processes

The Autophone project joined together two previously independent departments, and twelve months into the work they had yet to establish standard working practices. Many of the team members were new hires or transferred from other areas of the company that were down sizing. Some of the key leadership positions were vacant or held for only a short period of time (i.e., director of development). As a result, there was little sense of team unity on the project:

> The problems we are experiencing are organizational and process problems... There's a general lack of accountability. No one owns or is accountable for product integration or the quality of the life cycle of development.

[37] This analysis focused on v1.0 but also considered the implications of needing to coordinate with other versions.

The impression arising from the data is a work environment in which people are constantly responding to "fires" and crises as they occur:

> I think we kind of get stuck in that trap. We don't have time to think through things and end up addressing them issue by issue. Which in the end makes it seem much more frenzied and chaotic than it needs to be.
>
> Discussions are usually on an ad hoc basis, provoked by 'there's an issue or critical decision to be made right now.'

Not surprisingly, frustration levels were high. Team members described the difficulty of staying "close to the heartbeat" and keeping up with work changes as well as process turmoil. Several people complained of feeling helpless, and many appeared to have disengaged themselves as a coping mechanism:

> I honestly don't see how this can ever work, but I don't lose any sleep over it. It's gotten to the point where I go home and say 'If it works, it works. If not, I'll get a call tonight.' I don't care. I don't personally own it. It's way too big for me to get my arms around, so you have to back off.

Interdependency Challenges

The Autophone project fits the classic profile of a first time development project and illustrates interdependency management at its most challenging. The team struggled with the technical integration issues associated with any unproven technology compounded by (1) a geographic barrier and (2) historically embedded paradigms and processes, which now needed to be reconciled into one department. This case also reveals some of the futility associated with relying on traditional interdependency management techniques, which seek to identify and "freeze" interdependencies in advance, when a project exists in a volatile technical and market environment.

4.4 Results

4.4.1 General Level and Implications of Interdependency

Data from the interview transcripts and project documentation indicated that all six teams faced a complicated set of interdependencies, a situation many team members perceived as getting worse over time. In each case, the sheer size and complexity of the software product resulted in complicated technical relationships among various product components with subsequent implications for the coordination of design, development, and testing tasks.

For example, functionality in the Handphone product was divided among four core wireless components and twelve interfacing subsystems; the four components were further subdivided into fifty two coding units and allocated among many engineers. Technology in the Network product at Microsoft was similarly divided among four large subsystems and numerous people. While large system projects are hardly new, team members at each company with prior system development experience cited certain competitive market factors, namely shortened development cycles and increased requirements volatility due to uncertain or heterogeneous customer demands, as significant exacerbates of technical interdependency management.

In addition, the level of external interdependency was high and perceived to be increasing over time on all of the projects. One driver of external interdependency was an increased need for specialization and efficiency; more and more tasks were being outsourced to groups internal or external to the firm in order to take advantage of specialized skills and knowledge or efficiencies of scale and scope. For example, the Desk project relied on approximately forty other groups (inside and outside Microsoft) to provide files or pieces of the product. Desk in turn supplied technology such as toolbars and components to about ten groups in Microsoft.

The increased level of external coordination also reflected changes in product strategy at the level of the firm, namely a movement towards product platforms with high levels of cross-product component sharing. This type of external interdependency proved to be a significant challenge for several of the project teams, particularly at Microsoft where the culture has traditionally stressed team independence and autonomy.

The high degree of task interdependency had two striking implications at the level of work processes. First was a certain amount of randomness and unpredictability. For example, team members cited instances where seemingly minor or unrelated events nevertheless had large repercussions because they triggered a cascade of task contingencies. These ripple effects often resulted in seemingly unexplainable links across projects. At Lucent, for example, the testing and release of a feature to make automatic collect calls in Japan became tied to an Australian wireless product due to technical interdependencies in a globally shared file. At Microsoft, delay in the shipment of one product impacted tasks on numerous other projects due to code sharing interdependencies.

The second work process implication was a heightened tension between time to coordinate and time to produce. As the level of interdependence rose, individuals simply became overwhelmed by the number of dependencies needing to be tracked and managed. As a result, coordination efforts started to subsume production, or team members find themselves devoting time and attention to the 'wrong' interdependencies. Like the random unpredictability effect described above, this emerged as a consistent theme in multiple interviews on each case project. Team members talked about their struggle to balance individual and interpersonal tasks within the confines of a working

day and tight development cycle. Much of people's time (according to their statements in the interviews anywhere from 40-75%) was devoted to checking and rechecking work and responding to changes (in designs, schedules, etc.), suggesting a connection between the level of unpredictability and randomness and coordination effort. Finally, many of the interview subjects complained of deriving less intrinsic satisfaction from their work as more and more of their effort and attention was absorbed by coordination and checking, leaving less time for innovation.

The following quotations serve to illustrate the working environment on these teams:

Technical Complexity in a Competitive Market Environment

- "Traditionally we've only had one active [release] at a given time. We followed a waterfall model, so once it was tested and generally available, we split off and start the next release. It worked but the cost of being end to end is a very long development cycle. Customers have been pressuring [us] to turn products around faster, so we started to overlap waterfalls. These processes don't lend themselves to that kind of overlap because you're trying to split from something that isn't necessarily stable... It just adds to the cascading." (Autophone developer)

- "Many interdependencies were hidden before when things were done [more] sequentially." (Desk tester)

- "We're working in a very complicated area. You're trying to get a handle on your own stuff, much less make the time to figure out if you're synched up with everyone else... It's just real easy to kind of be blasting through things and not have the time to step back and think 'now wait a minute, let's look at this end to end. How is this going to come together?'" (Autophone developer)

- "Our customer needs proof very early on that we are going to deliver. They set up milestones and say 'show us proof' and certain external milestones can't change. This is not the way [we] usually work. Lucent uses waterfalls, and you don't take anything out there until it is ready to go." (Handphone developer)

(Increased) External Team Interdependency

- "On my first project, there were five people in my life, period. Now, I couldn't even count them all. A hundred? And they are spread across the whole company." (Desk program manager)

• "Managing external deliverables is one of our biggest issues. Things like compilers, debuggers and run time libraries. These are the traditional type of project interdependencies where we need a delivery from an internal or external group. As Network encompasses more and more features, it involves more and more of these outside interdependencies." (Network program manager)

• "We depend on a lot of other teams for stuff! In order to do wireless call processing, you need pieces from Recent Change, Audits, Packet Switch, Peripheral Control and Routing [subsystems]. Nobody could really survive individually. We all need to talk to other people." (Handphone developer)

• "This version of Data is going through a major architectural change. It's built on the same code base as Data 1.0, but now four critical components are being supplied by external teams... Plus our dependency on Desk has increased substantially; there's a lot more functionality delivered from Desk that we are dependent on." (Data program manager)

• "Microsoft's culture is such that it is a collection of product fiefdoms. Power [and rewards] come from having smart, motivated people who care deeply about customers and competitors in a given product space. Who cares about what's going on in the rest of the company? The [shared component] view flies in the face of this inbred philosophy." (Data developer)

Heightened Sense of Unpredictability, Randomness and Contingency

• "The real problems occur with hidden interdependencies-- the ones no one thought about in advance that pop up at the last minute. Or that just sort of materialize out of thin air because of some [new] need. Sometimes a solution to one interdependency creates problems for a third party." (Network program manager)

• "As you introduce more [shared] components, there's more randomness. I don't know where to focus my energies, where the weak spots are. Five things depend on one thing. What's the risk of that one thing?" (Desk developer)

• "[Some new code from another project] went in recently, which impacted a little bit how things work. We couldn't anticipate it because it is in a completely different area. The changes seemed innocuous but ended up breaking our code disastrously. We couldn't even imagine or think 'how could that impact this?' It was a complete nonlinear effect." (Handphone developer)

Tension Between Time to Coordinate and Time to Produce

• "Before we simply wrote the topic. Now 75% of my time is maintenance and behind the scenes work. Checking and rechecking, keeping up with changes. Maybe 25% is actually inventing and writing topics, which is the most rewarding." (Desk documenter)

• "The tension between production and coordination is huge. We definitely have a hard time getting work done." (Data developer)

• "Probably 20% of my time is spent actually coding; 80% is coordinating. For example, every morning I face at least thirty email messages. That's two hours worth of processing right there." (Autophone developer)

• "My life is one big interdependency these days. Everything I do, I am either a dependency or dealing with a dependency." (Desk program manager)

4.4.2 Team Strategies

The analysis of the six teams suggested three different strategies toward multiple interdependency management. The strategy of systemic manipulation involved pro-actively manipulating interdependencies via decisions at the level of product technology, strategy, and organizational structures. Teams applying this approach effectively simplified task interdependency through certain up-front design decisions. As a result, the subsequent coordination of tasks was greatly simplified. The teams also minimized external interdependencies, prioritized those that did exist, and sought to make "hidden" interdependencies visible to team members. For example, they took efforts to prioritize and standardize their external product environment and replicated that environment internally. They also used coordination mechanisms that increased people's awareness of other's dependence on them.

Systemic manipulation was also characterized by the use of techniques to increase coordination efficiency, thus enabling work to proceed forward at a relatively rapid pace. These teams centralized duplicate coordination in special team-level roles and batched integration efforts in time in order to trim coordination overhead. Finally, systemic teams were unique in their awareness of the implications of choices over time. They explicitly avoided certain strategies and structures in recognition of future complicated task coordination scenarios.

Sometimes teams were unable to alter the design of their product or project structure due to legacy factors, yielding a second strategy called constrained systemic manipulation. These teams were, therefore, forced to adopt additional structures, processes, and roles to coordinate work. For example, they used special forms of meetings, sub teams, and communication mechanisms to compensate for complicated task relationships dictated by their structure.

The third strategy, local reaction, consisted of more piece meal and ad hoc management responses, often triggered by coordination crises. These teams had multiple task forces and evolving processes. They used living documents and email as decision-making and coordinating mechanisms and often resorted to forms of slack. Furthermore, teams in this last category primarily limited their approach to interdependency management to internal roles, tools, and processes. They rarely considered aspects of structure or design (even though they did not face the same legacy as the constrained teams) and never prioritized their environment. Nor did they exhibit an appreciation for the link between technical, strategic, and organization design and task coordination.

4.4.3 Interdependency Management Approaches

The last section identified the team strategies and generalized their approaches to interdependency management. This section reports on the actual techniques teams used for managing multiple interdependencies. It distinguishes between two categories of techniques. First are up front design decisions-- at the level of the technical product, strategy, or organizational structure-- that had implications for task coordination. Second are the roles and processes used for coordinating. Product component

integration is examined in detail in Appendix E, given its centrality in terms of interdependency management.

The Network and Handphone Teams: A Strategy of Systemic Manipulation

Data on the Network and Handphone teams consisted of six and sixteen interviews, respectively, a questionnaire, and numerous project memos, marketing brochures, and product schematics. These data, plus extended observations and discussions with other internal and external organizational members, indicated that these teams drew upon a variety of levers to simplify and coordinate task relationships.

Up Front Design Decisions

Figures 4.1a and 4.1b show the product and project architecture of Network and Handphone, respectively. As indicated in the diagrams, both had a simple, layered, modular[38] product design and parallel product and project architecture. The Network product consisted of four subsystems (Base, Network, Graphical User Interface (GUI), and OLE/RPC), which were allocated to four sub-teams. Handphone was divided into four components and sub teams (Location/Registration, which broadcasts or pages from the calling unit switch when a call is placed; Originating processor, which establishes the first half of the phone call from the caller to the switch; Terminating processor, which establishes the second half of the phone call to the called party; and Hand Over, which performs location/registration and new call processing as the conversation toll path roams across cells).

---

[38] A modular architecture is one in which each of the physical parts implements one or a few functional elements in their entirety; an integral architecture, in contrast, is one in which functional elements are implemented using more than one physical chunk (i.e., functions are distributed across physical elements) (Ulrich & Eppinger, 1995).

FIGURE 4.1a
NETWORK PRODUCT ARCHITECTURE

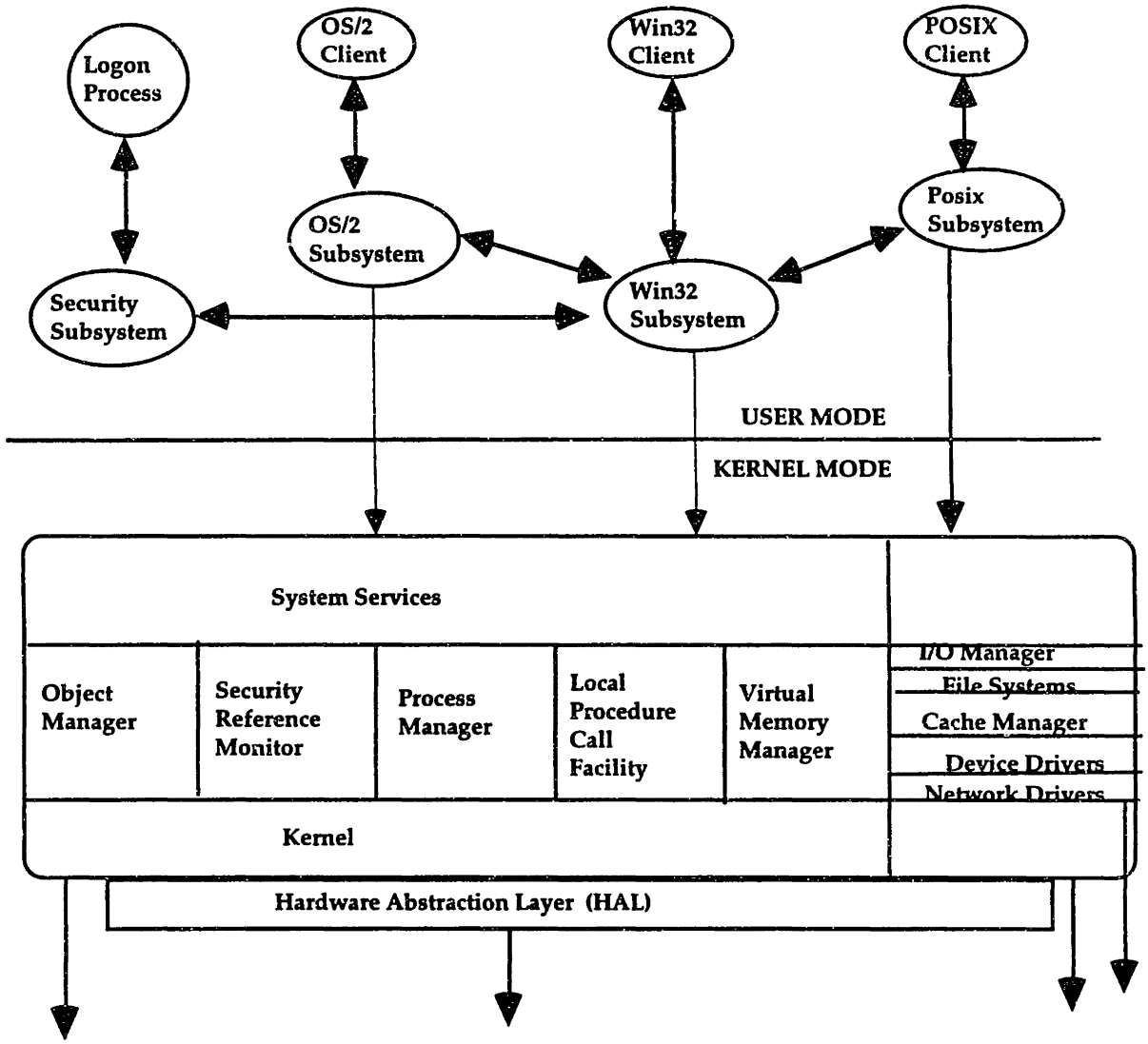
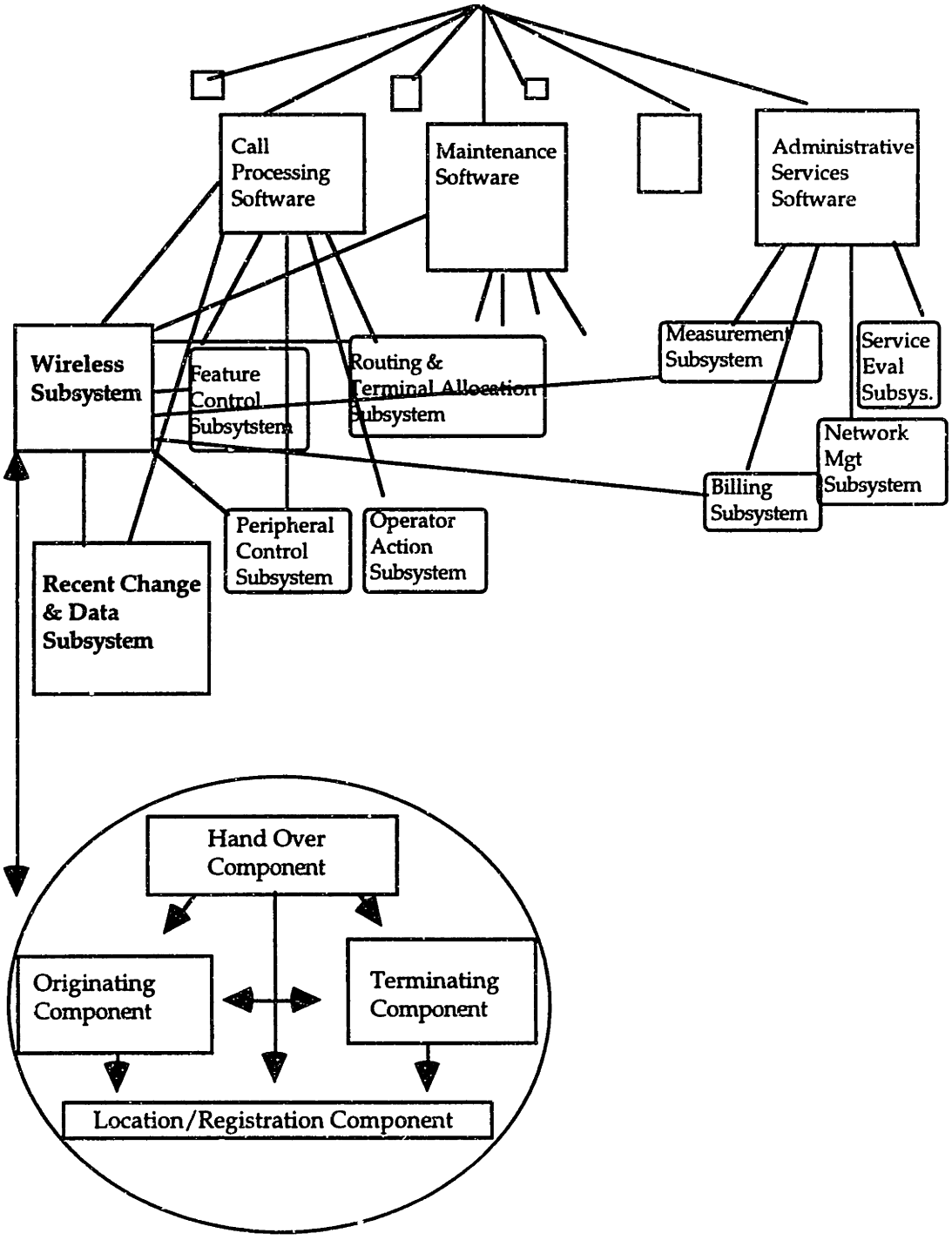


FIGURE 4.1b
HANDPHONE PRODUCT ARCHITECTURE



A modular structure with well defined interfaces enabled work to proceed relatively independently. The pieces of the product could be developed, tested, and integrated in parallel as opposed to sequentially. Layering made it easier to enhance and test the product since each layer can theoretically be replaced without affecting others.

Of course, some interaction occurred across component sub teams, but the interfaces between components were fairly well-defined with strict ownership understandings. As one Handphone developer put it:

> We try to define an interface that is as clear and simple as possible-- this is where and how we are going to touch. Defining this clearly doesn't mean we are not going to change it, because you always change it. But it has to be clear now so we can work in parallel and know what changes and avoid major architectural problems later on.

In those cases where product pieces drew upon or "came on top of" more core level functions and features, team members tried to "fake" certain functions and bypass blockages:

> In order for us to actually perform certain Hand Over functions, the originating and terminating components both have to work at a fairly stable level. But we couldn't wait until they worked; everything had to be done in parallel [due to schedule deadlines]. We did an excellent job of faking certain things during the first couple of weeks. For example, we simulated the data we had to create and manipulated things to make it look like the originating piece worked.

On Network, the parallel product-project design structure extended to the testing team. Testers were primarily organized by the four product components such that, for example, a developer-tester pair in the Base component would work closely and very interactively to find and solve bugs.

On Handphone, in contrast, the structure of the testing group did not parallel that of either the development group or the product. Although the two functions were in one department, developers were organized by the four components, but testers were organized by task (i.e., feature tester, deliverable tester, stress tester, etc.). This non-parallel structure was accompanied by more formal rules about the hand-off between development and testing. In particular, a tester who located a potential bug had to get it "approved" by a developer, meaning the developer had to agree it actually was a bug and assume responsibility for fixing it. The non-parallel organizational structure also resulted in more stilted interaction and often made it difficult for testers to locate the responsible developer:

> When there are problems, feature testers must provide feedback to where ever it should go. Finding where it should go is not always straightforward, and some developers need to be convinced that it is their problem.

Minimized External Interdependencies

Both teams sought to keep the number of externally supplied components to a minimum and utilized techniques to promote close collaboration when this was not possible. They deliberately avoided externally sourcing their four core components. Network and Handphone also assigned central contact points with primary responsibility for communicating with and maintaining each external relationship and often temporarily collocated the liaison. They used fairly structured communication processes with, for example, weekly pre-scheduled phone calls.

For example, the Handphone data engineer, while formally a member of the wireless department, was collocated for the duration of the project with the department supplying software from the ODA and Recent Change subsystems. As he explained it:

> I knew early on that my problem was not going to be interfacing with the Handphone developers; it was going to be with ODA and RC. I have to go beg them to do stuff because we are always behind. It's easier to do that down the hall than across another building.

The Network liaison for multi-media described the techniques he used to keep in touch with the external partner as follows:

> Every Wednesday morning we have a conference call with the multi-media group in the U.K. Even if there's nothing to talk about [in the sense of issues or decisions to be made], we talk to provide [them with] a general sense of where things are in the project.

Relations with other partners were sometimes governed by a "lift and modify" strategy. Network "lifted" certain features or functions from another operating system, for example, such as setup, shell, and help, and then modified them slightly in order to accommodate Network customer needs for higher reliability and robustness. Similarly, Handphone used established ISDN protocols and designs from previous wireless products.[39]

Targeted Up Front Product Design

Up front product design has several disadvantages. Namely, it takes time and can result in a loss of flexibility or the ability to respond to unexpected changes. Both Network and Handphone exhibited a targeted approach to product design in terms of both the amount of effort and techniques they applied to it. For example, they varied the amount of time and effort devoted to up front design depending on the complexity of the product component interactions. Network team members developed detailed designs in complicated product areas but "produced relatively few documents for

[39] Note one important difference in these two cases. Network was lifting from a concurrent development project while Handphone was reusing technology from past products.

things that have been around for a long time or are just being enhanced." While Handphone produced more design documentation than Network and spent longer doing it, a reflection of the Lucent process and culture, the data suggest some amount of discretion was applied based on the level of complexity.

Another way these two teams targeted design was through the use of both fixed and living specification documents. Some aspects of the products were specified in advance and allowed minimal subsequent change while others were developed using "living" specification documents in which designs were allowed to evolve over the course of the development cycle. In general, more core level functionality was "fixed" early on in order to facilitate progress and prevent "ripple or cascading" changes. User interface features and functions and peripheral functions (e.g., data on Handphone) were allowed to vary as information was gathered over the course of development.

These teams also targeted their control of design changes by only rigorously enforcing strict change policies at the very end of the development cycle (except in highly complex areas, as noted above). In other words, they targeted change in time. For example, in the weeks before product release, Network had daily 9:00 am meetings of the key functional leads, all of whom reviewed and discussed any proposed design and code changes.

Prioritized and Standardized External (Product) Environment

Although Network and Handphone sought to minimize external interdependencies and brought core work inside the team's boundaries as much as possible, some external interdependencies were inevitable. For example, both products existed in a complicated, heterogeneous use environment, and the teams needed to manage relationships with interfacing products and users. Network runs on four

hardware platforms (Intel, MIPS, Alpha and Power PC) and needs to be compatible with hundreds of application products and peripheral devices. Likewise, a wireless software product such as Handphone interfaces with a hardware mobile phone unit as well as other software (for maintenance, billing, connecting to emergency services, etc.). Network and Handphone relied on standardization and took efforts to prioritize their environment in order to manage these relations.

Standardization of interfaces was again an important technique, which enabled work to proceed relatively independently.[40] Network relied on Application Programming Interfaces (APIs), published points of entry between different layers of PC software products, which list procedure names, what parameters each layer will accept, etc. Handphone took advantage of global telecommunications standards, which govern how phone messages and data get translated and transported from one location to another.

The Network team took further steps to prioritize interfacing products by, for example, identifying the top one hundred applications (in terms of sales) and devoting testing resources to them first. Although Handphone interfaced with a narrower range of products, it needed to perform reliably under a vast number of calling scenarios. The team distinguished between "rainy day scenarios" (hard to reach conditions that were relatively rare) and "sunny day scenarios" (more common and easily recreated call conditions). Because approximately 80% of the real world cases fall in the latter category, the team made them the higher testing priority.

---

[40] The earlier description of standardization was across components within a product. This refers to standardization of interfaces across products.

Replicate External Environment Internally

Another technique the teams used to manage external interdependencies was to replicate the customer environment on site. In effect, they transformed some external interdependencies into internal ones. For example, Network placed multiple hardware machines in tester's offices. Handphone brought the hardware mobile phone equipment to the Lucent location and relied on simulators and labs to mimic user conditions. These techniques enabled the teams to begin testing relatively early in the development cycle and thus reduce the likelihood of surprises later on. They could also use multiple testing methods in parallel to discover "hidden" interdependencies:

> Our goal is to test the software as it is being developed. We want the test to start the same time the software starts. Not 'Oh, now that we have [the code], we can start to write the test.'
>
> In the past we would deliver the software to the site, but nothing worked. The simulator indicated everything worked fine, but the first time we used the real hardware mobile unit, it didn't work. It's because in the protocol, typically you have options and the simulator doesn't duplicate all those options very well... Those failures are easy to debug [early] but very hard once on [the customer] site.
>
> We performed [simulator], lab, and hardware equipment testing and integration in parallel. We tried to maintain the same progress on all three fronts so there weren't any surprises later on. No method is perfect. So when certain stuff was working on the simulator, we would go and try it out on the lab. Usually we found new issues there.

Centralized Coordination Tasks in Special Roles

The Network and Handphone teams centralized certain coordination tasks to trim the administrative overhead and enable team members to concentrate on production tasks. Supervisors also assumed personal responsibility for managing key resources in such a way as to minimize frustration levels and blocking on the projects.

On Handphone, responsibility for tasks such as scheduling meetings, reserving rooms, and setting up and maintaining labs had been delegated down to the engineer level as a result of corporate down sizing. As a result, engineers with fifteen or more years of technical experience often found themselves "running around, trying to find people, or trying to identify a meeting date and room." Lab facilities, in particular, represented a time drain for many team members:

> Lab time is scheduled in four or six hour blocks, so first there's contention for a spot. Then you need to get the lab in the right state for testing your particular product, which means loading and reloading different system releases. It can take two hours just to get it into the right state. If you don't get it right or run into problems, you can end up having no time left to actually run your tests.

In an attempt to solve these problems, one person on the Handphone team was assigned the role of "Wild Card." The Wild Card had no specific coding responsibilities but instead moved around and helped out as needed. Along with the feature engineer and feature manager, he effectively acted as a consultant to the team and assumed responsibility for scheduling all design and code inspection meetings. He also went into the lab in advance of developers and testers to ensure that the test facilities were in the proper working state. According to the people interviewed, this advance preparation made an enormous difference in their productivity:

> Usually, in order to do 10% of the tests, I have to know 100% of the lab. I have to know 'this machine does this, that machine does that.' [The Wild Card] became a lab expert, so the lab was always up and working when I went into test.

Network used a similar "Umbrella Support" person to handle unanticipated problems and requests in their product component integration sub group. On an earlier version of the product, two builders were responsible for integrating the components on

a daily basis. The build manager reflected on the coordination process under this arrangement:

> The builders would come in at 9:00 am and just go crazy all day long [handling unanticipated problems and requests from developers]. They used a white board and then a pad of paper to track the backlog of requests and people waiting to check their code in. It got to be reams and reams of paper, and they could never catch up. They were burned out and highly stressed. Plus, they often didn't have time to actually perform the integration, which just added to the backlog of problems and requests.

He decided to split the builders into two overlapping shifts. One person was responsible for the more routine parts of the job, namely performing the integration, while the second person was designated "umbrella support." According to the manager and others on the team, the result was a complete transformation of the work process. The first builder was able to work uninterrupted and, therefore, got the integration done on a regular basis.[41] A regular, predictable product integration in turn resulted in fewer change requests, and the umbrella support person was easily able to cover those that did occur.

Increase Visibility of Hidden Interdependencies

Managers and team leads on Network and Handphone were very aware of "hidden" interdependencies[42], and took active steps to unearth them early. For example, certain areas of the product were recognized to have "deep interdependencies." People working in those spots met frequently to try and discover them. A daily product component integration (described below) also helped both teams

---

[41] Product component integration is also quite manual and hardware intensive and, therefore, prone to error. The absence of distractions significantly decreased the probability of such errors.

[42] These included some "near neighbor" or first order relationships as well as second and third order ones.

identify interdependencies and code conflicts, as did using multiple testing methods in parallel (described above).

Product Component Integration

People on both teams were passionate about the importance of product component integration. They believed it was key to coordinating interdependencies and devoted a great deal of thought and attention to its management. They also invested resources in it by, for example, purchasing special, fast computers and placing their best technical people on it. (In both cases, gaining these resources required some degree of negotiation and persuasion of upper management.) Refer to Appendix E for a detailed description of how Network and Handphone designed this process.

Controlling Interdependencies Over Time

Finally, products exist in a use environment which evolves over time. This creates a set of interdependencies with past product versions currently in the field and necessitates tasks such as maintenance and configuration management. Network adopted specific strategies to minimize and manage these task relationships (Handphone was a version 1.0 product). In particular, they altered their product strategy.

Network deliberately chose to avoid or minimize "patching" bug fixes or new functionality on past releases. This minimized complications associated with configuration management such that "this only runs on version 3.5 with patch #7." Rather, the team's strategy was to add functionality via more frequent product releases.

The team further chose to not have a separate dedicated maintenance department or group. Instead, they adopted a Quick Fix Engineering (QFE) program, which was

the responsibility of current developers. Any customer reporting a problem deemed "mission critical" (blocking usage) was guaranteed that the problem would be diagnosed and solved within twenty four hours, if possible. Although this solution meant that maintenance and current development tasks were interdependent, Network management saw benefits in this conflict:

> Twenty engineers dedicated to maintenance assumes you have a lousy product. If I have a bug out in the field, most likely I have the same bug somewhere in the code I am working on. So it would behoove me to fix that bug in the code I am working on and then figure out how to fix it out in the field... The other good thing about doing maintenance this way is that the developer that writes buggy code will have to spend all of his time doing maintenance, which means he will be less effective. Which means... he won't get to write as much code on the current release. But that's OK-- it's a self-fulfilling fix for us.

Network also recognized the need to manage their resource interdependencies over time. Prior bad experiences had raised management's awareness of this factor:

> It took forever to get the first version of Network out. When it finally shipped, everyone was tired and burned out. Although they felt a degree of satisfaction, most didn't want to do it again. Once in a lifetime is enough! So we sat down and realized that if we did releases every year people wouldn't feel so bad. We could manage the attrition better.

Now Network encourages people to use the time between releases for rest and renewal. Many team members take extended vacations during this period. Others write project postmortem documents and reflect on what they learned or how things should be done differently in the future.

The Desk and Tollphone Teams: A Strategy of Constrained Systemic Manipulation

Data on the Desk and Tollphone teams consisted of seventeen and eight interviews, respectively, as well as project memos, presentation view graphs, product

schematics, and a questionnaire. While both teams endeavored to manage interdependencies proactively and systematically, Desk and Tollphone were constrained in their use of certain up front design techniques due to product and organizational legacies. These teams drew upon techniques such as prioritization, standardization, and slack, as well as special organizational structures, to counteract the problems presented by their legacy. They also used aspects of past working relationships derived from prior structures (e.g., friendship, respect) to overcome certain limitations imposed by their current design.

Up Front Design Decisions

Desk and Tollphone are what is known as "legacy products." They are currently in an n+1 version of release, and much of the product is original with the design intent of the version 1.0 product. New product features and functions often cut across this original design, creating a set of technical interdependencies that need to be coordinated. Yet the teams have limited ability to redesign the product architecture in order to simplify task coordination because the technology is embedded in multiple use environments around the world. To significantly alter the product would represent an unacceptable disruption in service to existing customers.

Likewise, prior organizational structures on each of the projects resulted in embedded task relations and ways of working. The teams' ability to coordinate in new ways was sometimes aided but often inhibited by these past designs and required them to adopt special forms of integrating mechanisms.

For example, product schematics indicate that the software code in Tollphone was organized into one major subsystem, Operator Administration (OA), which was further broken down into four main components: a position terminal processor (PTP),

which handled software at the customer site, an originator terminal processor (OTP), which handled interactions between the operator and caller's phone site, an automatic terminal processor (ATP), which included an announcement unit, and a Query processor, which performed maintenance and validation. The OA subsystem interfaced with approximately 50 other subsystems in the switch; the precise number varied depending on the particular type of product functionality being developed (Figure 4.1c). The product architecture was, therefore, not modular, nor was its design aligned with the current market. The interviews suggest that developers and designers were very aware of these architectural problems:

> Products cut across the design structure of the system. Everything is very divided up. In order to implement a product, you need to make changes in a lot of different places... Some connections [between subsystems] are obvious by reading the code, but many are not.

The Tollphone project exhibited a similar functionally distributed structure. A core group of developers and testers work in the OA department; they team up with people in other subsystem departments to produce a particular product. The development manager in Tollphone explained how this structure complicated the coordination of project start-up and development tasks:

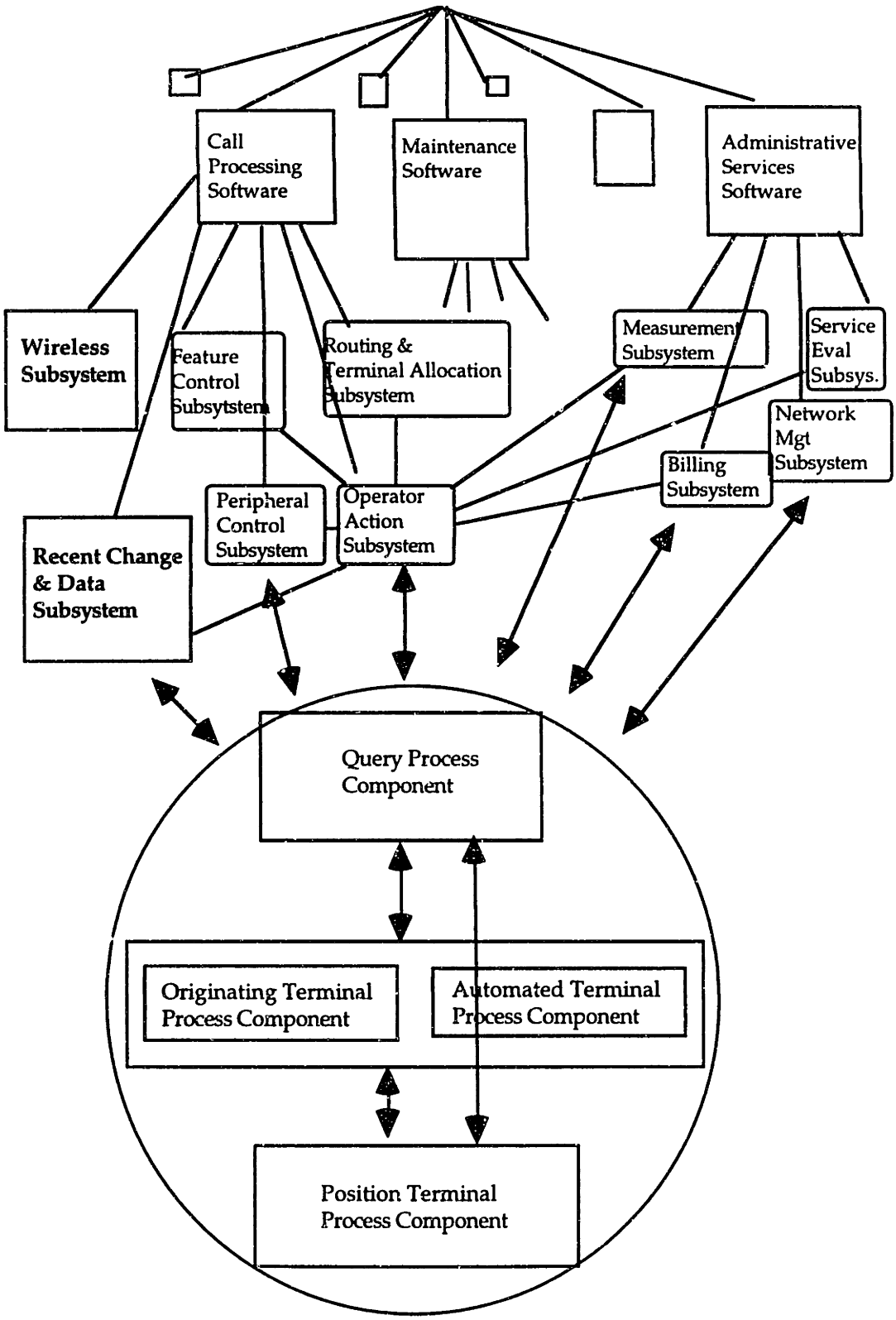
> I'm thinking of the process we use for committing to a product... Particularly with larger products, you may have people from a half dozen organizations who are contributing parts of that product. The process says that once the project is authorized, you must submit a development plan and staffing profile, which you get from each of the other managers involved. So you have to go to them and say 'product so and so has been committed for release on this date. We need six head counts worth of work from your group. Give me the staffing profile that commits those people to that interval to make that delivery.' And there's nothing in the process that says you socialize this well in advance... Those people may not know the business dealings that caused this to happen, and it's bound to come as something of a surprise every time... They have a bunch of priorities which may conflict... The negotiation of all of this ... can be difficult.

> If you have one developer who's working in the call processing part [in OA] and another developer working in the Recent Change/Data subsystem, their work interacts fairly closely. So it's important to be able to talk. 'When is your code going to be ready? When should we schedule a design review meeting? OOPS, I forgot that piece of data. What can we do about it?' Just those day-to-day interactions that make it easy... The problem you get into with these large functional organizations is that communication doesn't happen, or it happens in a very stilted and formal way. It makes it harder to get things done.

Ironically, this same manager pointed out that relationships developed in a prior more integrated (cross functional, customer-focused team) organizational structure helped the team members overcome these coordination barriers:

> It's clear that cross functional, customer-focused teams are the most effective. Tollphone had that type of organization, but now we are all broken up. Field support people moved to a different Vice President area, customer documentation-- who knows where they are. Recent Change/Data moved to another department... However, because they had been working so closely with our folks and are sitting in our aisle [in the case of Recent Change/Data], we are effectively still working as one department. There is an informal network that is going on there that is extremely helpful for our work.
>
> We kind of have a zombie organization. It's dead, but it doesn't know it because of these personal relations. If that ever changed-- people were moved physically or a whole new set of management came in-- our efficiency would drop dramatically.

FIGURE 4.1c
TOLLPHONE PRODUCT ARCHITECTURE



The architecture of the Desk project was similarly influenced by the past. The current organizational structure is one large Desk department, which is further broken down into separate teams of program managers, product managers, developers, and testers for each application. When asked why they maintained the application divisions rather than creating one Desk team, the lead program manager explained the choice as follows:

> Merging [the application teams] into one Desk team is in many ways the right thing to do, and we may eventually do it. But for now, knowledge of the code is important. There are millions and millions of lines of code in each of these applications, and it would be hard for people in Desk to be a super specialist in all of them.... We need to keep some amount of product focus vis a vis Lotus and WordPerfect... Plus, it would just be too big and unwieldy as one team.

Because the application developers were organized into separate sub groups, it was initially not clear who should design and develop the shared components. At first, Desk tried having people representing different applications collaborate on design, but, as one team member noted:

> Design is very subjective, and everyone has their own different priorities. Once you have multiple people working on design, the people here are too smart. What you end up with is three correct answers. What we really need is one correct answer.

Centralizing design within Desk proved equally problematic:

> Previously when we wanted to change how a Desk feature worked ... we had to go to all of the product teams with a list of things we wanted to do. One team would say 'we like 2, 4, 6, 8 and 10,' and another team would say 'we like 1,3, 5, 7 and 9.' Each group thinks they're [cooperating] because they're making five out of ten changes. But then you get in an awful situation where each group wants a different five, and we have to reconcile them.

The current solution was, therefore, to have a small integration team on each application sub group with a fixed amount of time in the schedule for integrating shared components. The lead program manager and lead developer in Desk described the solution as follows:

> Our solution is to declare a firm boundary of how much product definition the Desk group can take over. [The applications] donate one feature team worth of their group whose job it is to integrate all the shared technology and maintain an effective boundary.
>
> All of those problems go away now. For example, toolbars are written by Desk developers. We're back to the fun situation of a small group of people designing and developing it, not arguing in meetings or asking for permission. And we asked each application group to give Desk a fixed budget of integration time for the shared code. If you don't actually own that time, it's almost impossible when you're dealing with so many different groups to change things.
>
> So Desk can only do features that create integration time to fit into that budget. If they want to do something else, they work directly with the integration teams to take back some work and give them new stuff. It's more successful, and we can do bigger things.

The structure of the Desk documentation group did not parallel that of the rest of the team. The documenters, who perform tasks such as writing, editing, and producing the documentation, were organized into one sub team (i.e., no application product subdivision) within the Desk department. Like on Handphone, the non-parallel structure meant that a documenter trying to describe a feature had to talk with up to five different developers. The tasks within the two functions were also partitioned along different paradigms:

> [Documenters] writes about features from a user's perspective, as a user sees it. Developers work on features in terms of pieces of a module. So a developer may work on one aspect of a toolbar in milestone one and another aspect in milestone four. But we need to write about the whole toolbar-- the partitioning of the work isn't equal, and it creates a certain tension.

Minimized Number of External Interdependencies

The Desk team used aspects of organizational structure and process to control the large number of externally supplied components. For example, three external groups producing three interdependent components were temporarily reorganized into a 'cubed' team in order to minimize the number of group interfaces the Desk program managers had to keep track of.

Tollphone could not eliminate its external relationships with other subsystem department but cultivated certain external relationships carefully. For example, the Tollphone lead development manager noted:

> I personally have made a strong effort to work with the manager in RC/Data so we can have a special relationship. I make a point of maintaining and cultivating that relationship.

Targeted Up Front Product Design

Like Network, Desk targeted product design changes in time, instituting strict change control policies prior to product release. Near the end of the development cycle, the functional leads in the applications and Desk met every day in 5:00pm "war meetings" where they reviewed the status of bugs, considered proposed changes, and discussed new crises and events.

Standardized and Prioritized External (Project) Interdependencies

The Desk team drew upon several techniques to facilitate external sharing and coordination with other projects. One technique was process standardization in the form of a common development platform, tools, and schedule. The lead Desk developer was a particularly strong proponent of this, arguing that "if we can get enough things between the groups to be the same, getting everything done on time kind

of falls out of that." Other team members questioned its feasibility, given the strong established cultures and practices in each of the application groups.

A temporary transition solution was the use of a tiered approach. Tier 1 applications were tightly coupled in terms of their schedule and tools whereas Tier 2 were less so. Some of the shared components were also assigned a primary client or application who had slightly more leverage in terms of its design.

Desk used a similar tiered approach to control interdependencies between its core English product and foreign language versions. The English version of Desk was developed first in the U.S. and then immediately converted into German, French, Spanish, and Portuguese by the same team (or a team in Ireland). The team simultaneously developed a localization kit, which detailed exactly how the U.S. version was produced and what needed to be done during translation. Then vendors in countries around the world served as translation houses that replicated the product into other languages. Higher priority Tier 1 and 2 languages received constant file updates while Tiers 3 and 4 received final versions of the U.S. files only.[43]

Furthermore, the Desk team instituted a very systematic communication set-up between the U.S. and its Tier 1 and 2 translation partners. The two groups spent time visiting each other's sites and held weekly conference calls where the change information and materials were exchanged. One participant in the process described it as follows:

> It's very methodical. Always the same people delivering and receiving, using the same format, and posted on the same server. The consistency

[43] The former strategy represented a competitive market advantage because it meant that foreign and U.S. language products could be released simultaneously, but it necessitated extensive coordination with foreign development.

> really helps. In any body's life, the more consistency you can get, the more confident you can feel.

Other Desk team members maintained active status lists with issues to be resolved for each external partner and "touchable" metrics so they could track their progress over time.

Replicate External Environment Internally

Like Handphone, Tollphone took steps to replicate the customer calling environment on site so they could rigorously test the software code before it went into the field. For example, the software was placed in an actual working network "test bed" maintained by Lucent where it "soaked" for several days. Technicians monitored its performance there under various stress conditions such as high calling volumes.

Slack in Schedule and Feature Set

The Desk team dealt with high degrees of sharing by instituting more slack in the schedule in the form of increased buffer time. They also made sure that the set of shared features varied in terms of risk (i.e., it included both well understood and completely new features and functions).

Special Meetings

Most design decisions in Desk were reached by informal negotiation and brokering but "blessed" in meetings "like a marriage ceremony." Thus, meetings were used not for decision making, but to announce and reconfirm decisions already made in more informal settings.

The documentation subgroup in Desk instituted "4:00 pm stand-up" meetings. The theory was that without the comfort of chairs, people would stay focused on the issues, and the meetings would be shorter.[44]

Special Roles

In order to have the flexibility to respond to unforeseen development changes, the Desk documentation group designated a "floater" role. When developers changed their schedule or design, this person rotated to the area of the work most affected by the change.

Email

Both teams used email extensively, but it was viewed as a complement to, not a substitute for, face-to-face discussions. There was a general "reluctance to speak for third parties," and people seemed to feel that large group discussions were difficult to conduct over email. The nature of the discussions also often dictated more personal forms of interaction, as suggested in the following quote:

> Many things [having to do with shared components] need to be resolved in person, not email. The issues are so complex and involve technical and personal things.

The Data and Autophone Projects: A Strategy of Local Reaction

Information on the Data and Autophone projects consisted of five and six interviews, respectively, as well as product schematics, marketing and promotional material, observations, and a questionnaire. These data, coupled with additional conversations with managers at each firm, show these projects as the least pro-active and most chaotic in terms of the interdependency management. The teams failed to

---

[44] Note that Ford Motor Company is reportedly using "no chair meetings" on certain of its design teams (Jaroff, September 4, 1995).

make critical up front design decisions to simplify task coordination. Nor was there any indication that they prioritized different types of interdependencies. Instead, the teams relied on more traditional forms of coordination such as task forces, committees, and slack.

Up Front Design Decisions

Schematics indicate that the Data and Autophone products had a highly integrative architecture in the sense that functionality was distributed across key components. Changes in the software code in one component, therefore, depended on changes in other components. Both projects were structured as small sub-teams of peers with little if any integrating mechanisms across them. As a result, team members tended to either identify with their component or feel like they were simultaneously a member of multiple teams:

> I feel like there are six or eight teams I am a part of.

The product architecture of Data consisted of a shell and four key components: an underlying database engine, an embedded language compiler, which enables users to manipulate objects, Forms and Reports functionality, and visual designer tools such as Schema, Table, and Query. Two people described the cross-component inter-relationships as follows:

> We can't do a daily integration because of the way the interdependencies are distributed. Say you have six components and you try to smash them all together one day a week. Then component one, which depends on component two, which depends on component three... never has a chance to react to the changes the others made. The interdependencies are one-to-one-to-one, not one-to-one. It's integration hell.
>
> The shell sits on top, so it should roll up last. But there also exists calling back and forth between the shell and various components due to interactions. So if the shell wants to add new interfaces, they need to go

> early so the components can test it. So the shell should uniquely go both early and late... It's never going to work perfectly.

The project architecture the team imposed only served to complicate the coordination process. The four core components were designed, developed, and tested by teams with different organizational affiliations and geographic locations. One team supplied the database engine, a second group supplied the language component, a third team was responsible for developing the Forms and Reports component, and a fourth team developed the visual designer tools. One member of the fourth team described the resulting situation as follows:

> It's hard to get six teams to coordinate coherently. They tend to say 'Yes, it's true you depend on me, but I have three other groups depending on me, and they are higher priority from my perspective.'

The problem did not appear to be that the team members did not like or respect one another or get along; several people described good working relationships at the individual level. Rather, the sheer number and complexity of the inter component and inter team relationships tended to pull people in multiple diverse directions:

> If you read any of the software literature, they advocate avoiding interdependencies, and you feel and know why that is true [on a project like this]. Every time you depend on someone, you have to communicate with them, deal with them. You have to bend in some direction or they have to bend. And it's rare that two parties involved in a pairwise interdependency come together because they are being tugged by different things.

The Autophone project was similarly characterized by big blocks of product functionality connected by deep technical interdependencies and an organizational structure in which key components were implemented in different departments and locations. The 5ESS-2000 switch handled switching and links to other

telecommunications systems and was developed by another business unit in Illinois; the Executive Cell Processor (ECP) consisted of hardware and software necessary to set up and process calls and negotiate internal communication and was developed by a team in Illinois; development of the hardware and software for the Series 2 Cell Site, which performs actual call processing, was sub divided into an RCC and CCS component and distributed across Illinois and New Jersey.

A further complication was that the RCC and CCS code were partitioned using different paradigms. RCC code was partitioned and assigned on a per feature basis whereas CCS (which is closer to the hardware) was organized more functionally. A developer working on the RCC component described the problems she encountered integrating her code with that in CCS:

> Traditionally, an RCC person would have one point of contact [in CCS]. Each person would agree, 'OK, my changes are ready, yours are ready. Let's submit.' But now that [CCS] person may have dependencies such that there are a few other functional areas that have to come together to make it all click with RCC. Somehow, it comes up after the fact, 'Oh, so and so's code has to be there too.'

The need for smaller components slices with fewer interactions was particularly apparent given the intensely competitive and rapidly changing market environment of the Autophone project. For example, team members were unable to respond to customers demanding "late in the game adjustments to delivery" because the software was designed to be delivered in its entirety, not as particular features or functions:[45]

---

[45] Note the similarity with the Tollphone case here. The delivery paradigm in each project is changing more rapidly than the design of the software. In Autophone's case, the change is within a development cycle.

> We've each got our pieces. They're done and tested and ready to bring together, but now we want to bring them together piece meal. CCS wants to bring some of their functional areas, not others, but RCC is all wrapped up together. We thought we had talked it through-- you've got the right functionality to come together with my feature-- but when we went to do it, an official integration load effectively blew up... The problem is we're trying to change delivery after the code was developed. You don't have the flexibility at that point to break functional dependencies because you are not designing the code to deliver in that manner. You designed it to deliver all at once.

Core Components and Tasks Performed Externally

On both projects, core components-- features and functions that interacted with many other parts of the product-- were designed and developed externally by teams in other departments. On Data, four core components were distributed across four teams and two departments. Examples of outsourced core components on Autophone were Recent Change/Data and ODA.

Because these teams lacked a common departmental affiliation (they reported to different managers and were often governed by different incentive schemes), coordination across teams was difficult. Furthermore, there was little sense of product team unity, and people tended to be distracted by other demands within their own organization.

Collaborative Product Design, "Living" Specifications, and Strict Change Control Throughout the Development Cycle

The Data and Autophone projects exhibited other similarities in their approaches to interdependency management. Both teams tried to write very detailed design documents and specifications in large groups. The Data program manager described his recent experience in one such meeting as follows:

> Just trying to coordinate and print specifications from all of these teams is horrendous. It's hard to anticipate the size of the spec until you actually bring people together. Then you find out that each team's spec is 25-100 pages long, the entire thing is equal to two Manhattan phone books, and everyone has categorized things differently... My hot issue is someone else's ho hum that they don't care about at all. The program managers will leave the room saying 'Yes, we will all do X, ' and nothing will happen.

Both projects also relied upon flexible or "living" specification documents almost exclusively and experienced problems keeping the documents current and ensuring that people were working off of the same versions. Inevitably, developing code took precedence over maintaining documents, and documents were done after the product shipped, if then. Ironically, given the flexible specification approach, change control policies in each case were extensive but largely ignored. They existed throughout the development cycle and were decentralized into multiple committees:

> There are a bunch of [change] review boards! I can't even begin to name them all, and that's part of the problem .very time someone sees a crises, the micro organizational solution is to say 'we need someone to review this.' ... They're a bunch of spinning wheels. I don't think anyone has a global view.

No Standardization or Prioritization of External Interdependencies

There was little interface or process standardization on the Data and Autophone projects. Nor did the teams appear to assign a primary client or prioritize component use. For example, one team on the Data project adopted a completely different schedule than the other teams in terms of internal milestones and integration frequency. The Data lead program manager questioned the value of trying to define interfaces and processes across teams:

> There is value in letting each team define and have its own development process because believing in the commitments you make will make you a little more committed. It's a classic compromise and something that

everyone is just a little uncomfortable with, but also something everyone thinks will work.

Instead of standardization or prioritization, people used informal rules and heuristics to decide which product had priority in terms of component design and integration. Two popular decision criteria were "likelihood of shipment" and "technical challenge":

> The people working on [Data components] are very interested and challenged by the requirements of [another product]. Data has more of an end user focus, so the tools don't have high visibility. [The other product] is more of a developer focus-- it's more technically challenging because the tools are front and center and visible to the customer.
>
> We keep a running list of [host products] that depend on us, and once a quarter we reorder them in terms of priority. We use criteria like: do we believe they will actually ship when they say they will, which will ship first... And then there are 'jazz factors' like whether it's cool technology.

The Autophone project also had little success with interface or process standardization. The ECP and Cell code were developed on different platforms and used different header files, names, and change management systems. All sub teams followed a slightly different component integration process (See Appendix E).

Slack in Schedule, Feature Set, and Human Resources

The Data team put factors in their schedule to reflect the increased amount of time people spent coordinating (e.g., all senior people were 0% loaded, other people were 50-70% loaded).

Data also instituted some slack in the feature set. For example, the project sacrificed functionality in order to meet delivery deadlines on numerous occasions (they

tried and failed to share components at least five times). As the Data lead program manager noted:

> If there are two products with competing solutions and you are trying to find the be all and end all solution, sometimes it's easier and better to say 'let's just have two.'

Both projects utilized forms of human slack by increasing the ratio of coordinators to developers on the project.

Email

While all of the projects relied on electronic forms of communication such as email, the Autophone and Data projects tended to use these tools as a decision making forum. For example, an Autophone team member described an on-going technical dialogue about timing issues on the switch where, in more than twenty exchanges over two months, the team members debated different solutions. In Network and Handphone, in contrast, email was viewed as an arena for "small dialogues."

Increase Visibility of Hidden Interdependencies

Data relied on newsletters in an attempt to make interdependencies "public." But by the time the newsletter came out, the status of interdependencies had often changed. There was also no real incentive for people to either read or submit information to the publication, and no penalty for ignoring it.

Coordinating Interdependencies Across Time

Some of the most complicated interdependencies on the Autophone project involved managing task relationships across successive product versions. The team's strategy of overlapping product generations resulted in products and task relationships

that were complex and yielded unpredictable change effects. In particular, versions 2.0 and 3.0 of the product were each "split off" the prior release before it was completely stable, which necessitated some amount of mapping of code features and bug fixes forward and backward. One team member explained the situation as follows:

> Splitting the second release from the first was not pretty. Problems occurred because whenever you split, it's never late enough for the prior generic to have all the code there or early enough for the next release to have a base to do development on... Depending on when you split, you're going to have more or less mapping or more or less instability in the next release. When we split, we still had some late features being delivered to 1.0 plus a lot of bug fixes from testing at the customer site.

TABLE 4.1
MANAGEMENT APPROACHES BY INTERDEPENDENCY TYPE

	Management Approaches		
Type of Interdependency	Systemic Manipulation	Constrained Manipulation	Local Reaction
Product Technology (coordinating tasks related to the design, development, and testing of product)	• simple, layered, modular product design • parallel product and project structures • common functional task partitions • well-defined component interfaces; strict component ownership • target up front product design effort and approach to product complexity and stage of development cycle(fixed designs for core components, flex specs for user interface and peripheral functions, minimal changes at end) • daily product component integration performed internally by experienced technical people	• product design cuts across interacting subsystems • project design cuts across interacting departments • non-parallel functional structures • different task partitioning schemes across functions • integration sub teams in each department with fixed budget of integration time • "War meetings" • draw upon prior working relationships • "floater" role	• integral product design • project structured as small sub teams of peers with different organizational affiliations and geographic locations • different task partitioning schemes within a function • very detailed design documents written in large groups of peers • almost exclusive use of flexible spec documents • strict decentralized change control • infrequent product component integration, decentralized into sub teams • email as decision making forum
External Product Environment (coordinating tasks related to the product's use environment)	• prioritize and standardize external product environment • replicate product environment on site (hardware on site, labs, simulators)	• replicate product environment on site (test bed)	
External Project Environment (coordinating tasks across project team boundaries)	• develop core components internally • minimize number of externally supplied components and tasks • assign central contact point to each external partner; frequent pre-scheduled conversations • temporary collocation • "lift and modify" sharing	• temporarily redraw org'l boundaries (make 3 external partners into 1) • cultivate relationships with key partners • process standardization (common development platform, tools, and schedule); tiered • "marriage meetings" • slack in schedule and feature set • active status lists and touchable metrics	• core components developed externally • little process or interface standardization • slack in schedule and feature set • resolve conflicts across teams via informal rules and heuristics • newsletters

Resource Sharing (coordinating tasks that share resources)	• supervisors assume responsibility for minimizing resource blocking • "Wild Card" and "Umbrella" support roles	• "4pm stand-up meetings"	• slack in human resources (increase ratio of coordinators to developers)
Tasks Across Time	• avoid "patching" functionality on products via more frequent releases • assign maintenance tasks to current developers • build in time for renewal between projects		• map functionality and fixes across successive product releases

4.4.4 Mapping Management Approaches to Interdependency Type

The coordination techniques described above were effective at managing different types of interdependencies. Table 4.1 maps the techniques to the type of interdependency they primarily addressed.

4.4.5 Mapping Approaches and Interdependency to Project Performance

Although the data on project performance are incomplete and subject to error, it is possible to draw some tentative conclusions by looking across the three data sources.

Qualitative and Quantitative Data

Table 4.2 presents evidence of positive and negative performance outcomes for each project along four dimensions (schedule, software integration, quality, and team functioning). The data in the table represent extractions from the interview transcripts as well as project documents and tracking databases. It is, therefore, sparse, largely uncontrolled, and of questionable validity, although I took steps to verify it and only present evidence coming from more than one source. Nevertheless, certain preliminary results do emerge. The final column of the display captures the initial outcome assessment given this data.

TABLE 4.2
QUALITATIVE & QUANTITATIVE OUTCOME META-MATRIX: Product & Project Performance Outcomes

	EVIDENCE OF POSITIVE OUTCOMES & PERFORMANCE	EVIDENCE OF NEGATIVE OUTCOMES & PERFORMANCE	RESEARCHER'S OVERALL ASSESSMENT
DESK	Schedule: RTM date slipped from 6/30 to 7/15 Software Integration: Quality: Team Functioning: some tension betw apps and Desk (loss of autonomy; conflicting reqs) but balanced by high respect; pro-active, problem solving attitude	Schedule: Data slipped for several months, thus delaying Pro version Software Integration: "It sometimes took us days to figure out what's wrong with the build"; "Data is the outlier in terms of wanting to do stuff" Quality: "Majority of bugs were due to setup, which was very unstable" Team Functioning:	Some major problems but can largely be attributed to new product and org'l models: "Many interdependencies were hidden before when [products were produced] sequentially." Primarily localized problems (i.e., setup) Strong evidence that team members are aware of problems and have plans to address them in next release
DATA	Schedule: Software Integration: Quality: Team Functioning:	Schedule: product originally scheduled for shipment in 1996 but delayed to 1997 Software Integration: Yet to achieve a full product integration of all major components on an on-going basis Quality: sacrificed functionality from original product concept Team Functioning: little evidence of cohesiveness or cooperation; no clear leadership; "promises but no action"	Team members think in terms of components, not one product Major, on-going software integration problems On-going design and delivery issues with 4 external component teams
NETWORK	Schedule: product shipped on time Software Integration: daily and weekly builds very early in cycle Quality: 5 months after 3.51 release, 6 high priority repairs on cust sites Team Functioning: very strong team spirit and identity; proud andconfident attitude	Schedule: Software Integration: Quality: Team Functioning:	Few major problems for a product of this size and complexity Functioned effectively as a large team Some blocked component delivery due to higher priority project

	EVIDENCE OF POSITIVE OUTCOMES & PERFORMANCE	EVIDENCE OF NEGATIVE OUTCOMES & PERFORMANCE	RESEARCHER'S OVERALL ASSESSMENT
HAND-PHONE	Schedule: available on schedule when customer wanted it; 6 mo coding interval vs. avg 13 mo on comparable projects Software Integration: some problems at beginning, but quickly corrected after change to new process Quality: handover interval in 1/100 sec vs. requirement of <=1 sec; 17 common site scenarios passed first time; predicted total faults = 500 vs. actual 296 Team Functioning: "best working experience of my life"	Schedule: Software Integration: Quality: Team Functioning:	A highly successful project along a number of dimensions both objectively and in light of some major obstacles at the beginning (i.e., unassigned personnel, very aggressive schedule set by sales force)
TOLLPHONE	Schedule: Software Integration: Quality: Team Functioning:	Schedule: Software Integration: Quality: Team Functioning:	A legacy product (in U.S.) that has had to adapt to changes in technology and market
AUTO-PHONE	Schedule: Software Integration: Quality: Team Functioning:	Schedule: "trouble delivering features to customer on time"; three releases in progress while V1.0 delivered to customer 'on trial basis'; avg delivery interval 30+ mo and counting Software Integration: never able to consistently achieve integration of major components Quality: continually 'losing' functionality (intentionally and not) Team Functioning: "frenzied and chaotic"; "I feel I am part of 6-8 different teams"	No clear leader, ownership or accountability Major software integration and customer delivery problems Clear evidence of a project out of control

Network and Handphone both shipped on time and experienced few significant software integration problems. Data on product quality (as represented by the number of post release bugs) was extremely limited, but neither project appeared to have significant field problems. On Handphone, the number of pre-release bugs (all severity levels) exceeded estimates, but the testing on customer site was very rapid and problem free. The evidence on both projects is also very strong that team members found it to be a very positive working environment. This was particularly true of Handphone, which more than one person interviewed described as "the best working experience of my life."

The data likewise suggest a clustering of results for the Data and Autophone projects, which exhibited the local reaction strategy. There was little if any evidence of positive outcomes, but strong and consistent evidence of poor performance. Both projects experienced significant shipping problems; the Data schedule slipped one year while Autophone was still slipping at the last data collection point. Neither team was able to consistently achieve integration of product components and used similar adjectives to describe the process (i.e., "very unreliable," "integration hell," "a nightmare"):

> We've got some work to do to try and clean up the process and get integration off to a good start rather than having to churn through problems early on... Every time we do an integration, we seem to spend a week or two hacking at some critical problem before we can get things stable enough to perform testing.
>
> We've had to intentionally drop functionality between integration loads. And it's not at all uncommon to unintentionally drop things. Features and functions that used to work fine just don't anymore.

Finally, the data suggest irregular and chaotic team functioning on both projects with little member satisfaction in either the process or outcome.

For the Desk and Tollphone projects, the outcome and performance data are less readily interpretable. The Desk release date slipped approximately fifteen days (less than 5% of the total schedule), but this was largely attributed to the fact that another major product was going through manufacturing at the same time. Desk had a large number of very severe bugs before and after release, most of which were concentrated in the setup component. Data on the Tollphone schedule and bugs was not available.

Both teams had problems performing software component integration. On Desk, the application teams were working through issues of how to combine their processes. Tollphone experienced some integration problems early on, but these were largely resolved once the team converted to a new process. There appeared to be a moderate amount of conflict and chaos on each project, particularly Desk, but this was largely tempered by high experience levels, respect, and a problem solving attitude among the team leads.

We can tentatively concluded that, although Desk and Tollphone did experience some negative performance outcomes, it is not entirely appropriate to classify them as poor performers along with Data and Autophone, for two reasons. First, the interviews suggested that team members and managers were quite aware of the sources of the problems and often had plans in place to rectify them in the future. For example, the Desk group was already working on how to reorganize the development of the setup component in the next release. Second, as noted earlier, both projects were undergoing significant transition in their product architecture and product market, which required them to change their processes. The Desk project was trying to integrate five previously independent applications and converting from an unintegrated, asynchronous product release to annual integrated shipment. The Tollphone project was almost moving in the

opposite direction-- from annual or biannual system release to continuous streams of smaller functionality. Table 4.3 documents these transitions.

TABLE 4.3

TIME-ORDERED DISPLAY: Longitudinal Changes in Product Design & Release and Project & Process Design

CASE	PERIOD ONE		PERIOD TWO			
	PRODUCT DESIGN & RELEASE	PROJECT & PROCESS DESIGN	PRODUCT DESIGN & RELEASE	PROJECT & PROCESS DESIGN	KEY FACTORS DRIVING CHANGE	KEY ISSUES AND IMPLICATIONS
DESK	product specific components and manuals app-driven release; apps ship separately followed by Desk	indep cross-functional app teams; Desk integration team indep processes with "varying degrees of rigor" emphasis on "per product loyalty and focus"	about 50% shared components; one common Desk manual; large number of externally supplied components annual Desk-driven release; simulanteous app shipment; releases alternate minor/major functionality additions	Desk product team with cross functional application subteams; dedicated integration teams on each app; one documentation team common scheduling and integration processes; common definitions and criteria at milestone points Desk focus	Market: ~70% of app sales come thru Desk Customers: demand commonality and consistency across apps Technology: new tools like OLE enable component sharing Efficiency: less redundancy of effort; "implement a feature once, maybe 1.5 times, and everything over that is gravy" Specialization: focused expertise for component development	Loss of Autonomy: "I spend all my time supporting Desk not app features"; the "Desk tax"; fear that Desk "will take over" Increased Overhead: "takes more people to do same amount of work"; more time checking and "stone turning" Clash of application group practices, priorities and cultures: "a set of conflicting forces pulling you in different directions" Increased Schedule Risk
TOLL-PHONE	system organized into 50 functional subsystems; fairly localized features biannual shipment releases (100% of subsystems released at same time, customers get all of them); "big bangs" of functionality	"large org implemented large features with developers working in small number of modules" multi-functional department product integration performed internally	90% of features delivered as customized products new features introduced as patches or bug fixes continuous streams of functionality	"smaller org implementing smaller features with developers working in larger number of modules" functional depts (field support, doc, load building, project planning, RC/Data in different departments) product integration performed by external support organization	Competition: increased time to market pressures in long distance service Customers: calling needs/reqs differ by country (e.g., Sri Lanke autom ends int'l calls > 10 mins)	Feature delivery out of synch with the base project Multiple (increasingly divergent) product versions running around the world

Reported Product and Process Problems

As described in the methods section (Appendix D), the projects were also classified according to the type and severity of their self-reported problems. The project clusters derived from this analysis largely support the conclusions drawn above (Table 4.4).

Like most large projects, Handphone and Network experienced some minor problems (coded M on Table 4.4), which affected small numbers of people. Handphone also experienced significant blocking early on due to architecture uncertainty and their inability to gain hiring approval from upper management, but these were eventually resolved (coded L). Major ongoing problems were limited to two areas (coded X). Handphone shipment was almost delayed at the very end when another project sharing a common header file broke their code. Network under went several temporary delays because it was sharing certain features with another product or depended on a common component team also serving that product.

Autophone and Data, in contrast, are shown to have had major on-going problems in three to five areas (coded X). Product component integration was the most severe, but others included gaining consensus on a common development process, product delivery planning, and component sharing. Finally, the data on Tollphone and Desk are again mixed. Both projects experienced minor and major problems in several categories (coded X and M). Desk, in particular, appeared to have plans in place for resolving some of these.

TABLE 4.4
SUMMED INDICES MATRIX: Reported Product and Project Problems

Types of Problems

	Product Architecture Uncertainty	Vague, Missing or Incomplete Reqs & Design	Cross Project Design & Code Sharing	External Team Component Delivery	Common Component Team Dependence	Coord Betw Development & Testing	Coord Betw Software Dev & Data Dev	Coord Betw Development & Documentation	Impact of Design Inconsistency on Downstream	Product Component Integration	Defining a Development Process & Env	Product Delivery Planning	Configuration Management	Shared Machines	Shared Lab Facilities	Shared Manuf Capacity	Testing Tool Inadequacy	Finding & Fixing Bugs	HRM Blocking	Coord Betw App & OS
HANDPHONE	L	M	X	M	M	M	M			Q									L	
NETWORK		M	X	M	X							M						M		
AUTOPHONE										X	X	X					M			
DATA			X	X	X				M	X	X	M								
TOLLPHONE		M	X			M	M			M		X	X				M			
DESK			P	X	X			X	X		P	M	P	M		M		M		M

X = Major, On-going Problem (threatened shipment; >= 1 month delay; high citation rate

L = Major Problem, Lengthy Resolval Period

Q = Major Problem, Quickly Resolved

P = Major Problem, Plan Exists to Resolve in Next Releas

M = Minor Problem (minor inconvenience, affected small number of people)

Blank = No Problems Cited

Questionnaire

Appendix F describes the project performance survey. The response rate from internal and upper level managers was too small for reliable estimates (less than 5%).[46] However, the project rankings by internal experts matched those found from the other two data sources.

4.5 Discussion

Interdependency Management and Project Performance

The first goal of this research was to document teams' strategies toward multiple interdependencies. I was able to differentiate three different approaches and to monitor the actual techniques associated with those strategies. Table 4.5 presents a summary of the strategies and performance outcomes.

TABLE 4.5
TEAM DIFFERENCES IN STRATEGIES AND PERFORMANCE

Strategies	Teams	Performance
Systemic Manipulation	Network	High
	Handphone	High
Constrained Systemic Manipulation	Desk	Medium
	Tollphone	Medium
Local Reaction	Data	Low
	Autophone	Low

The study's key proposition is this: new product development teams which adopt a systemic manipulator approach to interdependency management experience fewer problems and achieve better project outcomes because they manage interdependencies within and across types and across time. Like spiders weaving a

[46] The response rate is low because some team members had transitioned to a new project. The delivery format of the questionnaire (email) also turned out to be inconvenient for many people (readability problems). Finally, one of my contacts failed to deliver some of the questionnaires.

web, such teams create their own "web of interdependencies" by the up front choices and decisions they make. Similar to the finding by Brown and Eisenhardt, an important distinguishing factor of these projects was therefore "not how they reacted, but how they acted... [how they] anticipated, built options, and changed the game, rather than just playing along" (p. 32) (Brown & Eisenhardt, March 1995).

The systemic manipulator teams, Network and Handphone, effectively created their own interdependency landscape through key up front design decisions. They coordinated tasks not merely via people and processes but also by the decisions they made at the level of the product (e.g., Network and Handphone used a modular product architecture with well-defined inter-component interfaces), the environment (e.g., Network prioritized interfacing products in its environment), the product strategy (e.g., Network decided to not perform extensive maintenance or configuration management), and the organization (e.g., Network and Handphone chose to invest in key physical and human resources). The structures and processes these teams used tended to promote a system view of both the product and the team and facilitated product component integration and team member cooperation.

Handphone and Network also prioritized interdependencies and gained leverage off of central tasks such as product component integration. They made heavy use of incentives to increase people's awareness of the effect their work had on others, most notably via a product component integration process. They sought and applied solutions for coordinating systems of multiple interdependencies in a relatively efficient manner. For example, they replaced duplicate individual coordination mechanisms with more centralized team solutions (e.g., Handphone "Wild Card;" Network and Handphone product component integration process) and thus freed up people's time and attention for production activities. Finally, both teams demonstrated some

awareness of the future task coordination implications of decisions and structures made today.

Yet systemic manipulation can be constrained by legacies of prior product and organizational structure. A further proposition emerging from this research is that interdependencies can easily become technically and organizationally embedded. Other researchers have observed that technological and organizational designs and strategic decisions often evolve over time in a relatively uncoupled fashion (Gulati & Eppinger, May 28, 1996). As a result, projects face constraints and task situations created by past structures and strategies which can be very difficult to dismantle. The problems Desk and Tollphone experienced were as much a reflection of their past decisions and coordination structures as their present. In effect, interdependencies in time one became formalized at time two in the form of organization and product structures, which in turn dictated new interdependencies at time three. Viewed this way, the process of organization design becomes one of managing within and across interdependency types as well as interdependencies across time.

In the case of Tollphone, an embedded technological product architecture created a complicated set of task relationships during project start-up, development, and testing. Although team members were clearly aware of the problems and knew how the product architecture should be re-aligned, they were prevented from doing so due to prior customer relations. In effect, Tollphone under leveraged by allowing excessively wide gaps to occur between major re-architectures of the product (Brown & Eisenhardt, May 1995). Ironically, the past was also the source of a solution in this case. Friendship networks created in prior organizational structures enabled the team to work despite functional and divisional barriers.

On Desk, the legacy was more one of organization. Prior project structures were designed to promote an application product focus. As the level of integration in Desk increased, it created a set of interdependencies across the applications, which required formerly independent team members to work together. Desk instituted special project structures, most notably a "fixed budget" integration team on each of the application groups, but also, like Tollphone, relied upon aspects of past working relationships, namely high mutual respect, to temper the level of conflict across the teams.

A final proposition emerging from this research is that teams that approach interdependency management in a localized, reactive way will perform poorly. Data and Autophone failed to prioritize either interdependencies or solutions and implemented coordination techniques in a relatively piece meal fashion. These local reactors rarely made the link between, say, the design of their product architecture or their product strategy, and task coordination. Instead, they resorted to yet more meetings, task forces, people, or redesigned processes. The result was a multitude of conflicting structures, all of which absorbed coordination time and effort. Like flies trapped in a web, these projects found themselves thrashing around in a tangled set of task relationships.

Both teams tried to integrate different components developed by different teams into one working product. The result was a bunch of "spinning wheels," uncoordinated tasks, and conflicting as well as duplicated coordination mechanisms. They used few if any mechanisms to increase the visibility of interdependencies beyond a "near neighbor" level.[47] Instead, they relied on broadcast mechanisms such as newsletters but

[47] In fact, none of the projects studied had good techniques for visualizing the complete "web" of interdependencies and thus was unable to completely describe the near and far neighbor relationships. The systemic manipulator cases appeared to worry more about the web problem--they wanted and actively sought tools to manage it from research.

failed to design incentives for paying attention to such sources. Nor did they design mechanisms for raising people's awareness of others dependence on them. Individuals, therefore, felt little if any pain when they changed their work and impacted others. The amount of time and effort people on Data and Autophone devoted to coordination was overwhelming in both a psychological (tended to produce feelings of frustration or apathy) and productive sense. Finally, similar to findings by Brown and Eisenhardt, Autophone over leveraged successive product generations over time, resulting in an "increasingly complex and steadily more incomprehensible product" (Brown & Eisenhardt, May 1995).[48]

Techniques

Design researchers have long argued that by configuring the product architecture in such a way as to minimize the interaction among subsystems, task coordination will be simplified (Alexander, 1964; Allen, November 1986; Clark, 1985; McCord & Eppinger, August 1993; Simon, December 1962; Ulrich & Eppinger, 1995). The cases presented here substantiate the importance of that technique. By focusing on critical up front product design decisions, systemic manipulator teams were able to simplify task coordination.

But these cases also suggest an important complication. Namely, the relevant unit of modularity (in terms of what the market values) can change over time, highlighting the importance of regularly rearchitecturing the product (Brown & Eisenhardt, May 1995). Originally, Lucent modularized its telecommunications systems in terms of functional subsystems. This made sense when the market bought system products, but now the relevant unit of modularity is a piece that cuts across subsystems.

---

[48] Note the contrast with Tollphone, which under leveraged across generations.

Project structure has likewise been shown to be a powerful means of coordinating work (Bowditch & Buono, 1985), and scholars today are increasingly advocating ever new (and more complicated) forms. The results of this research go against that prevailing trend. They suggest that a relatively simple structure is best. More important than any particular form, however, is the existence of a certain parallelism of structure (between product and project as well as across functional group structures). In the cases, we saw how complicated coordination becomes when this is not done because people don't know who to talk to (e.g., Desk developers and documenters, Handphone developers and testers).

Likewise, tasks within a given structure should be partitioned along a common paradigm (Von Hippel, 1990). This is relatively easy to do within functions (although Autophone didn't) but can be very difficult to achieve across functions (i.e., development and documentation). This research, therefore, supports literature which has identified task partitioning as a powerful coordinating mechanism but also highlights a fundamental assumption and complication obscured in that previous work- - that there exists a single partitioning scheme. McCord & Eppinger likewise found that partitions are not neat. They required overlap due to multiple dimensions of interdependency (McCord & Eppinger, August 1993).

If there was one "silver bullet" for managing multiple interdependencies that emerged from the study it was product component integration. A key proposition is, therefore, that projects can derive leverage off of designing this central process correctly. While other authors have previously written about the importance of component integration in product development (Cusumano & Selby, 1995; Iansiti, 1994; McCord & Eppinger, August 1993; Ulrich & Eppinger, 1995), the emphasis in this prior work has primarily been on integration frequency. The earlier and more frequently

components are integrated, the better in terms of project performance. The present study supports that conclusion but also offers an extension to theory by suggesting other benefits and key process design decisions (i.e., increased cooperation and easier bug fixing when process is performed internally by highly experienced technical people; cooperation incentives targeted to team). An interesting question for future research is whether integration frequency alone is sufficient to coordinate interdependencies or whether these other design factors also make a difference.

Finally, Dougherty and Bowman describe how firm down sizing (defined as restructuring and/or elimination of staff) affects a firm's ability to develop new products by breaking informal networks and increasing risk aversion (Dougherty & Bowman, Summer 1995). The cases described here substantiate that firm-project linkage but also suggest an additional variable, namely resource investment and allocation decisions.

Contributions

This research presents a more realistic view of interdependency and provides some evidence of where it comes from and why and how it can get more complicated over time. As such, it can be viewed as a response to Brown and Eisenhardt's recent call for more approaches that open up the "black box" of product development by providing depth and rich understanding of how firms actually develop products (Brown & Eisenhardt, April 1995).

The study shows how a complex technical product and a complex organizational system of actors performing tasks necessary to produce that product interact. It supports prior empirical work but also clarifies the linkage between coordination problems and project outcomes by showing the implications at the task level. As a

result, we are better positioned to design organizational solutions for improved performance.

CHAPTER V: Summary and Conclusions

This chapter summarizes the dissertation and discusses the implications of the research for theory and practice. It also reviews the major limitations of the methodology and suggests some directions for future research on this topic.

5.1 Summary of Approach and Findings

This dissertation focused on the concept of interdependency in organizations. The primary research questions were: what types of interdependency exist in complicated production environments, and what are the consequences of multiple types for the organization of work? The investigation took place in the context of large scale software product development, which is characterized by very high levels of task contingency. The analytical approach was that of induction and grounded theory building (Glaser & Strauss, 1967) using case study projects. The key contribution to theory is the identification of a need for a multiple interdependency perspective in organizational design.

The dissertation began by reviewing past empirical and theoretical research on interdependency across three theoretical paradigms. This review served to motivate the subsequent empirical investigation. In particular, it illustrated both the breadth of the concept (interdependency appears as a constant theme across time and disciplines) but also a lack of integration across paradigms and even studies within a single paradigm. A multi-dimensional framework was proposed, which attempted to make sense of this diversity.

The empirical part of the dissertation was broken down into two studies. The first used examples of interdependency from large scale software development to inductively derive a taxonomy of different types. Four primary types emerged, which

can be traced back to the product technology, external product environment, internal work unit environment, and firm-level structure and resource policies. Interdependencies were also shown to vary in terms of their predictability and task structure.

The second empirical study examined interdependency management in six large scale software development projects in two firms. Drawing upon a rich set of qualitative data at multiple levels of analysis, it sought to identify the strategies and techniques teams use for coordinating different types of interdependencies and form hypotheses about their normative implications. The study documented three different strategies, systemic manipulation, constrained systemic manipulation, and local reaction. It further proposed that a systemic manipulation strategy will produce the highest project performance and isolated a small set of coordination techniques that seem to be highly influential and effective (i.e., up front design decisions, parallel structures with tasks partitioned along a common paradigm, minimal external interdependencies, product component integration etc.).

5.2 Research Limitations

All research is characterized by limitations and biases (Cook & Campbell, 1979), and this study was no exception. First, aspects of the sample may limit the generalizability of the findings. The dominance of the two firms in their respective industries creates the possibility of idiosyncratic results where, for example, Microsoft or Lucent is better able to alter aspects of the product architecture because they control certain standards. Also of concern is the comparability of the results across the two settings. Features and functions in the telecommunications industry are typically developed as customized products to unique specification and requirements. PC

software products, in contrast, are more of a mass market product. Some of the strategic solution differences may reflect this distinction.

It is also important to note that the participating projects were limited to software technology. This was deliberate and reflective of the goals of the study.[49] At the same time, the types and frequencies of interdependency observed as well as the solution strategies may be limited to similar types of settings. It remains for future research to examine how much of the results apply to hardware projects or even outside the product development setting.[50] On the other hand, the fact that more and more hardware products are starting to resemble and incorporate software suggests that the results may be very applicable and even indicative of problems likely to be encountered more generally in the future.

A second limitation of this research arises from the nature of the analytical process. The interdependency types, solutions, and approaches identified here were not randomly selected nor rigorously measured. Rather, they emerged from the analysis as part of an inductive conceptualization process. Although attempts were made to apply rigor to the analysis, by, for example, displaying the data in a variety of formats and routinely challenging developing hypotheses, the results are subject to many limitations and some biases. Future research will have to test the propositions.

Finally, it is possible that a project could manage interdependency very effectively and yet still exhibit poor performance outcomes due to the existence of other

---

[49] In particular, the primary research question (what types of interdependency exist in large scale development?) dictated choosing a relatively small, highly variable set of projects. This sample size and high variance, however, presented problems in terms of drawing conclusions about the relative effectiveness of different interdependency management solution strategies.

[50] Informal conversations with researchers working in other domains suggest that the results may be quite generalizable (e.g., lean aircraft initiative, drug delivery process).

factors. Project outcomes may also influence as well as be influenced by interdependency processes, an issue associated with the ambiguity of causal ordering (Cook & Campbell, 1979). Thus, performance outcomes may not be reflective of interdependency management at all.

For example, although the Handphone project appeared to manage interdependency very well, it was also the smallest project in terms of amount of new code developed. How much of the success can be attributed to the reduced complexity? Similarly, were the chaos and inefficiency observed in the Autophone project indicative of poor interdependency management or merely a reflection of working in an emerging highly competitive arena?

The above factors argue for modeling performance at the interdependency level of analysis rather than the overall project performance level, as was done here. Such a modeling would provide a tighter link between interdependency process and outcome. Unfortunately, this study was unable to make that connection due to data unavailability.

Instead, attempts were made to control for project variance and external factors ex post, but again this was complicated by differences across companies and projects. For example, the two firms use different code counting conventions, making it difficult to control for product size and complexity. Staffing and work hour norms also differ, as do accounting practices. I compared evidence across data sources and, in particular, relied on the rich contextual qualitative data I had access to in an attempt to elucidate such factors. Consistent with the overall inductive approach, the results of the analysis are hypotheses about the effectiveness of different interdependency management

techniques and their implications for project performance, but the verification of these hypotheses awaits future research.

5.3 Implications

This study has both managerial and theoretical implications.

Implications for New Product Development Managers

For managers, this research presents interdependency as an element of project strategy that needs to be managed over time. Key questions managers need to think about are: (1) What interdependencies really matter right now, given the need to meet the multiple goals of speed, quality, and leverage? and (2) How may today's choices and decisions create a future 'web' of complex task relationships?

The multiple interdependency framework encourages managers to differentiate between different types of interdependencies and, therefore, facilitates targeting of approaches. The strategic taxonomy-- both the identification of key organization design decisions and the listing of alternative approaches-- should enable managers to actively create their own set of interdependencies. The taxonomy can further assist managers in problem diagnosis, assessment of change, and setting of project priorities before and during project execution.

Many firms are experimenting with distributed, multi-functional teams as a possible way to achieve multiple goals in new product development. The results of this research study do not bode well for that trend insofar as the two distributed projects studied (Data and Autophone) found it to be very, very difficult to manage. Yet this research also identified some techniques managers can use to potentially facilitate task coordination across distances. For example, if the Data and Autophone teams had been able to standardize their processes and development environment across locations and

had parallel product-project architectures with aligned task schemes and a good product component integration process, would the results have been different?

Implications for Theory

A paradigm of multiple interdependencies makes several contributions to organization theory. First, it explicitly recognizes the multivariate nature of work by taking into account the pattern of interdependencies as a whole. The result is a more sophisticated and complex view of what interdependence can mean (Pennings, 1975). An indirect benefit of having a thorough, concrete description of interdependency patterns in product development projects is that it will stem the counterproductive tendency to say that "the number of interdependencies is infinite" or "everything depends on everything else." If we can be precise in defining and drawing boundaries around relationships, then the number is likely to be much less than are imagined (Eppinger, Whitney, Smith, & Gebala, 1994; Weick, March 1976).

A second important contribution of this research agenda is that it treats interdependency as a variable (in terms of both time and dimensionality) rather than a constant (Weick, 1974). It explores the multifaceted nature of interdependence as well as how various types of interdependence may be related or unrelated to one another in complicated ways.

Other researchers have described organizational design as "a process filled with contradictions and dilemmas" (Pfeffer & Salancik, 1983). The model presented here contributes to our understanding by making the conflicts between and among interdependencies explicit. This study, therefore, suggests an alternative conceptualization of the organization design process, one that is more strategic and

explicitly acknowledges the inherent tradeoffs and conflicts created by multiple interdependencies as well as the link between decisions and tasks.

The act of organization design is not limited to simply imposing or "matching" structures, processes, and mechanisms on different interdependencies in isolation from one another. Doing so can create duplication of effort and coordination conflicts such that the design solution imposed to manage one type of interdependency has negative repercussions or conflicts with other design solutions needed to manage other types of interdependencies. Rather, tasks can be designed to highly interdependent (by, for example, requiring the input of several people) or work can be structured to be highly independent (if it is partitioned and assigned as such) (Eccles & Nohria, 1992; Wagerman, 1995).

The skill of organization design is one of coming up with sophisticated yet relatively efficient ways of designing to an overall pattern, attempting to minimize misfit in that pattern (as opposed to achieve fit in a single element of it), and maintaining some degree of flexibility in structures and interdependencies. When an organization (or project) can do this, it becomes a source of strategic advantage.

5.4 Future Research Directions

If nothing else, this research will hopefully motivate other research on interdependency and encourage scholars to think more critically about their use of this variable. Several opportunities for future work are also worth noting.

First, there is a need for further investigation of the multiple interdependency concept in other settings. Are multiple interdependencies an issue in other

technological environments? In other types and sizes of firms? Are the solutions identified here generalizable to those settings?

Second, the implications of interdependency management over time need to be explored. This study supplied evidence of the importance and difficulty of managing interdependencies over time, but the study was cross sectional in nature. We need to conduct longitudinal studies of interdependency management, which track the interaction between tasks and structure.

Third, this research focused primarily on the structures and processes used to coordinate multiple interdependencies. We also need to understand the affective and behavioral processes elicited when working in highly contingent settings. Armed with knowledge of structure and affect, we will be better positioned to design coordination solutions to manage work relationships.

APPENDIX A
Social Science Citation Index Analysis[51]

One question the literature survey in chapter 2 does not address is the relative relationship among the three theoretical perspectives on interdependency. For example, can they be traced back to a common source article or do later perspectives represent branches off of an earlier line of research? How has the relative balance and influence of research within and across different perspectives varied across time? Such questions are beyond the scope of this research, but I present some preliminary evidence and suggest ways we might address them here.

Informal analysis and conversations with scholars suggested that people most often associate Thompson's 1967 article with the concept of interdependency. This conjecture is supported by data from this survey. I reviewed 93 interdependency references in total (54 information processing, 26 resource-based theories, and 13 sense-making theories).[52] Eighty-six of these were published after 1967 (49, 24, and 13, respectively), of which 31 (36%) referenced Thompson.

Next I conducted a forward search of references to Thompson using data from an electronic bibliographical index. The Social Science Citation Index (SSCI) is an electronic database of citations in journals across a broad span of the social sciences (e.g., business, management, political science, sociology, etc.). Source articles are available since 1972.[53] An electronic search of the database in March 1997 for all references to Thompson (1967) revealed 3145 'hits.' Figure A.1 shows how these citation

---

[51] Thanks to MIT Sloan School Senior Associate Reference Librarian Kate Pittsley for valuable help in performing this search.

[52] Because the articles were not randomly selected, we cannot draw any conclusions or generalizations from this distribution.

[53] In a confusing bit of nomenclature, 'source article' refers to the journal publication that contains a reference to the target article, in this case (Thompson, 1967).

references were distributed across the years 1972 (the year the database began) through 1996 (the last full year at the time of this study).

As seen in the histogram, the rate of citations has remained fairly steady since 1978, averaging about 130 per year during that interim. Figure A.2 shows how the 3145 references are distributed by journal subject category. Not too surprisingly, business and management journals account for the vast majority of the references (79%), but Thompson's work has also influenced other fields as diverse as sociology (9%), finance (4%), and computers (2%).

It is important to note some potential problems with this analysis. First, the database itself is far from perfect. Source articles only include original journal publications, not reprints. Nor can we be sure exactly how or why the reference is made, other than by going back to the original articles (the database only contains the reference citation and occasionally a brief abstract). In particular, Thompson's book contained a number of other ideas that have been highly influential in management theory.

A more significant problem is the lack of a standard reference format. Some authors use two initials (J.D.) while others use only one (J.). The year (1967) is usually included but occasionally not, and misspellings are rampant. Thus, a single search may not reveal the full universe of source articles. Recognizing this, I performed the search on four major variants: Thompson-JD-1967-Org*, Thompson-J-1967-Org*, Thompson-1967-Org* and Thompson-Org*.

Another potential source of bias is the fact that the set of source journals in the database has not remained constant over the years (nor is it the full universe of potential

journals). This bias would be reflected in Figure A.1 in the form of artificially high (low) peaks (valleys) as source journals enter (exit) the database. In Figure A.2 certain subject areas are likely to be under-represented due to the social science orientation of the database. For example, searching a scientific citation index like Applied Science and Technology (AST) would likely reveal many more computer science source references to Thompson (1967) than what was found here simply because AST accesses more computer science journals.

Despite these somewhat significant limitations, this citation analysis is useful in that it substantiates the fundamental impact of this key reference and, therefore, the centrality of the concept of interdependency.[54] It is also interesting that the reference rate has remained fairly constant over the years, suggesting that this is not some 'fad' but rather more of a perennial concept in organization theory.

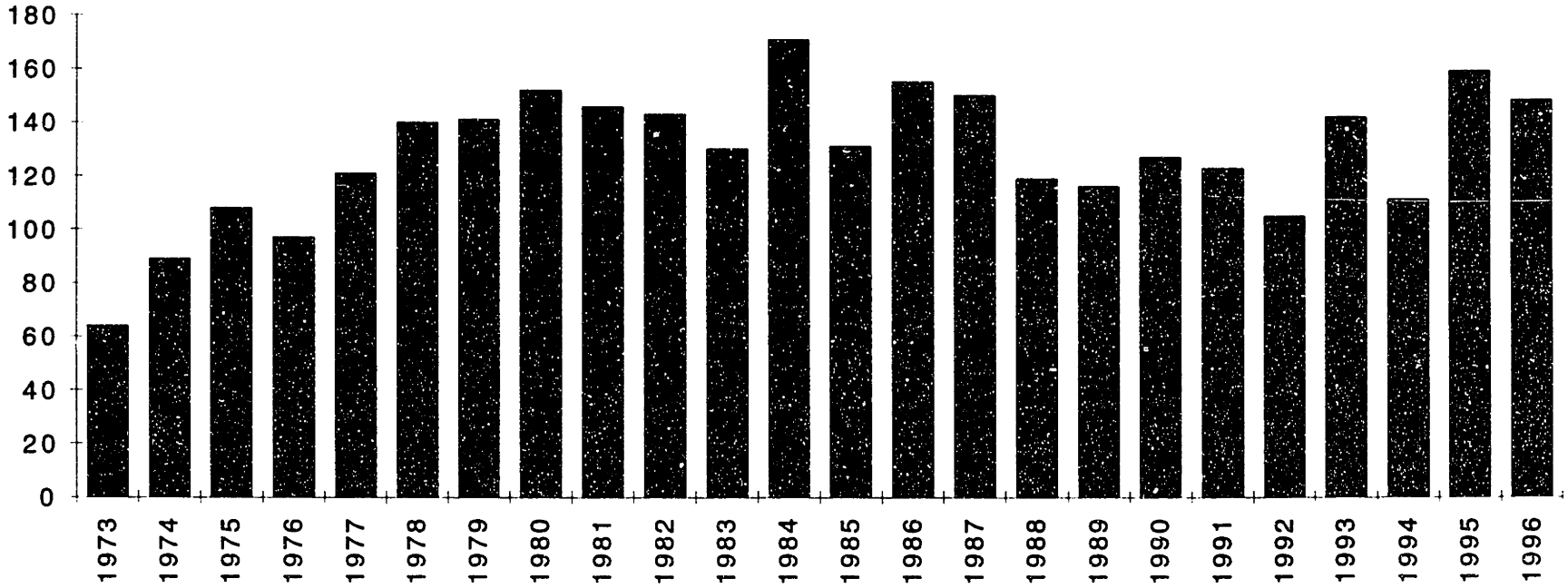
One future approach would be to take the 3145 (or perhaps the subset of 2487 from business and management journals) and analyze the content of their abstracts or titles for key words. Such data could be used to confirm the existence of the three paradigms proposed here, analyze their distribution and evolution over time, or suggest alternative categorization schemes.[55]

---

[54] The librarian who helped me perform this search, who has had extensive experience in this area, has never encountered anything close to this high of a hit rate.

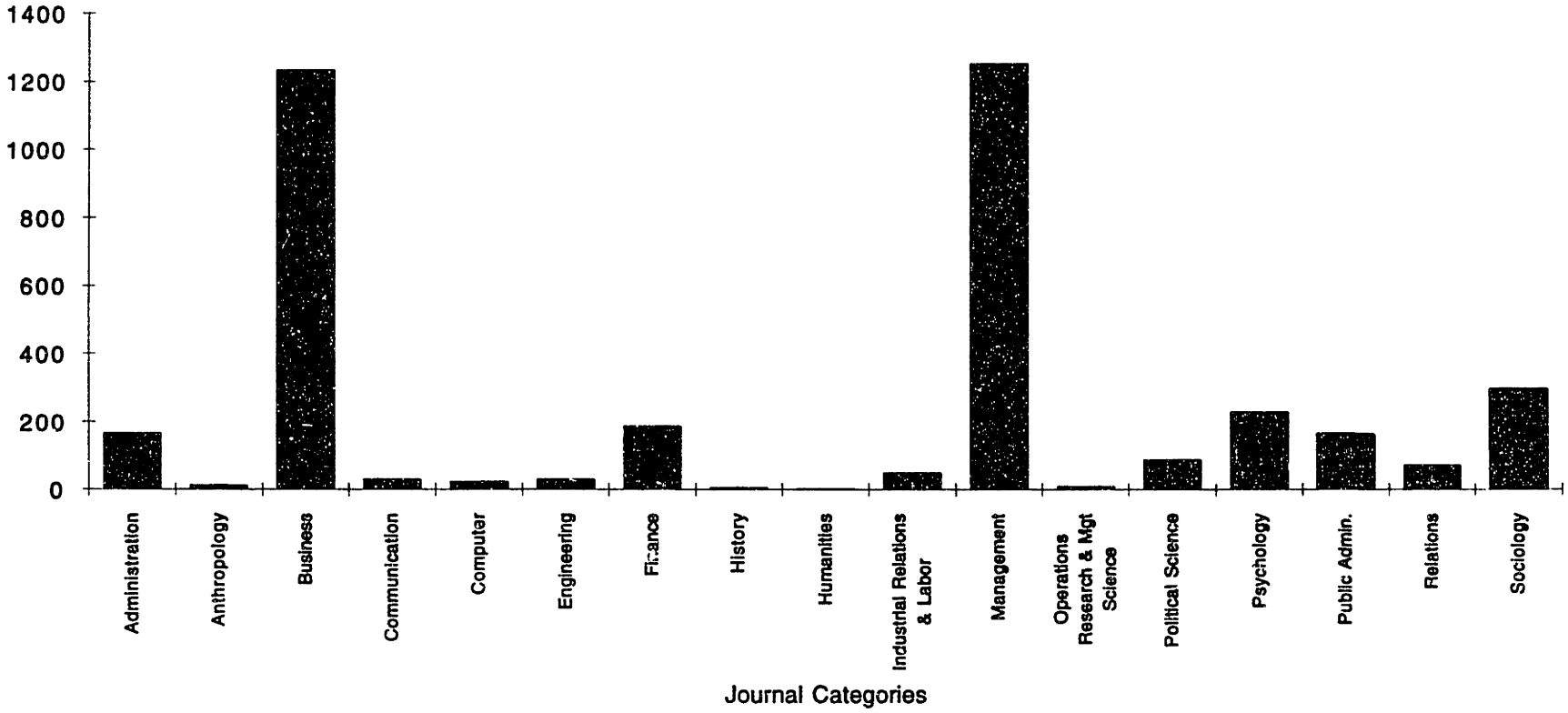
[55] Unfortunately, this was not possible in this study. The SSCI is extremely expensive to use for anything more than simple keyed-in requests; downloading all 3000+ articles would have been prohibitive. Instead, I collected a variety of references over several years and analyzed their content as described in chapter 2.

FIGURE A.1
REFERENCES TO THOMPSON (1967), 1972 - 1996



Source: Social Science Citation Index

FIGURE A.2
REFERENCES TO THOMPSON (1967) BY JOURNAL SUBJECT CATEGORY, 1972 - 199



Source: Social Science Citation Index

APPENDIX B
DIMENSIONS OF THE THEORETICAL ANALYSIS

The following are the dimensions used to compare the three theoretical perspectives in chapter 2.

THEORETICAL STRUCTURE:
What are the basic underlying assumptions of this perspective in terms of the nature of technology, organizations, etc.?
- Ontology- is reality objective and external to the individual or the product of individual consciousness? Can we make sense of an organization by measuring it or is it necessary to consider the way people experience it by studying their perceptions and interaction with the world (realism - nominalism)
- Epistemology- is knowledge real/hard or subjective/soft? (positivism-antipositivism)
- Human Nature- do people respond mechanistically to their environment/are they conditioned by external circumstances or are people depicted as active, creative, free willed entities? (determinism-voluntarism)
- Methodology- is the theory searching for universal laws or trying to explain and understand by getting close to the phenomenon? (nomothetic-ideographic)
- Model- does the model primarily focus on conditions/identify factors responsible for a particular outcome or does it focus on dynamics and the how and why? (factor/variance-process)
- Causal Agency- who or what is depicted as causing change?

technical imperative- technology is an objective, external force what has a deterministic impact on organizations (situational control, technology is an i.v.)
organization or strategic imperative- technology is the product of on-going human action (rational actor control, technology is a d.v.)
sociotechnical- technology is physically constructed thru social interaction and political choices of actors
social constructionism- shared interpretations around a technology arise and affect its development
marxism- technology is used to further the political and economic interests of powerful actors
technology is a social object- technology is a moderator variable between human agents and organizations; its physical form and function may remain the same, but its meaning is defined by its context of use

CONCEPTUALIZATION OF INTERDEPENDENCY:
How do authors in this perspective conceptualize interdependency? What do they conceive of as the sources and consequences of interdependency? What factors affect the level of interdependence?
- Is interdependence an independent variable or a dependent variable?
- What are the antecedents and consequences of interdependence?
- What dimensions or constructs affect interdependence?

- What are the implicit assumptions about interdependency?

STRENGTHS AND WEAKNESSES:
What aspects of interdependency does this perspective emphasize or highlight? What does it ignore or neglect?

OUTCOMES OF PERSPECTIVE:
What sort of managerial recommendations with respect to interdependency does this perspective lead to? What are its implications for organizational design and team affect or behavior?

APPENDIX C
AFFINITY DIAGRAM METHODOLOGY

This appendix describes the steps used to categorize the interdependency examples and construct the affinity diagrams shown in Figures 3.2a - 3.2d of Chapter three.

Broad Thematic and Factual Analysis

Before attempting to classify the interdependency examples, I began with a relatively unstructured review of the data. Initially, I focused on one project at one company and read through all of the interviews and documentation I had collected on it. I made notes in the margins of the transcripts and documents and looked for broad themes about interdependency.

Gradually, I expanded this analysis to include the other projects at this firm and ultimately the other company. The result of this review was the identification of nine general themes about interdependency in new product development (Figure C.1). Note that the concept of multiple interdependencies first emerged at this point.

Simultaneous with the above analysis, I wrote case summaries of each company and project which followed a fairly structured outline. The goal here was to simply get the basic facts about each firm's technology and organization down while the information was still relatively fresh in my mind. I was also able to identify certain areas where my notes were confusing or inadequate (i.e., the system technology architecture at Lucent) and obtain supplemental information while it was still relatively easy to do so.

With that foundation, I was ready to focus on interdependency as a unit of analysis. Below are the steps I followed to do so and subsequently cluster the examples.

Classification of Interdependency Examples

Step 1: Gather Raw (Qualitative) Data

The original data consisted of 71 interviews across six case study projects. The data collection was described in Section 3.2.2 in the text of Chapter three. The unit of analysis was task interdependency, so my first step was to reduce the interview transcripts to a briefer, more manageable form. I used several methods to accomplish this. First, I concentrated on extracting the examples of interdependency from each interview. Rather than adopting strict a prior definitions, I applied a very broad and inclusive criteria at this point, essentially searching the interviews to pinpoint examples of contingent relationships.

Every example was assigned a three digit ID code. The first digit identified the project, the second digit identified the functional role of the speaker, and the final place holder was a number which uniquely identified each example in the interview text. For instance, code 'PHdat89' referred to an example cited by a data engineer on the (Lucent) Handphone project; it was demarked by 89 in the text margin. (Obviously, some examples were duplicates. I describe how I dealt with these in step 3 below.)

The nature of the examples varied widely. Some were relatively simple and straightforward. For example, "Development depends on marketing for the product vision" (NTmgt1). More often, they were relatively extensive and detailed descriptions encompassing the nature of the interdependency, the entities involved (activities, people, projects), as well as affective reactions and consequences:

> We discovered that if the cache manager could adjust weights dynamically it would be more efficient. So the cache manager developer and the owner of the weighting part of the system negotiated an interface. That's a dependency that just sort of materializes out of thin air because of a need and is often tracked very informally... [That interdependency] creates problems for a third party because when the cache manager weights pages, all seven file systems need to be modified to pass on some weights... Making a global change like that really impacts a lot of people. We therefore need to evaluate it- is it worth with? Is the performance benefit really that important? [What are the test implications?]" (NTmgt25)

As indicated in the above example, several interdependencies were often intertwined in one statement. In those cases, I broke the example down to its most basic elements and reassigned sub codes as necessary. An example could also be quite technical. I referred to product architecture diagrams, process information, and supplemental interviews to both understand the technology and decipher acronyms and phrases used uniquely in each firm.

To help preserve the literal completeness of the examples and forestall premature closure on a model, I proceeded to condense and standardize the format of each example in several steps. First I isolated each of the examples along with their margin comments and enough surrounding material to provide a context for the statement. I then condensed the original statements down somewhat by focusing in on key sentences or phrases and eliminating filler language (e.g., "umm") while nevertheless preserving as much of the subject's original wording as possible. The result at this stage was a concise list of fairly literal descriptions of what each subject said about the respective example.

Step 2: Interpreting Raw Data in Terms of Interdependency

Several factors complicated the next stage of the analysis. First, subjects do not always express themselves in a manner that is convenient for theoretical analysis. In

this case, team members talked about interdependencies in many different ways. Some spoke in terms of their interdependence with other people; others used the technical language of a product design. Yet for the purposes of organization design, interdependency should ideally be described at the task level (Malone & Crowston, March 1994). I therefore needed to abstract the interdependency from each description and translate (many of) the examples into a relationship among tasks.[56] Furthermore, people sometimes used emotionally charged terms (i.e., "We live in interdependency hell!" (NTmgtX). The KJ methodology requires converting such affective language into more fact-based statements (I kept additional notes on the affective dimensions.)

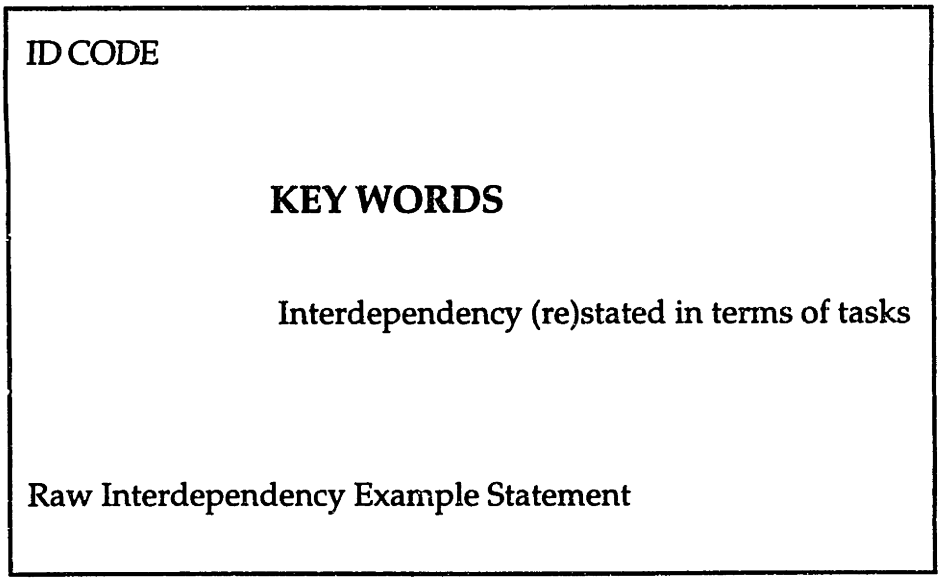
In order to facilitate the analysis and mapping to tasks, I created a core list of product development tasks at each firm based on the interviews and process documentation I had access to. For example, in the case of Microsoft I began with the task list created by Cusumano and Selby (Cusumano & Selby, 1995). I then used notes and handouts from the 'Microsoft Product Development Cycle Course' and interviews I conducted with team members and managers to expand and add a level of detail and definition. At Lucent, I relied on the firm's on-line process methodology descriptions as well as key informants. I also compared the task lists to some general product development task descriptions (Hayes & Clark, 1985; Ulrich & Eppinger, 1995) in order to ensure that the tasks were somewhat generalizable. These core task lists served several additional functions in the analysis. In particular, they were a reference point for understanding what a specific task was and often clarified the nature of the interdependency.

---

[56] In retrospect, I could have specified in the interviews that interdependency be described in terms of tasks. At the time, however, I didn't want to bias the examples by insisting that people state them as task relationships. To be honest, I was also unconvinced at that point that tasks were the right way to think about interdependency. Not providing more guidance in the interviews was probably a mistake and certainly complicated the data analysis. On the other hand, the language of tasks may be somewhat unnatural for managers and technical workers to use and therefore could have introduced other forms of biases.

Next I created a data template for each project in the form of a spreadsheet. Each row corresponded to an example of interdependency on the project, and each column specified specific information about that example (e.g., ID code, raw data in the form of the subject's description of the example, interdependency restated at the task level if necessary, key words, etc.) The above condensation process reduced the data from several hundred pages of transcripts to approximately twenty-five pages, concise enough to allow an overall view of the interdependency data yet documented and tracked minutely enough to trace general observations back to the transcripts for substantiation and refutation.

Although the spreadsheet formats greatly simplified and reduced the amount of data, they were cumbersome and awkward to deal with particularly in terms of identifying patterns across projects. To overcome this barrier I wrote a computer program which printed the data about each interdependency example out on a 3"x5" card in a standard format. The format is shown below:



As I began grouping and sorting the carded examples looking for clusters, one pattern that was immediately evident was the existence of duplicate examples both across and within projects at a company. In other words, people on the same project often cited the same example as did people on different projects (the latter occurred because the projects were in concurrent development and often shared resources and technology). Although duplicate data points are typically a problem in analysis, here they actually serve to strengthen the results. The fact that people cited the same examples is reassuring in that it suggests that the examples are not idiosyncratic but actually do capture a significant underlying concept.

Examples mentioned by more than one person were reconciled and consolidated. Examples that applied to more than one case project were counted more than once. Thus, there were no duplicates within a project but some duplicates across projects. Steps 1 and 2, therefore, constituted a form of data "scrubbing" in which I sought to make the qualitative language data uniform and suitable for analysis. Steps 3-5 described next structured the examples into hierarchical clusters and assigned labels to those groupings.

Step 3: Form 1st-Level Groupings and Assign Titles

The carded examples were grouped into clusters following the steps outlined in the Language Processing Method Manual (Juguilon, 1996), a close modification of the original KJ method (Kawakita, 1991). I began with several rounds of unconstrained pick-up in which I marked the examples most likely for consideration; unmarked statements were removed from initial analysis. By repeatedly inspecting the list of statements at each round and marking them, I was gradually able to eliminate examples that were outliers. This was followed by a second stage in which I formed first level groupings of the cards based on the similarity of the examples.

As noted by Glaser and Strauss and others, a major hurdle in analytical induction and clustering concerns the criteria for grouping (Aldenderfer & Blashfield, 1984; Bailey, 1994; Glaser & Strauss, 1967; Kawakita, 1991). How close does a "match" have to be in order to be considered a match with others in the group? Whereas quantitative analyses typically apply statistical tests such as mean-squared distance for comparison decisions, such "objective" criteria are not possible with language based examples. As suggested by prior researchers using this method (Griffin & Hauser, Winter 1993; Juguilon, 1996; McCord & Eppinger, August 1993), I formed the groupings by 'image' as well as the similarity between words and the 'distance' between examples. In effect, I created a mental picture of each card and then tried to find the common thread among the images. Images between cards that were "strong" (close) were grouped together.

As recommended by the originators of this method, I explicitly tried to avoid grouping based on prior ideas or logical relations (i.e., cause and effect or problem solving relationships). Thus, no attempt was made to specify the logic by which clusters should occur a priori. For example, I could have adopted a technological perspective and grouped all of the examples according to the physical components of the product or explicitly looked for support of Thompson's (1967) framework. I resisted doing so as it would have run counter to the goal of the inductive process, which was to create a description of interdependencies which was consistent with the way new product development participants think about their work. Kawakita does suggest limiting the number of primary groupings to between three and seven as a way of forcing focus on the "vital few," a recommendation I adhered to.

I next wrote a title for each first-level grouping that conveyed the common thread or meaning in the cluster of cards.[57] As described by Kawakita, title making is a process of going "up the ladder of abstraction" so that the title captures the common element in the underlying examples but at one higher level of analysis. For instance, a cluster of examples having to do with relationships among functions and features in the product was initially labeled "component-to-component" and later integrated with other clusters under the general heading of "component interdependencies."

Step 4: Form 2nd (3rd) level groupings and assign titles

Each of the primary groupings was further characterized as a set of secondary groupings; in the case of very complex clusters (types), the secondary groupings were sorted into tertiary sets as well. I applied the same approach for title making as in first level clusters.

Step 5: Lay out the groups and show the inter-relationship among them

Finally, the grouped structures were laid out on a bulletin board to explore the internal structure of the first level categories as well as the relationships among them. At this stage, the analytical process changed from one based on intuition and image to one based on logic and possible cause and effect relationships. The resulting arrangement is what is referred to as a KJ Diagram.

Considerable care was taken to ensure that the categories (types) of interdependency identified were defensible. For example, at several points I moved back and forth between the data display (carded examples) and the original transcripts to see whether or not full quotations supported or documented the emerging taxonomy. To get an indication of intra rator reliability, I periodically put the sort away and later

[57] This corresponds to calculating the average or standard deviation of a set of numeric data.

repeated the clustering exercise without referring to the earlier results (Nutt, 1984). Differences were reconciled, and I repeated this sorting/stepping away process until the classifications were relatively stable (McKelvey, September 1978). Inter rator reliability was tested by having an assistant familiar with new product development in general but not with the specifics of these cases independently review and cluster the examples. Disagreements (in clusters and labels) were discussed and resulted in further refinement of the categorization. Finally, I fed the groupings back to my primary contacts at each firm at regular intervals during the analysis process in order to incorporate their reactions and feedback as well.

FIGURE C.1
CASE STUDY THEMES

• People find the level of interdependencies to be overwhelming and perceive the situation as getting worse over time.
The subjects were nearly unanimous in their perception that the level of interdependencies was growing over time and starting to strain individual and team level capacities.

• Most product teams participate in multiple inter project relationships and many play different roles in each.
One implication of component and code level interdependencies is that each product team is involved in a number of dependency relationships and often plays different roles in each. For example, a given team may constitute a component supplier in one scenario and a component user in another. Or a team may be responsible for providing its technology to two distinct products at the same time. These multiple relationships can create various forms of conflict.

• People tend to focus their attention on different dependencies.
Another implication of multiple interdependencies is that different people focus their attention on different interdependencies (or on different aspects of a given dependency). One determinant of this interpersonal variation is the very fact that people are being tugged in different directions. Additional factors are functional responsibility, task facilitation or blocking, time pressure, personal satisfaction and challenge and the organizational reward system.

• Interdependencies are hard to make visible. People therefore need to continually devote time and attention to 'discovering' dependencies.
The subjects talked about why interdependencies are sometimes hard to make visible and the steps they took to continually search for "hidden" dependencies.

• Interdependencies often represent variables over time.
Once people are able to identify a given dependency, "there's a feeling that as long as I understand what the dependency is, and it won't change, it's OK." But dependencies often resemble variables over time- the information or functionality one depends on, who provides it and who potentially constitutes the blocking factor as well as the overall context of dependency can all change over the course of a project. Again, this has implications for how people approach and mange dependencies.

- Interdependencies extend across time.

The temporal nature of interdependencies- the fact that they extend over time- can lead to conflicts, lags, interference and blocking.

- There exists a tension between time to coordinate and time to produce.

At every level and discipline of the organization, people are struggling to find the time to both coordinate with others and accomplish their own work. The existence of multiple, conflicting interdependencies can create a double bind. The more interdependencies exist, the more they have to be managed in person (usually via face to face contact). This is due to the fact that the issues are more complicated and people are reluctant to 'speak for' third parties.

- As the level of interdependency rises, team members experience a loss of autonomy and independence.

Everyone (members of teams providing component technology as well as teams relying on those providers) described experiencing feelings associated with a loss of autonomy and control as a result of dependency. This has implications for team morale, motivation and inter-team relationships.

- Interdependencies are socially, culturally and historically embedded.

Each product team has a method- a way of developing software- as well as different historical and cultural practices which they a have developed over time. As teams start to share technology, however, they find themselves depending on a group of teams all of whom do things completely differently. The build process is an excellent example of the types of technical and organizational conflicts that can result.

APPENDIX D
RESEARCH METHODOLOGY AND DATA ANALYSIS

The case study represents but one of several ways of conducting social science research (Cook & Campbell, 1979; Judd, Smith, & Kidder, 1991). Other approaches include experiments, surveys, histories, and archival analysis. Traditionally, research approaches were portrayed as being arrayed hierarchically, with case studies appropriate for exploratory phases of an investigation, surveys and histories suited for description, and experiments deemed necessary for explanatory or causal inquiries. Such a categorization belies the reality of actual research. Experiments with an exploratory motivation have always existed (Milgram, 1965), while some of the best case studies have been descriptive (Whyte, 1955), and explanatory (Allison, September 1969). As a result, most researchers today agree that, although there do exist significant differences among the research strategies, purpose is not a distinguishing factor (Yin, 1984). The present case study was primarily designed to achieve description and explanation about the interdependency concept and its implications for task coordination.

The firms were guaranteed anonymity and confidentiality at the project and individual levels. I also agreed to let representatives at each company review the results prior to publication to ensure that nothing proprietary or incorrect was revealed, although I retained final editorial control. See Figure D.1 for the actual research agreement.

Data Sources and Collection

The study draws upon a combination of qualitative and quantitative data from five of the six major sources of evidence for case studies (Yin, 1984). The data collection

was guided by a case study protocol, which outlined the instrumentation as well as the general rules and procedures used in the field. The use of such a protocol has been advocated as a major tactic in increasing the reliability of case studies (Yin, 1984).

Field Research and Interviews

Field research consisted of 33 interviews over a two and one-half month period on-site at Microsoft. This was complemented by comparable time (two months plus four additional week-long visits) and 38 interviews at Lucent Technologies. At each company, one or two key informants helped me to select projects and an initial set of people to interview. I subsequently followed a snowball sampling technique in which I asked each interview subject to recommend additional people I might talk with (Cook & Campbell, 1979). Of the 82 people contacted, no one declined to be interviewed, and I was able (given time constraints and schedules) to speak with 71 people (87% of those contacted). The interviews were roughly distributed across projects, functional areas, and levels of experience with the exception of the Autophone project at Lucent, which was under-represented.

Initial interviews were open-ended (I asked the subject for facts, opinions, and insights regarding interdependency on their project as well as basic background about their project and job). For example, I began by asking subjects about their prior education and job experience and then asked about the current project and their roles on it. Subsequently, I asked if interdependency was an issue in their work, and how they thought about it. I also asked what parts of product development and team management were complicated by interdependency, and how (if) interdependency changed over time. The interviews became more focused and semi-structured over the course of the investigation as I gained a feel for the projects, settings and issues (still conversational in tone, but I followed a set of questions defined in the protocol).

All interviews were recorded on audio tape.[58] Shortly after each interview, I transcribed the tape and noted down emerging key issues. I reviewed these notes before subsequent interviews to remind myself of themes that resonated across people and projects. In line with Glaser and Strauss' suggestion, interviewing ended when the results began to converge (Glaser & Strauss, 1967). Ultimately, approximately 100 hours of conversation were recorded and transcribed.

At the end of the field investigation phase I thanked the subjects for their participation. I also prepared individual project summaries, which my primary contacts at each firm reviewed. Their comments, corrections, and suggestions were subsequently folded into the analysis.

Secondary Sources and Other Data

At each company I was given complete freedom to roam around and directly observe meetings, informal conversations, and the overall ambiance. I was also granted access to various forms of documentation and archival records (e.g., copies of electronic mail messages, project schedules, design documents, memos, company announcements, written project reports, organizational charts, and reports off of project tracking databases). Of particular usefulness in terms of analyzing each project's coordination mechanisms was documentation on their development process (an on-line manual at Lucent and a four day course taught by current project team members which I attended at Microsoft). Finally, I maintained a large notebook on each company over the two and one-half year course of this study in which I collected other formal studies on the two sites and relevant articles from the mass media on each company, industry, and

[58] I asked each subject's permission before doing so. Only one person declined to be recorded, although the tape was occasionally turned off during politically sensitive or proprietary parts of the conversation.

technology. This data enabled me to understand the larger technical, market, and institutional context each firm operated in as well as control for the possibility of historical effects.[59]

Data on Project Performance

Since measures of objective performance that are comparable across different companies and technologies do not exist, I adopted a multi-method approach to measuring project performance. In addition to a questionnaire administered to team members, managers, and independent observers, I constructed various forms of matrices based on data from interviews, project documentation and tracking databases I had access to. This approach is similar to that taken in numerous studies of new product development (Ancona, 1990; Ancona & Caldwell, 1992; Iansiti & Clark, 1994).

Before writing the survey, I met with the staff responsible for maintaining and analyzing data on each of the projects. Together, we discussed the biases and limitations of the information in various reporting systems and identified appropriate candidate measures. I then obtained the subjective performance assessments along approximately the same category dimensions by surveying upper level or division managers, functional managers, and team members approximately one year after the initial round of data gathering. Respondents were also encouraged to provide explanations of why they gave particular ratings. Finally, I had an expert panel at each firm exhaustively evaluate the projects, similar to Ancona (Ancona, 1990).[60] Table D.1 displays the survey design. See Table D.2 for the performance categories and control variables. Appendix F contains a copy of the actual survey.

---

[59] This proved foresightful given that AT&T voluntarily decided to split itself into three companies midway through this study.

[60] Note that whereas project managers and team members each evaluated one project (their own), experts evaluated all three projects at their respective company.

To ensure the accuracy of the assessment data, evaluations were made independently and submitted directly to me. I also tailored the questionnaires slightly by using language appropriate to each company and project group. The second round of lagged data collection enabled me to test for the possibility of biases associated with real time performance assessment.

Data Analysis

Data analysis, which consists of examining, categorizing, tabulating and recombining evidence to address the initial questions or propositions of a study, has been cited as one of the least developed and most difficult aspects of doing case research (Yin, 1984). In addition, although much theory and experience and many tools exist for collecting and analyzing numeric data, different approaches are needed for more qualitative information.

I used an iterative, multi-method analysis approach consisting of four basic steps: (1) a content analysis of interviews and product, project and process documentation to identify alternative approaches to interdependency management as well as team strategies, (2) the development of project profiles assessing the internal and external contexts within which the teams operated, (3) a multi-dimensional project performance ranking, and (4) an assessment of the proposed relationships among the key constructs.

A crucial aspect of this analysis was the ability to display the data so I could begin forming and examining hypotheses. I used several types of data display techniques (Miles & Huberman, 1984) to facilitate this process, as described below.

Identifying Interdependency Approaches and Deriving Team Strategies

Once the interview data were collected, I reviewed the transcripts and listed the techniques individuals and teams used to manage interdependency. I adopted a very broad initial definition of technique. For example, I included both structural solutions such as drawing particular team or sub-team boundaries as well as more process oriented approaches such as special forms of meetings. Some of the techniques represented individual coping mechanisms, an example being the use of status lists or heuristics for dealing with the overload associated with multiple interdependencies. Others were at the team level and took the form of particular roles or processes. I also recorded techniques at the level of product architecture or strategy because these too were often ways for dealing with interdependency. I supplemented this list with additional information from documentation I obtained on each company's official development process.[61] I coded each technique in terms of the type of solution it seemed to represent. For example, "team design for sharing components" and "prioritized partners" were eventually grouped together and coded "minimized external interdependencies."

I then developed short summaries of the key techniques used on each team representing their different strategic approaches to interdependency management. In developing these, I was most interested in how the teams dealt with the challenges associated with facing multiple interdependencies simultaneously. For example, evidence suggesting that the team made conscious tradeoffs between and among different interdependencies or deliberately devoted resources to certain interdependencies in a discretionary manner was noted as "prioritized interdependencies." For another team whose members unanimously decried their lack

[61] When deviations existed between official and actual software development processes, I relied on the actual work process descriptions obtained in the interviews.

of time and frustration due to the number of contingent task relationships, I wrote "team members trying to deal with all interdependencies at once with little sense of discrimination or hierarchy."

Developing Project Profiles

Interdependency management occurs within a specific context, and a team's actions, choices, and outcomes must be understood within that context. To focus solely on solutions and strategies without attending to context amounts to "context stripping," with attendant risks of misunderstanding the meaning of events. In order to avoid such a bias, I collected information about each team along a number of contextual dimensions and then evaluated their approaches to interdependency management within that larger setting.

For example, Tables D.3 and D.4 are meta-matrices or master charts assembling descriptive context data from each of the projects in a standard format (Miles & Huberman, 1984). Such descriptive context charts display the key factors that make up the context of team behavior and performance, including technical and strategic factors (such as the product complexity and competitive market environment), people and cultural factors (such as leadership, power, conflict and the level of experience, knowledge and skill on the team), internal organizational factors (such as the physical environment and resource availability), and historical factors in terms of past product and project events.

A major limitation of the design of this study was its cross-sectional nature; I was observing and trying to understand interdependency management at a single point in time. During the course of the analysis, it became readily apparent that one also needs to understand the course of events, rather than simple snap shots, in order to grasp,

understand, and explain why certain teams adopted particular solutions. In order to overcome the deficiencies associated with the cross-sectional design, I developed time ordered matrices (Table 4.3 in chapter 4) (Miles & Huberman, 1984). This enabled me to compare changes in product, project, and process chronologically.

Outcome Assessment

Organizational performance in general is very difficult to specify, measure, and interpret. This is particularly true in the case of new product development projects due to (1) the large number of extraneous internal and external factors that can potentially affect performance and (2) the inherent multi-dimensionality of the performance concept (Cusumano & Nobeoka, 1992).

For example, performance has been shown to be a function of, among other things, the newness of the technology (both in absolute terms and to the team or company), the complexity of the technology, the volatility of the market (as manifested in the amount of change in the requirements), the size of the product and team, the level of experience of the team members, and group tenure. Studies have also focused on different aspects of performance (i.e., input, output, financial) and explored the inherent tradeoff among different dimensions: flexibility versus efficiency, speed versus quality, etc. (Ancona, 1990; Clark, Chew, & Fujimoto, 1987; Clark & Fujimoto, 1989; Cusumano & Kemerer, November 1990; Cusumano & Nobeoka, 1992; Eisenhardt, 1989; Van de Ven & Ferry, 1980). As a result of both of the above considerations, it can be difficult to get a valid and reliable measure of project performance. Most studies resort to multiple measures in order to increase the reliability of the comparison (Ancona, 1990; Cusumano & Kemerer, November 1990; Cusumano & Nobeoka, 1992).[62]

---

[62] This multiplicity refers to both the dimensions of performance (schedule, budget and quality, for example), as well as the nature of the measures (subjective and objective formats) (Cook & Campbell, 1979).

In addition to subjective performance ratings from the questionnaire, I constructed two forms of matrices based on the objective data I had access to. The first was an unordered effects matrix in which I sought evidence of positive and negative performance outcomes along the dimensions of schedule, product component integration, product quality, and team functioning (Table 4.2 in chapter 4) (Miles & Huberman, 1984). Explanations for particular outcome effects were also noted on the display.

The second form of outcome display was a case-ordered summed display of team member reported problems (Table 4.4 in chapter 4) (Miles & Huberman, 1984). This form of display orders projects according to a main variable of interest, in this case the type and severity of certain problems, so that one can begin to see the pattern of differences and identify high, medium, and low projects. In order to form this ordering I first extracted all problem reports from the interviews and classified them under different problem headings. I then put them in a rough conceptual order from minimal, trivial, and short term in nature to substantial, long term or unresolved.

For example, a problem such as fixing a bug was coded as a minor inconvenience affecting one or a small number of people (if mentioned infrequently). Other problems which threatened shipment, caused a delay in the schedule of more than one month, or were cited by a large percentage of the team members interviewed were coded as major ongoing problems. Examples are product architecture uncertainty, inadequate staffing, or frequent mention of a major bug. To order the projects I began with a rough estimated ordering, entered the data, then re-arranged the rows (and sometimes the

columns) until a systematic order appeared with highly effective management at the top (minimal problems cited) to least effective at the bottom (many reported problems).[63]

Assessing Relationships

The above three analysis steps, therefore, yielded data on team strategies, interdependency management approaches and project performance. Finally, I analyzed how existing patterns of interdependency and management strategies and approaches were related to evaluations of performance in order to propose relationships among these sets of variables.

The above sections present a fairly sanitized description of the data collection and analysis processes-- suggesting a linear progression from study design through data collection and on to conclusions. In reality, the process was much more iterative and highly idiosyncratic. I developed a conceptual grasp of the questions being investigated as the study progressed, and the data was integrally involved in the conceptualization process itself (Bailyn, 1977; Van Maanen, 1997). The analytical process was highly non mechanistic and involved intuition, creativity, serendipity and speculation combined with analytical discipline (Weick, 1989).

[63] Note one potential problem associated with this method. It attributes greater problem awareness to inferior management when, in fact, better performing teams may be more cognizant of their problems than poor performers. (Thanks to Kathy Sutcliffe at University of Michigan for suggesting this.) Given that the high problem projects were associated with poor performance in the other data sources, I do not think this effect is working in my data.

FIGURE D.1
SAMPLE RESEARCH AGREEMENT

Nancy Staudenmayer is a doctoral candidate and research associate in the Management of Technological Innovation program at the MIT Sloan School of Management. She worked at AT&T Bell Laboratories for six years prior to attending MIT and attained a Masters Degree in Statistics from the University of California, Berkeley. Staudenmayer's dissertation focuses on the processes companies use to coordinate and control interdependencies in large scale software development projects. AT&T and Microsoft are the sources of empirical data for the investigation. This document outlines the nature of the research relationship between [company name] and Nancy Staudenmayer.

Reciprocity

Staudenmayer will be granted access to [company name] to study the thesis topic as she defines it and will maintain final editorial control. However, all publications will be forwarded to [company name] prior to publication to correct errors of fact and to ensure that no commercially sensitive information is revealed. Staudenmayer agrees to give a presentation of the results, if desired, and will meet individually with interested parties at any time during the project.

Anonymity

The dissertation will maintain complete anonymity at the project and individual levels of analysis.

a. Project

The specific name and proprietary technical characteristics of the projects and features sampled in the study will be disguised (e.g., "the XYZ switch", "the ABC operating system"). Some technical details (e.g., size, high level functionality) will be revealed but may be re-scaled if commercially sensitive.

b. Individual

All individuals participating in the study will remain completely anonymous. They will be identified in data documents with an ID number known only to the principle researcher (N. Staudenmayer).

Confidentiality

All data gathered during the course of the study will remain completely confidential. Access to the information, without the prior consent of [company name], will be restricted to N. Staudenmayer and her MIT thesis committee. Off site, all data will be stored in a locked office on the MIT campus.

Informed Consent

This agreement establishes informed consent at the corporate level. In addition, all individuals within [company name] maintain the right to not participate or withdraw from the study at any time. A subject may also request that certain interview or observational data not be recorded.

TABLE D.1
SURVEY DESIGN

		LUCENT			MICROSOFT	
	HANDPHONE	TOLLPHONE	AUTOPHONE	DESK	DATA	NETWORK
TEAM LEAD RESPONDENTS						
Dev Mger						
Test Mger						
Project Mger						
Document Mger	n.a.	n.a.	n.a.			n.a.
TEAM RESPONDENTS						
UPPER LEVEL MGT. RESPONDENTS						
EXPERT RESPONDENTS						

TABLE D.2
SURVEY CATEGORIES AND VARIABLES

CONTROL VARIABLES	DEFINITION	DATA SOURCE
Individual		
Prior System Project Experience	number of major system projects respondent has worked on	survey item #1
Product		
Product Size and Complexity	number of non-commentary source lines of code, broken down into executable vs. non-executable categories	two key project informants
Requirements Volatility	degree of change in requirements after initial definition	two key project informants and survey item #2
Project		
Cross-Project Sharing and Dependency	level of externally delivered components	two key project informants and survey item #3
Team Size	number of full-time contributors by function	two key project informants

OUTCOME VARIABLES	DEFINITION	DATA SOURCE
Interdependency Mgt Effectiveness		
Rework	amount of actual and elapsed time spent reworking due to adjustments in requirements or changes in code	survey item #4
Wait Time	amount of blocking or waiting time (broken down by blocking factors)	survey item #5
Ratings of Effectiveness of 5 Types	effectiveness of task coordination in 5 interdependency categories	survey item #6
Ratings of Working Relationships Within & Across Subsystems	effectiveness of working relationships within and across product subsystems	survey item #7
Outcome Satisfaction Level	level of satisfaction with eventual product outcome	survey item #8
Product & Project Performance		
Adherence to Schedule	meet or exceed initial schedule estimate	survey item #9
Adherence to Budget	meet or exceed initial budget estimate	survey item #10
Adherence to Functional Requirements	meet or exceed functional requirements	survey item #11
Defect Rate (Year 1)	number of system-level crash bugs during first year of product release	survey item #12

TABLE D.3
MICROSOFT PROJECT CONTEXT

CONTEXT CATEGORY	DIMENSION	DESK	DATA	NETWORK
Technical & Strategic Factors	**product description**	an integrated application suite for PC products; available in Standard and Pro versions	an entry level database management program for PC products; sold as both a product and an application in Desk suite	a network operating system for PC products
	product dimensions	~ 2 M LOC; 1440 files; 600 page manual; 360 release formats ~ 200 THC	~ 1 M LOC ~ 75 THC	~ 5.6 M source LOC in product; 3+M LOC for tests; 6000-8000 files; 18 hours to do a clean build ~ 250 THC
	customers	mass market; traditionally individuals, but increasingly corporations	mass market; sophisticated individual "power users"	mass market; individuals and corporations needing highly reliable OS
	competitors	Lotus, Word Perfect		UNIX, IBM AS400, VAX
	schedule	~ 12 month cycle	~ 12 month cycle (goal)	~ 9-12 months
	key technical and strategic challenges	Integrating user interface, features and schedules of formerly independent application products Customer delivery model changing to annual 12/24 month cycle Integrating code, schedule and processes of formerly independent application teams (See Longitudinal Display)	Four key functional components provided by external teams: added huge risks" A component user and a component provider (to Desk) Cross product component design and schedule conflicts: "a set of conflicting forces pulling you in different directions" Motivating component providers: "little emotional satisfaction"; "a heinous job"	Achieving rapid development cycles and high reliability given large size and complexity of product Balancing flexibility and adaptiveness with rigor and control Feature sharing
People & Cultural Factors	**demographics**	young (avg. 30 years); primarily male	young (avg. 30 years); men and women in key management roles	young-middle age (avg. 35-40 years); primarily male
	team culture and atmosphere	Value ability to produce: "good people build things" High respect: "apolitical"; "brother against brother"		Highly disciplined: "we are hard core" Non-bureaucratic

	level of experience, knowledge and skill	very high company experience (avg. 6.3 years); very high domain experience (experience on successive versions); very little other industry experience	medium company experience (avg. 3 years); high domain experience (in tools and databases); some other industry experience, particularly on component teams	very high company experience (avg. 5.5 years); very high domain experience (on earlier versions and large system development in other industries)
	leadership and power	Developers: "everyone wants to be located near developers" X Team: "they are king"		Developers and Testers: "developer and test peers pretty much decide what will happen" Heads of Development, Testing and Program Mgt: "they set the direction of team at daily 9 am meeting"
	conflict	"tribal" relationship between apps; some resentment in apps about "Desk tax" and "loss of autonomy"; documenters perceive problematic relationship with developers, but not vice versa	among products sharing same components; component team members uncertain of their status relative to product teams and feel under-appreciated	with other OS team regarding marketing messages, design and delivery of shared components and scheduling.
Internal Organ'l Factors	**physical setting**	occupy multiple floors in a common building; private offices; relatively plush decoration	occupy multiple buildings on a common site; private offices; moving to a new building	multiple floors of one building; single offices; modern
	resources	no major resource problems, except time (especially for leads)	testers "way under-staffed"; also time, especially for program managers	petitioned management for heavy investment in fast machines for component integration (granted)
Historical Factors	**product history**	products formerly shipped as independent applications with separate schedules and code	earlier versions of product were built on same base as V1.0 and had only 2 external component dependencies	most product pieces have been developed internally, but the number of external deliverables is increasingly significantly over time
	team history	formerly organized as independent project teams; Desk team formed from members of app teams		most leads and key team members have worked together before

TABLE D.4
LUCENT PROJECT CONTEXT

CONTEXT CATEGORY	DIMENSION	HANDPHONE	TOLLPHONE	AUTOPHONE
Technical & Strategic Factors	**product description**	a wireless communication protocol for use in portable handsets in Japan; reusing ISDN standard; hardware supplied by Hitachi;	an application offering operator, credit card, third person and directory assistance services	an infrastructure for mobile phone customers; hardware supplied by Qualcomm; using newer and less proven technology
	product dimensions	~ 50,000 NCSL from wireless subsystem and 12 other subsystems on base of 10M ~ 45 THC (plus ~ 60 load builders, mgt, system eng, customer support)	~ 1-2 M NCSL from operator administration subsystem and 1-50 other subsystems depending on ~ 100 THC + support	~ 1-2 M LOC ~ 120 THC in IL, 50 THC in NJ + support
	customers	custom development for a Japanese telecom	captive supplier to AT&T (90% of features); some custom development for LECs and foreign telecoms	custom development for cellular service providers (Cellular One, Ameritech)
	competitors	competitive product already in market	Northern Telecom, Alcatel, Siemens	Qualcomm, Ericsson, Motorola, Northern Telecom; standards war
	schedule	~ 10 month cycle (proj committed 6/94, reqs complete 4/95, HLD 8/95, full team 4/95, code complete 11/95)	small (< 3 THC) U.S. features average 10 mos; large (4-50 THC) int'l features avg 20 mos	
	key technical and strategic challenges	Achieving very aggressive schedule set by sales force before reqs gathered Achieving functional goals to beat existing technology on market	Customer delivery model changing from biannual release to per customer request Transitioning from system to feature products (See Longitudinal Display)	First time development effort in new, unproven technology; extremely competitive market with uncertain requirements An unproven technology
People & Cultural Factors	**demographics**	middle age (avg. 35 years); very heterogeneous in terms of gender, race and ethnicity	older (35-50 years)	middle age (avg. 35 years);
	team culture and atmosphere	Engineers: Urgency: Cooperative:		Little team unity or personal accountability "Frenzied and chaotic"

	level of experience, knowledge and skill	very high company experience (avg. 12.6 years); high domain experience (on other wireless products)	very high company experience (avg 15 years); very high domain experience (same people and location for 15 years)	medium company experience (avg 16.2 years); little prior wireless experience
	leadership and power	Feature Engineer and "Wild Card"		
	conflict	very little; some cross team conflict when another wireless project submitted late change	very little	significant cultural clashes between IL "processors" and NJ "hackers"
Internal Org'l Factors	**physical setting**	core team on 1 floor in 1 bldg; major components developed in separate bldg on same site; 1 component developed in UK; hardware developed in Japan; 2-4 person offices; utilitarian decoration	core team on 1 floor in 1 bldg; co-located with RC/Data; shared offices	cross-site development between NJ and IL
	resources	significantly understaffed initially and key roles missing during key periods (i.e., data eng during HLD)		
Historical Factors	**product history**	two earlier wireless efforts were "disasters"	OA subsystem design started in 1982; under continuous development since 1984; U.S. and Int'l code split in 1988	None
	team history	"most people have worked together on other efforts"	U.S. and Int'l teams were split but recently merged; knowledge drain during downsizings in 1992 and 1994; change from cross-functional to functional org	None

APPENDIX E
APPROACHES TO PRODUCT COMPONENT INTEGRATION

Product component integration (known as the "build" function at Microsoft and the "load" function at Lucent) refers to the process by which the software component pieces get integrated or "knit together" to form one working product. It is a crucial process, which many tasks depend on and therefore central to managing many types of interdependencies.[64] The following quotations from Network and Autophone team members capture the centrality of the task in terms of team functioning and activities:

> The build process by its very nature is a bottleneck. Everything developed has to pass through the build team. Testers can't test without it, developers, program managers, and product managers depend on it. These groups need clear and consistent expectations about the build so they can plan their time and work accordingly.
>
> Regular builds act like a heartbeat on the team.

The very centrality of this process suggests that it might be a key lever for managing multiple interdependencies. Table E.1 presents a comparison of the organizational design solutions the six teams imposed on product integration. Analysis suggested that the approaches differed in four respects: task allocation (who performed the integration, and how they were related, both organizationally and geographically, to the team), resource investment (whether or not the team devoted extra resources in the form of capital for computers and highly skilled technical people to the process), incentive structures (if or how they incented people to frequently test and integrate their components), and timing (when and how often integration was performed, and how dynamic the process was).

---

[64] One indicator of its centrality is the passion that the build process seems to inspire in people. Virtually everyone I interviewed had something to say about the role of product component integration, how it was being performed on their project, and its impact on their work.

Teams adopting a systemic manipulation strategy tended to manage product component integration as a self-regulating team process. It was performed and managed by a dedicated, highly technically experienced group internal to the team, and they drew upon a variety of both technical and social incentive mechanisms to ensure that people cooperated. Integration was performed very frequently and regularly, usually once a day, but also quite dynamically depending on the state of the project. In effect, the build served as both a central work coordinating process (by enabling team members to easily respond and adapt to the latest changes made by others) and a pacing mechanism (by serving as a visible signal to both managers and team members of the current state and progress on the project).

Product Component Integration on the Network and Handphone Projects

Of all six case study projects, the Network team appeared to have devoted the most careful thought and analysis to how product component integration should be performed and the advantages or disadvantages associated with different approaches. This attention to detail grew out of some painful experiences in prior releases.

The Network build was performed and managed by a small, dedicated group internal to the Network team; 3-4 people were directly involved with integration and about 9 others tested the builds and performed some miscellaneous activities. The build manager has twenty years of experience in the software industry. He believed very strongly that the build team should play a "policeman or mother hen role" on a project by controlling both the number of code changes and their quality through a variety of incentive mechanisms:

> The quality of the build is inversely related to the number of check ins. Sometimes we 'open the flood gates' and let anyone check anything in. Other times we use a more controlled, phased in approach where we will only allow changes in certain pieces but keep everything else stable...

> Eventually when the system gets full and buggy we go to a more restricted check in mode where people can only make changes that fix high priority bugs.

This manager also talked about the need to "train" developers and testers such that they understood and appreciated the impact their work had on others and delivered high quality, well tested code:

> Product component integration is very much a social phenomenon, not just technical. We need people to understand the impact they are having on the rest of the team and the value of cooperating. [We use several mechanisms to incent developers to check their code in regularly.] First we rely on support from the development team leads. They promote quality in the build process and emphasize the professional responsibility of submitting clean check ins. Shame is also an incentive. Before, when the process was more random, if it broke no one knew who was responsible. Now it is easy to pinpoint, and that changed developers behavior. Humor helps too. We make people wear goat horns or pay money. We actually bought a stereo with the money we collected.

Handphone also organized the build (load) process internally. Although the manager was less experienced than on Network, she worked very closely and interactively with the feature engineer on the project. Like Network, Handphone team members only came to see the importance and value of their approach after some early bad experiences. They quickly found, however, that the use of incentive mechanisms attuned to the particular personality profiles on the team as well as an internal structure changed people's behavior and increased cooperation levels:

> The first couple of times people were a little sloppy. They didn't have the discipline to always compile and test their check in. So we sent a notice via email-- 'you made a mistake. It cost the group because every time we have to rebuild it costs a lot of time.' It embarrassed people but then they got serious and made sure they didn't break it.
>
> Doing the integration internally can lead towards people wanting to do better code because it's not Joe Schmo they're hurting-- it's their team mate

> who now won't be able to make the phone call work. It brings it very close to home if its broken... With an outsourced public load, there is a perception that 'I never want to break that.' And when you do break it, your name is very well known because those builders will remember. But it's funny, because even when you break it, what are you breaking? You may be breaking something that is completely divorced from you. So you don't have the same ownership for what you broke.

Note that Network tended to use a public humiliation incentive scheme--potentially the whole team (more than two hundred people) were informed when someone caused the build to fail. The Handphone feature engineer, in contrast, contacted individuals privately to point out the impact they were having. These different incentive structures were appropriate, given the demographic and personality differences in the two companies. Lucent tends to be a population of older workers who operate in a more respectful and considerate work environment. However, the data suggest that both projects also targeted incentives to the individual. For example, the Network build manager noted that he sometimes communicated privately with developers who were more sensitive to criticism, while the Handphone manager would sometimes resort to a public announcement if the developer repeatedly failed to cooperate.

Performing the build internally also enabled the teams to integrate the components selectively and dynamically, depending on the particular needs of the project. This flexibility meant that they could take more risks while simultaneously remaining "good citizens" in the sense of not breaking functionality in other projects:

> When product integration is managed by people internally to your team, it means you can build selectively-- only the pieces or times you want-- in a more secure environment. If you have problems, you don't have to worry about hurting other projects. It's more flexible, probably the biggest benefit, so you can take more liberties, beneficial liberties, early on.

Finally, this approach to integration resulted in certain benefits in terms of testing, compiling, and task coordination among the developers. In particular, it eliminated so-called "chain errors" and compile duplication, thus saving human and computer resources:

> One advantage it gives you is that you have one central place to build all the software. For example, my software may depend on someone else's software on the terminating team. Very close. But just to get their code together with my code to build these two together can be a big nightmare because they may have code that is dependent on yet another person's code... And that stream can go on and on... You can do it individually, and have to deal with all the dependencies with everyone else, or you can put it into one spot and build it all at once.
>
> It saved a lot of people and computer resources for everyone. Say I need to build my software. In order to compile my software I have to go get somebody else's piece and bring them in to meet. That's one interaction. But that won't necessarily work in reverse. What I built may not be sufficient for the other person because they have other stuff that they didn't hand me... So now you have two people who each have a chunk of the same identical software shared between them but not symmetrically. And we're both using computer resources to compile. [When done collectively] we do one compile for everyone. One that is bigger. But that big one does not equal the sum of everyone's little ones. It's much, much less.
>
> Say I make one line of code change. Then I need to compile and test that change. Fifteen or twenty people may be making changes and compiling in parallel. The problem is that the system starts to slow down. Also, when I test my change I may not test it against those fifteen other changes... [The way we do it now] there's faster turnaround and increased performance time because fewer people are using CPU time. There's also potentially a shorter integration test interval because when I do a test, I'm not just testing my one line of code, I'm testing it with the fifteen others.
>
> Say you have two people modifying code at the same time. Each developer makes changes to the code, and they take those changes and build them... But each developer only sees his own changes. Each builds against the approved base and not against the other's changes. So each tests out OK but when you put them together you get breakage and it won't work.

Product Component Integration on the Desk and Tollphone Projects

The Desk product component integration process was undergoing a transition as the application teams, which previously performed their builds independently, sought to define a common process. Although the interviews suggest that people recognized the benefits associated with a well-defined process-- and most of the application teams had one-- they were encountering difficulties gaining consensus about how it should be performed given the new integrated Desk product and organization. As one team member observed "We've got the technical stuff [associated with the build] down. We don't have a handle on the organizational part yet."

For example, there was conflict over what time the build should be performed. One application team preferred to check in their day's work at 5:00pm whereas another liked to synch late in the evening (people on this team tended to come in mid-day and work later). The alerting mechanisms also differed across the application groups. When the build failed, some teams alerted only those people responsible for the problem. Other groups sent out a message to the entire team when the integration was completed successfully. The teams also used different rules for determining how to transfer responsibility for performing the integration. Although these differences may seem trivial, they represented "historical habits" in each of the teams, and people were quite reluctant to give them up.

Tollphone team members likewise displayed a high knowledge of and appreciation for the centrality of product integration. In this case, recent cost cutting efforts at Lucent forced the team to make certain changes in how integration was performed that they felt were far from ideal. For example, capital for computer resources was generally unavailable. As a result, the integration process was quite slow and introduced lags and delays in people's work:

> Computer expenses are quite tangible and therefore the focus of cost saving efforts. If I sit here for a day and a half waiting for a build, well, they pay us whether we sit here or not.

Tollphone integration was performed by two builders in a supporting organization. Again, a series of cost cutting measures had a negative impact on the process when contractors with little skill and experience were assigned to it:

> This past Spring the two loaders were moved out of the Tollphone organization and put in the load organization. They eventually took other jobs. The new people are ... novices who make a lot of errors and poor decisions... It's gone to strangers and the performance and commitment have dropped.

Other people on the project concurred that product integration was a "critical path job," and the project therefore benefited by having skilled people perform it.

Product Component Integration on the Data and Autophone Projects

It is somewhat difficult to analyze the approaches Data and Autophone took to product component integration because, quite simply, both teams lacked a well-defined process and were never able to achieve integration on a regular basis (as of the last data collection point). However, the various attempts they did make differed strikingly from that on the other cases.

Most notably, in neither case did anyone on the project have primary responsibility for the integration function. The responsibility was either shared (diffused) across the team (Data) or outsourced (Autophone). Nor did the team seem to recognize the centrality of the process by, for example, allocating money or experienced people to it. The two projects sought to perform integration infrequently relative to the

other cases (weekly or biweekly) on a fixed, pre-set schedule but rarely achieved this goal without significant problems. Finally, neither team indicated that they utilized any form of social levers to encourage cooperation.

Data, like Desk, was experiencing significant problems defining a common integration process across the four teams contributing code to the project:

> The build is a nightmare in this kind of web. Say I'm a component team. I depend on some other components and some other components depend on me. I want to make a drop once a week into someplace so I can bring enough stuff together so I can have an exe running on my machine that I can test. Add five components and two shells and watch the debates as to how that's going to happen.
>
> These teams are just not used to giving up that much control to someone outside the tea. Yet the only way it will work is if there is someone imposing control or some sort of penalty for breaking the build like losing face in front of 150 people.

Rather than trying to define a common process, Data first tried a "big bang" approach where they essentially "threw all of the software into a pot and stirred." As one developer recalled:

> It never made soup, and we had no idea why or where it was broken. Then people had to run around through the hallways quite a bit before we could get it to build.

At the time of the interviews, the team was experimenting with a new "roll up" approach where components were bunched together and then integrated sequentially. Each component team was responsible for making sure their code worked with existing technology previously rolled up. Most team members were dubious about this approach and indeed later interviews confirmed that it was not successful.

On the Autophone project, integration was performed by contractors in New Jersey who "lacked the full depth of knowledge" about the product. The key software development tasks (RCC and CCS) were therefore "remote to their build function," which presented complications in terms of work planning and predictability:

> Geography hurt us because its very difficult to coordinate and get a good feel for what's ready to go into a load... We never know what's going in when so we can target for it.

The integration was scheduled to be performed every two to three weeks on a pre-set schedule, but, like Data, the team rarely achieve this goal:

> We end up doing a series of corrective builds to fix some little problem to get over the hurdle. They just keep popping them off like popcorn.

TABLE E.1
COMPARISON OF PRODUCT COMPONENT INTEGRATION DESIGN

	SYSTEMIC	MANIPULATOR	CONSTRAINED	MANIPULATOR	LOCAL	REACTOR
	NETWORK	HANDPHONE	DESK	TOLLPHONE	DATA	AUTOPHONE
TASK ALLOCATION	• performed and managed by dedicated internal team group	• performed and managed by dedicated internal team group	• ownership transferred to app team responsible for latest integration problem	• performed and managed by two dedicated people outside the dept	• performed by each component team in sequential rollup; no one group responsible for management	• performed and managed off-site by two dedicated people
RESOURCE INVESTMENT (capital, human)	• very high hardware investment for fast computers • placed highly experienced technical people on build team	• invested in some additional computer queues • inexperienced lea but heavily trained and supported by feature engineer	• intended to purchase more and faster computers	• cut hardware in company wide cost cutting measure • performed by inexperienced contractors	• none	• performed by inexperienced contractors
INCENTIVE STRUCTURE (for checking in frequently and cleanly)	• appeal to professional responsibility • shame/public humiliation • humor (e.g., pay money, wear goat horns)	• appeal to professional responsibility • shame/private humiliation • personal ownership	• avoid having to perform integration next time • humor (e.g., hat, door poster)	• personal ownership • humor (e.g., bat on your chair)	• none	• none
TIMING	• very frequent and predictable schedule (on avg, twice a day, five days a week) • dynamic check in policy	• very frequent and predictable schedule (on avg, every night, five days a week) • dynamic check in policy	• very frequent but unpredictable schedule (on avg, once a week, five days a week) • dynamic check in policy	• infrequently (on avg, once every two weeks or once a month)	• Intended: once a week; Actual- never consistently achieved	• Intended: every 2-3 weeks on a fixed, pre-set schedule; Actual: never consistently achieved

APPENDIX F
SAMPLE PROJECT PERFORMANCE SURVEY

INSTRUCTIONS:
This survey is part of a doctoral dissertation at the MIT Sloan School of Management on interdependency management in large scale software development projects. The purpose of the research is to identify what types of interdependency exist in large scale projects and the alternative ways they can be coordinated and managed. The dissertation was recently awarded "Best Dissertation Proposal" in a national competition of similar studies indicating that this is an important topic with results that have broad applicability to both academic researchers and managers.

This particular survey focuses on the X project. The questions ask you to assess various aspects of task coordination on the project. For example, what was the level of task blocking? How effectively were tasks coordinated within and across the project? Some of the questions are directed at the level of functional product pieces (i.e., X1, X2, X3, and X4), while others ask you to rate the product overall.

Please answer each question honestly based on your knowledge of and experience with the project. You need not (and should not) limit your responses to the areas representing your work assignment. For example, if you primarily worked on X1 but also feel even somewhat informed about X2 please respond to questions on both pieces. I am asking for your technical and subjective judgements and opinions, not "fact."

Even if you are unsure about an answer, I prefer a rough estimate to a blank. Feel free to expand upon or qualify your answers by writing directly on the survey itself. If you have any questions or concerns about the survey, please contact me at nstauden@mit.edu or (617) 258 - 7269.

Your answers will be strictly confidential. They will be grouped with those of other people, and no individual will be identified in the final report. Nor will the name of the actual project be publicly revealed.

When you are finished with the questionnaire, please put it in the enclosed envelope, seal the envelope and sign across the back flap. Then return it via interdepartmental mail to

Thank you for your cooperation.

Nancy Staudenmayer
Doctoral Candidate
MIT Sloan School of Management
(617) 258 - 7269
nstauden@mit.edu

NAME OF PROJECT BEING EVALUATED: X

YOUR NAME: _____

SECTION 1: INTRODUCTORY QUESTIONS
The following questions will be used for coding purposes and to control for differences across projects. Where indicated, please provide the actual numbers or your best estimate for each functional product piece and the product overall.

1. How many large feature projects have you worked on (defined as greater than 30 technical head count) ? _____

2. How would you describe the level of requirements churn (after requirements were initially defined) on each functional product piece, relative to other product pieces of similar size and complexity?
(1 = Significantly below average, 2 = Somewhat below average, 3 = About average, 4 = Somewhat above average, 5 = Significantly above average)

Originating Process: _____
Terminating Process: _____
Hand Over Process: _____
Location/Registration Process: _____
X Overall: _____

3. Please indicate the level of external dependency for each functional product piece. External dependency includes software code, components and tools being developed in another subsystem or functional department. When making your assessment, please consider both the number of external providers and the percentage of code that they supplied.
(1 = Very Low, 2 = Low, 3 = Medium, 4 = High, 5 = Very High)

Originating Process: _____
Terminating Process: _____
Hand Over Process: _____
Location/Registration Process: _____
X Overall: _____

SECTION 2: INTERDEPENDENCY MANAGEMENT
The following questions address issues of task coordination and interdependency management. Again, please provide the actual numbers or your best estimate for each functional product piece and the product overall.

4. Please indicate in the table below (a-b) the approximate actual (not planned) begin and end dates for the initial design, (c) the number of additional weeks spent on rework due to adjustments in requirements or changes in code, and (d) the percentage of the elapsed time actually spent reworking each functional product piece.

	Originating Process	Terminating Process	Hand Over Process	Location/ Registration Process
(a) Initial Design Start Date (MM/YY)				
(b) Initial Design End Date (MM/YY)				
(c) Additional Weeks Spent on Rework				
(d) Percentage of Elapsed Time Actually Reworking				

5. Listed below are several possible sources of blocking or waiting time. For each blocking source, please indicate (a) the average number of times it was a factor on the project and (b) the average amount of time you spent waiting per episode. For example, if gaining access to a testing laboratory and waiting for delivery of a tool were the two primary blocking factors for Hand Over, you would indicate so as follows:

	Hand Over	
	(a) Avg. Freq	(b) Avg. Wait Time
Gaining access to a laboratory or machine	20 times	2 hours
Waiting for delivery of a component or tool	two comps	1.5 weeks

If there were other important blocking factors, please indicate so (and describe) under the 'other' category.

Insert Blocking & Waiting Table Here

6. Please rate the effectiveness of task coordination on each functional product piece in each of the following five categories. Things you may wish to consider when evaluating effectiveness are the number of people involved, the timeliness of the coordination and the success of the resolution.
(1 = Extremely Ineffective, 2 = Quite Ineffective, 3 = Effective, 4 = Quite Effective, 5 = Extremely Effective)

	Originating Process	Terminating Process	Hand Over Process	Location / Registration Process	X Overall
Coordination of tasks necessary to translate product concept into a realizable product and design and integrate product components or subsystems					
Coordination of tasks necessary to achieve technical compatibility with other hardware and software products in the technical or user environment					
Coordination of tasks across projects within the firm due to the development and use of common technologies					
Coordination of tasks associated with releasing the product into the marketplace					
Coordination of tasks associated with the sharing of physical and human resources across and within projects					

7. Please evaluate the effectiveness of working relationships both within and between the functional product pieces.
(1 = Extremely Ineffective, 2 = Quite Ineffective, 3 = Effective, 4 = Quite Effective, 5 = Extremely Effective)

Within Product Pieces:
Originating Process: _____
Terminating Process: _____

Hand Over Process: _____
Location/Registration Process: _____

Between Product Pieces:
Between Originating Process and Terminating Process: _____
Between Originating Process and Hand Over Process: _____
Between Originating Process and Location/Registration: _____
Between Terminating Process and Hand Over Process: _____
Between Terminating Process and Location/Registration: _____
Between Hand Over Process and Location/Registration: _____

8. Relative to your past project experiences, how satisfied were you with the overall final outcome on each functional product piece?
(1 = Extremely Unsatisfied, 2 = Quite Unsatisfied, 3 = Satisfied, 4 = Quite Satisfied, 5 = Extremely Satisfied)

Originating Process: _____
Terminating Process: _____
Hand Over Process: _____
Location/Registration Process: _____
X Overall: _____

(If information is unavailable at the level of functional product pieces for the following questions, just rate the product overall.)

9. To what extent did the functional piece meet or exceed its initial schedule estimate?
(1 = Exceeded schedule by more than 10%, 2 = Exceeded schedule by 10% or less, 3= Came in on schedule, 4 = Came in 10% or less under schedule, 5 = Came in more than 10% under schedule)

Originating Process: _____
Terminating Process: _____
Hand Over Process: _____
Location/Registration Process: _____
X Overall: _____

10. To what extent did the functional piece meet or exceed its initial budget estimate?
(1 = Exceeded the budget by more than 10%, 2 = Exceeded the budget by 10% or less 3= Came in on budget, 4 = Came in 10% or less under budget, 5 = Came in more than 10% under budget)

Originating Process: _____
Terminating Process: _____
Hand Over Process: _____
Location/Registration Process: _____
X Overall: _____

11. To what extend did the functional piece meet or exceed its actual customer requirements?
(1 = Failed to meet a significant number of requirements, 2 = Failed to meet some requirements, 3 = Met all of its requirements, 4 = Met some additional requirements, 5 = Met a significant number of additional requirements)

Originating Process: _____
Terminating Process: _____
Hand Over Process: _____
Location/Registration Process: _____

X Overall: _____

12. Please indicate the number of major problems in each functional piece during the first year of the product's release. Major problems are defined as system-crash level bugs (i.e., category E or H IMRs).

Originating Process: _____
Terminating Process: _____
Hand Over Process: _____
Location/Registration Process: _____
X Overall: _____

Are there any additional comments you would like to make about task coordination and the management of interdependencies on this project?

_____

_____

BIBLIOGRAPHY

Adler, P. S. (March-April 1995). Interdepartmental Interdependence and Coordination: The Case of the Design/Manufacturing Interface. Organization Science, 6, No. 2, pp. 147-167.

Adler, P. S., A. Mandelbaum, V. Nguyen, & E. Schwerer (March 1995). From Project to Process Management: An Empirically-Based Framework for Analyzing Product Development Time. Management Science, 41, No. 3, pp. 458 - 484.

Aldenderfer, M. S., & R. K. Blashfield (1984). Cluster Analysis. Vol. 44. Beverly Hills: Sage Publications.

Alexander, C. (1964). Notes on the Synthesis of Form. Boston: Harvard University Press.

Allen, T. J. (1977). Managing the Flow of Technology. Cambridge: MIT Press.

Allen, T. J. (November 1986). Organizational Structure, Information Technology and R&D Productivity. IEEE Transactions on Engineering Management, EM-33, No. 4, pp. 212-217.

Allison, G. (September 1969). Conceptual Models and the Cuban Missile Crisis. American Political Science Review, LXIII, No. 3, pp. 689 - 718.

Ancona, D., T. Kochan, M. Scully, J. Van Maanen, & D. E. Westney (1996). Managing for the Future: Organizational Behavior and Processes. Cincinnati: South-Western College Publishing.

Ancona, D. G. (1990). Outward Bound: Strategies For Team Survival In An Organization. Academy of Management Journal, 33, No. 3, pp. 334-365.

Ancona, D. G., & D. F. Caldwell (1987). Management Issues In New Product Teams in High Technology Companies. InAdvances in Industrial Relations (pp. 199-221). Greenwich: JAI Press.

Ancona, D. G., & D. F. Caldwell (1990). Beyond Boundary Spanning: Managing External Dependence in Product Development Teams. Journal of High Technology Management Research, 1, pp. 119-135.

Ancona, D. G., & D. F. Caldwell (1992). Bridging the Boundary: External Activity and Performance in Organizational Teams. Administrative Science Quarterly, 37, pp. 634-665.

Andres, H. P. (August 25-27, 1995). The Effects of Task Interdependence, Goal Conflict and Coordination Strategy on Software Project Success. Proceedings of the First Americas Conference on Information Systems. Pittsburgh.

Astley, W. G., & A. H. Van de Ven (1983). Central Perspectives and Debates in Organization Theory. Administrative Science Quarterly, 28, pp. 245-273.

Athey, S., & S. Stern (1996). An Empirical Framework for Testing Theories About Complementarity in Organizational Design, Working Paper. Cambridge: Massachusetts Institute of Technology, Sloan School of Management.

Bailetti, A. J., & J. R. Callahan (1995). Managing Consistency Between Product Development and Public Standards Evolution. Research Policy, 24, pp. 913-931.

Bailey, K. D. (1994). Typologies and Taxonomies: An Introduction to Classification Techniques, Vol. 102. Beverly Hills: Sage Publications.

Bailyn, L. (1977). Research as a Cognitive Process: Implications for Data Analysis. Quality and Quantity, 11, pp. 97-117.

Baker, W. (1992). The Networked Organization in Theory and Practice. In N. Nohria & R. Eccles (Eds.), Networks and Organizations (pp. 397 - 429). Boston: Harvard Business School Press.

Barney, J. (1991). Firm Resources and Sustained Competitive Advantage. Journal of Management, 17, pp. 99 - 120.

Bastien, D. T., & T. J. Hostager (1988). Jazz as a Process of Organizational Innovation. Communication Research, 15, pp. 582-602.

Berkowitz, L. (1957). Effects of Perceived Dependency Relations Upon Conformity and Group Expectations. Journal of Abnormal and Social Psychology, 55, pp. 350 - 354.

Bluedorn, A. C., & R. B. Denhardt (1988). Time and Organizations. Journal of Management, 14, No. 2, pp. 299-320.

Boehm, B. W. (January 1984). Software Engineering Economics. IEEE Transactions on Software Engineering, SE-10, No. 1, pp. 4-21.

Boehm, B. W., & P. N. Papaccio (October 1988). Understanding and Controlling Software Costs. IEEE Transactions on Software Engineering, 14, No. 10, pp. 1462-1477.

Bowditch, J. L., & A. F. Buono (1985). A Primer on Organizational Behavior (third ed.). New York: John Wiley & Sons, Inc.

Brooks, F. P. (1975). The Mythical Man Month. Menlo Park: Addison-Wesley.

Brown, S. L., & K. M. Eisenhardt (April 1995). Product Development: Past Research, Present Findings and Future Directions. Academy of Management Review, 20, No. 2, pp. 343 - 378.

Brown, S. L., & K. M. Eisenhardt (March 1995). Product Innovation as Core Capability: The Art of Dynamic Adaptation, Working Paper. Stanford Business School.

Brown, S. L., & K. M. Eisenhardt (May 1995). Leveraging Product Innovation: Innocent Traps, Adaptive Organizations and Strategic Evolution, Working Paper. Palo Alto: Stanford Business School.

Burns, T., & G. M. Stalker (1968). The Management of Innovation. London: Tavistock.

Burrell, G., & G. Morgan (1979). Sociological Paradigms and Organizational Analysis. London: Heinemann.

Christensen, C. M., & R. S. Rosenbloom (1995). Explaining the Attacker's Advantage: Technological Paradigms, Organizational Dynamics and the Value Network. Research Policy, 24, pp. 233-257.

Clark, K. B. (1985). The Interaction of Design Hierarchies and Market Concepts in Technological Evolution. Research Policy, 14, pp. 235-251.

Clark, K. B. (October 1989). Project Scope and Project Performance: The Effects of Parts Strategy and Supplier Involvement on Product Development. Management Science, 35, No. 10, pp. 1247-1263.

Clark, K. B., W. B. Chew, & T. Fujimoto (1987). Product Development in the World Auto Industry. Brookings Papers in Economic Activity, 3, pp. 729-771.

Clark, K. B., & T. Fujimoto (1989). Lead Time in Automobile Product Development: Explaining the Japanese Advantage. Journal of Engineering and Technology Management, 6, pp. 25-58.

Clark, K. B., & T. Fujimoto (1991). Product Development Performance. Boston: Harvard Business School Press.

Clark, K. B., & S. C. Wheelwright (1992). Organizing and Leading 'Heavyweight' Development Teams. California Management Review, 34, No. 3, pp. 9 - 28.

Cook, T. D., & D. T. Campbell (1979). Quasi-Experimentation: Design and Analysis Issues for Field Settings. Boston: Houghton Mifflin.

Crozier, M. (1964). The Bureaucratic Phenomenon. Chicago: University of C o Press.

Curtis, B., H. Krasner, & N. Iscoe (November 1988). A Field Study of the Software Design Process for Large Systems. Communications of the ACM, 31, No. 11, pp. 1268 - 1287.

Cusumano, M. A. (1991). Japan's Software Factories. New York: Oxford University Press.

Cusumano, M. A. (1992). Shifting Economies: From Craft Production to Flexible Systems and Software Factories. Research Policy, 21, pp. 453-480.

Cusumano, M. A., & C. F. Kemerer (November 1990). A Quantitative Analysis of U.S. and Japanese Practice and Performance in Software Development. Management Science, 36, No. 11, pp. 1384-1406.

Cusumano, M. A., Y. Mylonadis, & R. S. Rosenbloom (1997). Strategic Maneuvering and Mass-Market Dynamics: The Triumph of VHS Over Beta. In M. L. Tushman & P. Anderson (Eds.), Managing Strategic Innovation and Change (pp. 75 - 98). New York: Oxford University Press.

Cusumano, M. A., & K. Nobeoka (1992). Strategy, Structure and Performance in Product Development: Observations from the Auto Industry. Research Policy, 21, pp. 265-293.

Cusumano, M. A., & R. W. Selby (1995). Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets and Manages People. New York: Free Press.

Daft, R. L. (1983a). Learning the Craft of Organizational Research. Academy of Management Review, 8, pp. 539 - 546.

Daft, R. L. (1983b). Organization Theory and Design. New York: West Publishing Company.

Daft, R. L., & R. H. Lengel (May 1986). Organizational Information Requirements, Media Richness and Structural Design. Management Science, 32, pp. 554-571.

Daft, R. L., & N. B. Macintosh (1981). A Tentative Exploration into the Amount and Equivocality of Information Processing in Organizational Work Units. Administrative Science Quarterly, 26, pp. 207-224.

Dellarocas, C. (1995). A Coordination Perspective on Software Architecture: Towards a Design Handbook for Integrating Software Components. Ph.D. Dissertation, Massachusetts Institute of Technology.

Deutsch, M. (1973). The Resolution of Conflict. New Haven: Yale University Press.

Dougherty, D. (May 1992). Interpretive Barriers to Successful Product Innovation in Large Firms. Organization Science, 3, No. 2, p. 179-202.

Dougherty, D., & E. H. Bowman (Summer 1995). The Effects of Organizational Downsizing on Product Innovation. California Management Review, 37, No. 4, pp. 28-44.

Drazin, R., & A. H. Van de Ven (1985). Alternative Forms of Fit in Contingency Theory. Administrative Science Quarterly, 30, pp. 514-539.

Eccles, R. G., & N. Nohria (1992). On Structure and Structuring. In R. G. Eccles, N. Nohria, & J. D. Berkley (Eds.), Beyond the Hype (pp. 117-143). Boston: Harvard Business School Press.

Eisenhardt, K. M. (1989a). Building Theories from Case Study Research. Academy of Management Review, 14, No. 4, pp. 532-550.

Eisenhardt, K. M. (1989b). Making Fast Strategic Decisions in High Velocity Environments. Academy of Management Journal, 32, No. 3, pp. 543-576.

Eisenhardt, K. M., & B. N. Tabrizi (1995). Accelerating Adaptive Processes: Product Innovation in the Global Computer Industry. Administrative Science Quarterly, 40, pp. 84-110.

Eisenhardt, K. M., & M. J. Zbaracki (1992). Strategic Decision Making. Strategic Management Journal, 13, pp. 17-37.

Emerson, R. M. (1962). Power-Dependence Relations. American Sociological Review, 27, pp. 31-40.

Eppinger, S. D., D. E. Whitney, R. P. Smith, & D. A. Gebala (1994). A Model-Based Method for Organizing Tasks in Product Development. Research in Engineering Design, 6, pp. 1 - 13.

Factor, R. M., & W. B. Smith (July/August 1988). A Discipline for Improving Software Productivity. AT&T Technical Journal, 67, No. 4, pp. 2 - 9.

Galbraith, J. R. (1973). Designing Complex Organizations. Reading: Addison-Wesley Publishing.

Gerstberger, P. G. (1971). The Preservation and Transfer of Technology in Research and Development Organizations. Ph.D. Dissertation, Massachusetts Institute of Technology, Sloan School of Management.

Gerstenfeld, A. (1967). Technical Interaction Among Engineers and Its Relation to Performance. Ph.D. Dissertation, Massachusetts Institute of Technology, Sloan School of Management.

Glaser, B. G., & A. L. Strauss (1967). The Discovery of Grounded Theory. Chicago: Aldine Publishing.

Granovetter, M. S. (1973). The Strength of Weak Ties. American Journal of Sociology, 78, No. 6, pp. 1360-1380.

Graves, T. (1996). Code Decay, Presentation. Lucent Technologies, Bell Laboratories.

Gresov, C. (1989). Exploring Fit and Misfit with Multiple Contingencies. Administrative Science Quarterly, 34, pp. 431 - 453.

Gresov, C. (1990). Effects of Dependence and Tasks on Unit Design and Efficiency. Organization Studies, 11, No. 4, pp. 503-529.

Gresov, C., & C. Stephens (1993). The Context of Interunit Influence Attempts. Administrative Science Quarterly, 38, pp. 252 - 276.

Griffin, A., & J. Hauser (Winter 1993). The Voice of the Customer. Marketing Science, 12, No. 1, pp. 1 - 27.

Gulati, R. K., & S. D. Eppinger (May 28, 1996). The Coupling of Product Architecture and Organizational Structure Decisions, Working Paper #3906. Cambridge: Massachusetts Institute of Technology, Sloan School of Management.

Guzzo, R. A., & G. P. Shea (1987). Group Effectiveness: What Really Matters. Sloan Management Review, pp. 25 - 31.

Hackman, J. R., & R. Wagerman (1995). Total Quality Management: Empirical, Conceptual and Practical Issues. Administrative Science Quarterly, 40, pp. 309-342.

Hart, S. (1995). A Natural Resource-Based View of the Firm. Academy of Management Review, 20, No. 4, pp. 986 - 1014.

Hauptman, O. (1986). Influence of Task Type on the Relationship Between Communication and Performance: The Case of Software Development. R&D Management, 16, No. 2, pp. 127 - 139.

Hayes, R. H., & K. B. Clark (1985). Exploring the Sources of Productivity Differences at the Factory Level. In K. B. Clark (Eds.), The Uneasy Alliance (pp. 147 - 194). Boston: Harvard University Press.

Henderson, K. (Autumn 1991). Flexible Sketches and Inflexible Databases: Visual Communication, Conscription Devices and Boundary Objects in Design Engineering. Science, Technology and Human Values, 16, No. 4, pp. 448 - 473.

Henderson, R. M., & K. B. Clark (1990). Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms. Administrative Science Quarterly, 35, pp. 9 - 30.

Hrebiniak, L. (1974). Jobs Technology, Supervision and Work Group Structure. Administrative Science Quarterly, 19, pp. 395-411.

Hutchins, E. (1990). The Technology of Team Navigation. In J. Galegher, R. E. Kraut, & C. Egido (Eds.), Intellectual Teamwork (pp. 191-220). Hillsdale: Erlbaum Publishers.

Hutchins, E. (February 1991). Organizing Work by Adaptation. Organization Science, 2, No. 1, pp. 14-39.

Iansiti, M. (1994). Shooting the Rapids: System Focused Product Development in the Computer and Multimedia Environment, Working Paper #95-026. Boston: Harvard Business School.

Iansiti, M., & K. B. Clark (1994). Integration and Dynamic Capabilities: Evidence from Product Development in Automobiles and Mainframe Computers. Industrial and Corporate Change, 3, No. 3, pp. 557-605.

Imai, K., I. Nonaka, & H. Takeuchi (1985). Managing the New Product Development Process: How Japanese Companies Learn and Unlearn. In K. B. Clark (Eds.), The Uneasy Alliance (pp. 337-375). Boston: Harvard University Press.

Ito, J., & R. Peterson (1986). Effects of Task Difficulty and Interunit Interdependence on Information Processing Systems. Academy of Management Journal, 29, pp. 139 - 149.

Jaroff, L. (September 4, 1995). Trying to Top the Taurus. Time, pp. 58 - 59.

Judd, C. M., E. R. Smith, & L. H. Kidder (1991). Research Methods in Social Relations. Fort Worth: Harcourt Brace Jovanovich.

Juguilon, A. (1996). The Language Processing Method. Cambridge: Center for Quality of Management.

Katz, D., & R. L. Kahn (1966). The Social Psychology of Organizations. New York: John Wiley & Sons.

Katz, R., & T. J. Allen (1985). Project Performance and the Locus of Influence in the R&D Matrix. Academy of Management Journal, 28, No. 1, pp. 67-87.

Kawakita, J. (1977). A Scientific Exploration of the Intellect. Tokyo: Kodansha.

Kawakita, J. (1991a). The KJ Method: Chaos Speaks for Itself. Tokyo: Chuo Koron-sha.

Kawakita, J. (1991b). The Original KJ Method. Tokyo: Kawakita Research Institute.

Kemerer, C. F. (1997). Software Project Management: Readings and Cases. Chicago: Irwin Publishing.

Kmetz, J. L. (1984). An Information Processing Study of a Complex Workflow in Aircraft Electronics Repair. Administrative Science Quarterly, 29, pp. 255-280.

Kogut, B. (1988). Joint Ventures: Theoretical and Empirical Perspectives. Strategic Management Journal, 9, pp. 319-332.

Koushik, M. V., & V. S. Mookerjee (1995). Modeling Coordination in Software Construction: An Analytical Approach. Information Systems Research, 6, No. 3, pp. 220 - 254.

Krackhardt, D. (1994). Graph Theoretical Dimensions of Informal Organizations. In K. Carley & M. Prietula (Eds.), Computational Organization Theory (pp. 89 - 111). Hillsdale: Lawrence Erlbaum.

Krackhardt, D. (June, 1996). Groups, Roles and Simmelian Ties in Organizations, Working Paper. Pittsburgh: Carnegie Mellon University.

Krackhardt, D., & R. Stern (1988). Informal Networks and Organizational Crises: An Experimental Simulation. Social Psychology Quarterly, 51, No. 2, pp. 123 - 140.

Landler, M. (September 21, 1995). AT&T, Reversing Strategy, Announces a Plan to Split Into 3 Separate Companies. New York Times, Section 1, p. 1.

Lawrence, P. R., & J. W. Lorsch (1967). Organization and Environments: Managing Differentiation and Integration. Homewood: Irwin.

Lorsch, J. W. (1977). Organization Design: A Situational Perspective. In J. R. Hackman, E. E. Lawler, & L. W. Porter (Eds.), Perspectives on Behavior in Organizations (pp. 439-447). New York: McGraw Hill.

Lynch, B. (1974). An Empirical Assessment of Perrow's Technology Construct. Administrative Science Quarterly, 19, pp. 338-356.

Malone, T., K. Crowston, J. Lee, & B. Pentland (1993). Tools for Inventing Organizations: Toward a Handbook of Organizational Processes, Working Paper #3562-93. Cambridge: Massachusetts Institute of Technology, Sloan School of Management.

Malone, T. W. (February 1988). What is Coordination Theory?, Working Paper #2051-88. Cambridge: Massachusetts Institute of Technology, Sloan School of Management.

Malone, T. W., & K. Crowston (March 1994). The Interdisciplinary Study of Coordination. ACM Computing Surveys, 26, No. 1, pp. 87 - 119.

March, J. G., & H. A. Simon (1958). Organizations. New York: John Wiley.

Markus, M. L. (June 1983). Power, Politics and MIS Implementation. Communications of the ACM, 26, No. 6, pp. 430-444.

Markus, M. L., & D. Robey (1988). Information Technology and Organizational Change: Causal Structures in Theory and Research. Management Science, 34, No. 5, pp. 583-598.

McCabe, T. J. (1976). A Complexity Measure. IEEE Transactions in Software Engineering, SE-2, pp. 308 - 320.

McCord, K. R., & S. D. Eppinger (August 1993). Managing the Integration Problem in Concurrent Engineering, Working Paper #3594-93. Cambridge: Massachusetts Institute of Technology, Sloan School of Management.

McGrath, J. (1990). Time Matters in Groups. In J. Galegher, R. E. Kraut, & C. Egido (Eds.), Intellectual Teamwork (pp. 23-61). Hillsdale: Erlbaum.

McGrath, J. E., J. R. Kelly, & D. E. Machatka (1984). The Social Psychology of Time: Entrainment of Behavior in Social and Organizational Settings. In S. Oskamp (Eds.), Applied Social Psychology Annual (pp. 21-44). Beverly Hills: Sage Publications.

McKelvey, B. (September 1978). Organizational Systematics: Taxonomic Lessons From Biology. Management Science, 24, No. 13, pp. 1428-1440.

Meyer, M., & A. Lehnerd (1997). The Power of Product Platforms. New York: Free Press.

Meyer, M. H., & J. M. Utterback (Spring 1993). The Product Family and the Dynamics of Core Capability. Sloan Management Review, pp. 29 - 47.

Miles, M. B., & A. M. Huberman (1984). Analyzing Qualitative Data: A Source Book for New Methods. Beverly Hills: Sage Publishing.

Milgram, S. (1965). Some Conditions of Obedience and Disobedience to Authority. Human Relations, 18, pp. 57-75.

Miller, D. (1981). Toward a New Contingency Approach. Journal of Management Studies, 18, pp. 1 - 26.

Miller, D. (1996). Configurations Revisited. Strategic Management Journal, 17, pp. 505-512.

Miller, D. (May 1992). Environmental Fit Versus Internal Fit. Organization Science, 3, No. 2, pp. 159 - 178.

Miller, D., & J. Shamsie (1996). The Resource-Based View of the Firm in Two Environments: The Hollywood Film Studios in 1936 and 1965. Academy of Management Journal, 39, No. 3, pp. 519 - 543.

Mitchell, T. R., & W. S. Silver (1990). Individual and Group Goals When Workers are Interdependent: Effects on Task Strategies and Performance. Journal of Applied Psychology, 75, No. 2, pp. 185-193.

Morelli, M. D., S. D. Eppinger, & R. K. Gulati (August 1995). Predicting Technical Communication in Product Development Organizations. IEEE Transactions on Engineering Management, 42, No. 3, pp. 215-222.

Nadler, D. A., & M. L. Tushman (1983). A General Diagnostic Model for Organizational Behavior: Applying a Congruence Perspective. In J. R. Hackman, E. E. Lawler, & L. W. Porter (Eds.), Perspectives on Behavior in Organizations (pp. 112 - 124). New York: McGraw Hill.

Nobeoka, K., & M. A. Cusumano (November 1995). Multiproject Strategy, Design Transfer, and Project Performance: A Survey of Automobile Development Projects in the US and Japan. IEEE Transactions on Engineering Management, 42, No. 4, pp. 397-409.

Nonaka, I. (1990). Redundant, Overlapping Organization: A Japanese Approach to Managing the Innovation Process. California Management Review, Vol. 32, No. 3, pp. 27-38.

Nutt, P. C. (1984). Types of Organizational Decision Processes. Administrative Science Quarterly, 29, pp. 414-450.

Orlikowski, W. J., & J. J. Baroudi (1991). Studying Information Technology in Organizations: Research Approaches and Assumptions. Information Systems Research, 2, No. 1, pp. 1 - 28.

Pennings, J. M. (1975). Interdependence and Complementarity: The Case of a Brokerage Office. Human Relations, 28, No. 9, pp. 825-840.

Pennings, J. M. (September 1975). The Relevance of the Structural-Contingency Model for Organizational Effectiveness. Administrative Science Quarterly, 20, pp. 393-410.

Perlow, L. (1995). The Time Famine: An Unintended Consequence of the Way Time is Used at Work. Ph.D. Dissertation, Massachusetts Institute of Technology, Sloan School of Management.

Perrow, C. (1967). A Framework for the Comparative Analysis of Organizations. American Sociological Review, 32, No. 2, pp. 194-208.

Perrow, C. (1984). Normal Accidents: Living With High-Risk Technologies. New York: Basic Books.

Perry, D. E., N. A. Staudenmayer, & L. G. Votta (July 1994). People, Organizations and Process Improvements. IEEE Software, pp. 38-45.

Peteraf, M. (1993). The Cornerstones of Competitive Advantage: A Resource-Based View. Strategic Management Journal, 14, pp. 179 - 191.

Pfeffer, J., & G. R. Salancik (1978). The External Control of Organizations: A Resource Dependence Perspective. New York: Harper and Row.

Pfeffer, J., & G. R. Salancik (1983). Organizational Design: The Case for a Coalitional Model of Organizations. In J. R. Hackman, E. E. Lawler, & L. W. Porter (Eds.), Perspectives on Behavior in Organizations (pp. 102 - 111). New York: John Wiley.

Pimmler, T. U., & S. D. Eppinger (1994). Integration Analysis of Product Decompositions. Design Theory and Methodology, DE-68, pp. 343 - 351.

Piore, M. J., R. K. Lester, & K. M. Malek (March 25, 1997). The Division of Labor, Coordination and Integration: Case Studies in the Organization of Product Design in the Blue Jeans Industry, Working Paper. Cambridge: Massachusetts Institute of Technology, Sloan School of Management.

Prahalad, C. K., & G. Hamel (1990). The Core Competence of the Corporation. Harvard Business Review, 68, No. 3, pp. 79 - 91.

Resnick, M. (1992). Beyond the Centralized Mindset: Explorations in Massively-Parallel Microworlds. Ph.D. Dissertation, Massachusetts Institute of Technology.

Rockart, J. F., & J. E. Short (1989). IT and the Networked Organization: Toward More Effective Management of Interdependence (Management in the 1990s Research Program Final Report No. 1). Massachusetts Institute of Technology Sloan School of Management.

Rosenberg, N. (1982). Technological Interdependence in the American Economy. Cambridge, England: Cambridge University Press.

Roth, K. (1995). Managing International Interdependence: CEO Characteristics in a Resource Based Framework. Academy of Management Journal, 38, No. 1, pp. 200 - 231.

Rousseau, D. (1985). Issues of Level in Organizational Research: Multi-Level and Cross-Level Perspectives. In L. L. Cummings & B. M. Staw (Eds.), Research in Organizational Behavior (pp. 1 - 37).

Royce, W. W. (August 1970). Managing the Development of Large Software Systems. IEEE WESCON.

Rumelt, R., D. Schendel, & D. Teece (1991). Strategic Management and Economics. Strategic Management Journal, 12, pp. 5 - 30.

Sabbagh, K. (1995). 21st Century Jet: The Making of the Boeing 777. London: Macmillan.

Salancik, G. R. (1987). Power and Politics in Academic Departments. In M. P. Zanna & J. M. Darley (Eds.), The Compleat Academic: A Practical Guide for the Beginning Social Scientist. Hillsdale: Erlbaum.

Salancik, G. R., & H. Leblebici (1988). Variety and Form in Organizing Transactions: A Generative Grammar of Organization. Research in the Sociology of Organizations, 6, pp. 1 - 31.

Salancik, G. R., & J. Pfeffer (1974). The Bases and Use of Power in Organizational Decision-Making: The Case of a University. Administrative Science Quarterly, 19, pp. 453-473.

Salancik, G. R., & J. Pfeffer (1988). Who Gets Power and How they Hold On To It: A Strategic-Contingency Model of Power. In M. L. Tushman & W. L. Moore (Eds.), Readings in the Management of Innovation (pp. 179 - 195). New York: Harper Business.

Sanderson, S. W., & M. V. Uzumeri (1990). Strategies for New Product Development and Renewal: Design-Based Incrementilism, Working Paper. Troy: Rensselaer Polytechnic Institute, Center for Science and Technology Policy.

Schelling, T. C. (1978). Micromotives and Macrobehavior. New York: W.W. Norton.

Schiesel, S. (April 13, 1996). Lucent: A Sassy Spinoff Enjoys a Lot of Luck. New York Times, Section 3, p. 1.

Scott, W. R. (1990). Technology and Structure: An Organizational Level Perspective. In P. S. Goodman, L. S. Sproull, & Assoc. (Eds.), Technology and Organizations (pp. 109 - 143.). San Francisco: Jossey-Bass.

Selznick, P. (1957). Leadership in Administration. New York: Harper and Row.

Shiba, S., A. Graham, & D. Walden (1993). A New American TQM. Portland: Productivity Press.

Simon, H. A. (December 1962). The Architecture of Complexity. Proceedings of The American Philosophical Society, 106, No. 6, pp. 467 - 482.

Sitkin, S. (1996). Documentation as Learning, Working Paper. Durham: The Fuqua School of Business, Duke University.

Smith, R. P., & S. D. Eppinger (December 1994). Identifying Controlling Features of Engineering Design Iteration, Working Paper #3348-91. Cambridge: Massachusetts Institute of Technology, Sloan School of Management.

Smith, S. A., & M. A. Cusumano (September 1993). Beyond the Software Factory: A Comparison of "Classic" and PC Software Developers, Working Paper #96-93. Cambridge: Massachusetts Institute of Technology, Sloan School of Management.

Sneed, E. L., G. D. Huensch, J. J. Kulzer, & H. Moerkerken (November/December 1994). Distributed Switching Architecture Trends and Concepts. AT&T Technical Journal, 73(6), pp. 19 - 27.

Steward, D. V. (August 1981). The Design Structure System: A Method for Managing the Design of Complex Systems. IEEE Transactions on Engineering Management, EM-28, No. 3, pp. 71-74.

Sumner Jr., E. E. (April 20, 1993). A Researcher's View of Life In 5ESS Development, Presentation to Center for Coordination Science. Cambridge: Massachusetts Institute of Technology, Sloan School of Management.

Teece, D., G. Pisano, & A. Shuen (1990). Firm Capabilities, Resources and the Concept of Strategy, Working Paper. Berkeley: University of California, Berkeley.

Thompson, J. D. (1967). Organizations in Action. New York: McGraw Hill.

Tichy, W. F. (1992). Programming-in-the-Large: Past, Present and Future. ACM.

Tushman, M. L. (1976). Communication in Research and Development Organizations: An Information Processing Approach. Ph.D. Dissertation, Massachusetts Institute of Technology, Sloan School of Management.

Tushman, M. L. (1978). Technical Communication in R&D Laboratories: The Impact of Project Work Characteristics. Academy of Management Journal, 21, pp. 624-645.

Tushman, M. L. (1979). Work Characteristics and Sub-Unit Communication Structure: A Contingency Analysis. Administrative Science Quarterly, 24, pp. 82-98.

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

Van de Ven, A. H. (May 1986). Central Problems in the Management of Innovation. Management Science, 32, No. 5, pp. 590-605.

Van de Ven, A. H., & A. L. Delbecq (June 1974). A Task Contingent Model of Work Unit Structure. Administrative Science Quarterly, 19, No. 2, pp. 183-197.

Van de Ven, A. H., A. L. Delbecq, & R. Koenig (1976). Determinants of Coordination Modes Within Organizations. American Sociological Review, 41, pp. 322-338.

Van de Ven, A. H., & D. L. Ferry (1980). Measuring and Assessing Organizations. New York: Wiley.

Van de Ven, A. H., & M. S. Poole (1989). Methods for Studying Innovation Processes. In A. H. Van de Ven, H. L. Angle, & M. S. Poole (Eds.), Research on the Management of Innovation Grand Rapids: Harper and Row.

Van Maanen, J. (1997). Different Strokes: Qualitative Research in the Administrative Science Quarterly from 1956 to 1996, Working Paper. Cambridge: Massachusetts Institute of Technology, Sloan School of Management.

Vaughan, D. (1990). Autonomy, Interdependence, and Social Control: NASA and the Space Shuttle Challenger. Administrative Science Quarterly, 35, pp. 225 - 257.

Von Hippel, E. (1990). Task Partitioning: An Innovation Process Variable. Research Policy, 19, pp. 407-418.

Wagerman, R. (1995). Interdependence and Group Effectiveness. Administrative Science Quarterly, 40, pp. 145-180.

Weber, M. (1978). Economy & Society. Berkeley: University of California Press.

Weber, R. P. (1985). Basic Content Analysis, Vol. 49. Beverly Hill: Sage Publications.

Weick, K. E. (1974). Middle Range Theories of Social Systems. Behavioral Science, 19, pp. 357 - 367.

Weick, K. E. (1989). Theory Construction as Disciplined Imagination. Academy of Management Review, 14, No. 4, pp. 516 - 531.

Weick, K. E. (1990). Technology as Equivoque: Sensemaking in New Technologies. In P. S. Goodman (Eds.), Technology and Organizations (pp. 1-44). San Francisco: Jossey-Bass.

Weick, K. E. (March 1976). Educational Organizations as Loosely Coupled Systems. Administrative Science Quarterly, 21, pp. 1-19.

Weick, K. E., & K. H. Roberts (1993). Collective Mind in Organizations: Heedful Interrelating on Flight Decks. Administrative Science Quarterly, 38, pp. 357 - 381.

Wernerfelt, B. (1984). A Resource-Based View of the Firm. Strategic Management Journal, 5, pp. 171 - 180.

Wheelwright, S. C., & K. B. Clark (1992). Revolutionizing Product Development. New York: Free Press.

White, H. C., S. A. Boorman, & R. L. Breiger (1976). Social Structure for Multiple Networks: Blockmodels of Roles and Positions. American Journal of Sociology, 81, No. 4, pp. 730-780.

Whyte, W. F. (1955). Street Corner Society: The Social Structure of an Italian Slum. Chicago: University of Chicago Press.

Woodward, J. (1965). Industrial Organizations: Theory and Practice. London: Oxford University Press.

Yin, R. K. (1984). Case Study Research, Vol. 5. Beverly Hills: Sage Publications.

Zachary, G. P. (1994). Show-Stopper: The Breakneck Race to Create Windows NT and the Next Generation at Microsoft. New York: Free Press.

Zerubavel, E. (1981). Hidden Rhythms: Schedules and Calendars in Social Life. Chicago: University of Chicago Press.