Global Text

The New Software Engineering

The New Software Engineering

Sue Conger

For any questions about this text, please email: drexel@uga.edu

The Global Text Project is funded by the Jacobs Foundation, Zurich, Switzerland

This book is licensed under a Creative Commons Attribution 3.0 License

This edition was scanned and converted to text using Optical Character Recognition. We are in the process of converting this edition into the Global Text Project standard format. When this is complete, a new edition will be posted on the Global Text Project website and will be available in a variety of formats upon request.

THE NEW SOFTWARE ENGINEERING

CONTENTS

CHAPTER 1
OVERVIEW OF SOFTWARE ENGINEERING 1

Introduction 1
Software Engineering 2
Applications 5
  Application Characteristics 5
  Application Responsiveness 13
  Types of Applications 17
  Applications in Business 22
Project Life Cycles 23
  Sequential Project Life Cycle 23
  Iterative Project Life Cycle 29
  Learn-as-You-Go Project Life Cycle 31
Methodologies 34
  Process Methodology 34
  Data Methodology 34
  Object-Oriented Methodology 35
  Semantic Methodologies 37
  No Methodology 38
User Involvement in Application Development 39
Overview of the Book 40
  Applications 40
  Project Life Cycles 40
  Part I: Preparation for Software Engineering 40
  Part II: Project Initiation 40
  Part III: Analysis and Design 41
  Part IV: Implementation and Operations 41
Summary 41

PART I
PREPARATION FOR SOFTWARE ENGINEERING 45

CHAPTER 2
LEARNING APPLICATION DEVELOPMENT 46

Introduction 46
How We Develop Knowledge and Expertise 46
  Learning 46
  Use of Learned Information 48
  Expert/Novice Differences in Problem Solving 48
  How to Ease Your Learning Process 50
Application Development Case 50
  History of the Video Rental Business 51
  ABC Video Order Processing Task 51
  Discussion 53
Summary 54

CHAPTER 3
PROJECT MANAGEMENT 57

Introduction 57
Complementary Activities 58
  Project Planning 58
  Assigning Staff to Tasks 62
  Selecting from Among Different Alternatives 64
Liaison 67
  Project Sponsor 67
  User 67
  IS Management 69
  Technical Staff 69
  Operations 69
  Vendors 69

Other Project Teams and Departments 70
Personnel Management 70
Hiring 70
Firing 71
Motivating 71
Career Path Planning 72
Training 72
Evaluating 72
Monitor and Control 74
Status Monitoring and Reporting 74
Automated Support Tools for Project Management 79
Summary 80

CHAPTER 4
DATA GATHERING FOR APPLICATION DEVELOPMENT 83

Introduction 83
Data Types 83
Time Orientation 84
Structure 84
Completeness 86
Ambiguity 86
Semantics 86
Volume 86
Data Collection Techniques 87
Single Interview 87
Meetings 92
Observation 94
Temporary Job Assignment 95
Questionnaire 95
Document Review 97
Software Review 98
Data Collection and Application Type 98
Data Collection Technique and Data Type 98
Data Type and Application Type 99
Data Collection Technique and Application Type 101
Professionalism and Ethics 102
Ethical Project Behavior 103
Ethical Reasoning 106
Summary 107

PART II
PROJECT INITIATION 111

CHAPTER 5
ORGANIZATIONAL REENGINEERING AND ENTERPRISE PLANNING 113

Introduction 113
Conceptual Foundations of Enterprise Reengineering 113
Planning Reengineering Projects 117
Reengineering Methodology 119
Identify Project Sponsor 120
Assign Staff 121
Scope the Project 122
Create a Schedule 123
Identify Mission Statement 124
Gather Information 124
Summary of the Architectures 125
Translating Information into Architecture 128
Architecture Analysis and Redesign 133
Implementation Planning 140
Enterprise Analysis Without Organization Design 143
Automated Support Tools for Organizational Reengineering and Enterprise Analysis 143
Summary 143

CHAPTER 6
APPLICATION FEASIBILITY ANALYSIS AND PLANNING 148

Introduction 148
Definition of Feasibility Terms 148

Feasibility Activities 150
Gather Information 150
Develop Alternative Solutions 159
Evaluate Alternative Solutions 170
Plan the Implementation 172
Evaluate Financial Feasibility 187
Document the Recommendations 193
Automated Support Tools for Feasibility Analysis 194
Summary 195

PART III
ANALYSIS AND DESIGN 199

Introduction 199
Application Development as a Translation Activity 202
Organizational and Automated Support 209
Joint Application Development 210
User-Managed Application Development 216
Structured Walk-Throughs 217
Data Administration 218
CASE Tools 222
Summary 225

CHAPTER 7
PROCESS-ORIENTED ANALYSIS 227

Introduction 227
Conceptual Foundations 227
Summary of Structured Systems Analysis Terms 228
Structured Systems Analysis Activities 231
Develop Context Diagram 234
Develop Data Flow Diagram 241
Develop Data Dictionary 261
Automated Support Tools 270
Summary 270

CHAPTER 8
PROCESS-ORIENTED DESIGN 279

Introduction 279
Conceptual Foundations 279
Definition of Structured Design Terms 280
Process Design Activities 293
Transaction Analysis 294
Transform Analysis 295
Complete the Structure Chart 303
Design the Physical Database 310
Design Program Packages 312
Specify Programs 317
Automated Support Tools for Process-Oriented Design 319
Strengths and Weaknesses of Process Analysis and Design Methodologies 322
Summary 324

CHAPTER 9
DATA-ORIENTED ANALYSIS 328

Introduction 328
Conceptual Foundations 329
Definition of Business Area Analysis Terms 329
Business Area Analysis Activities 339
Develop Entity-Relationship Diagram 339
Decompose Business Functions 356
Develop Process Dependency Diagram 363
Develop Process Data Flow Diagram 372
Develop and Analyze Entity/Process Matrix 381
Software Support for Data-Oriented Analysis 387
Summary 387

CHAPTER 10
DATA-ORIENTED DESIGN 391

Introduction 391
Conceptual Foundations 391
Definition of Information Engineering Design Terms 392

Information Engineering Design 401
Analyze Data Use and Distribution 401
Define Security, Recovery, and Audit Controls 410
Develop Action Diagram 424
Define Menu Structure and Dialogue Flow 438
Plan Hardware and Software Installation and Testing 445
Automated Support Tools for Data-Oriented Design 453
Summary 456

CHAPTER 11
OBJECT-ORIENTED ANALYSIS 459

Introduction 459
Conceptual Foundations of Object-Oriented Analysis 459
Definition of Object-Oriented Terms 461
Object-Oriented Analysis Activities 463
Develop Summary Paragraph 464
Identify Objects of Interest 468
Identify Processes 473
Define Attributes of Objects 479
Define Attributes of Processes 483
Perform Class Analysis 486
Draw State-Transition Diagram 492
Automated Support Tools for Object-Oriented Analysis 497
Summary 497

CHAPTER 12
OBJECT-ORIENTED DESIGN 501

Introduction 501
Conceptual Foundations 501
Definition of Object-Oriented Design Terms 502
Object-Oriented Design Activities 508
Allocate Objects to Four Subdomains 509
Draw Time-Order Event Diagram 512
Determine Service Objects 517
Develop Booch Diagram 521
Define Message Communications 525
Develop Process Diagram 529
Develop Package Specifications and Prototype 533
What We Know and Don't Know from OOA and OOD 534
Automated Support Tools for Object-Oriented Design 534
Summary 535
Appendix: Unix/C++ Design of ABC Rental 539

CHAPTER 13
SUMMARY AND FUTURE OF SYSTEMS ANALYSIS, DESIGN, AND METHODOLOGIES 554

Introduction 554
Comparison of Methodologies 554
Information Systems Methodologies Framework for Understanding 555
Humphrey's Maturity Framework 562
Comparison of Automated Support Environments 565
Research Relating to Analysis, Design, and Methodologies 568
Business and Technology Trends that Impact Application Development 569
Legacy Systems 570
Repositories and Data Warehouses 570
Client/Server 571
Multimedia 572
Globalization 572
Summary 574

CHAPTER 14
FORGOTTEN ACTIVITIES 579

Introduction 579
Human Interface Design 579
Conceptual Foundations of Interface Design 579
Develop a Task Profile 580
Option Selection 590
Functional Screen Design 601
Presentation Format Design 605
Field Format Design 620
Conversion 625
Identify Current and Future Data Locations 626

Define Attribute Edit and Validate Criteria 627
Define Data Conversion Activities and Timing 627
Select and Plan an Application Conversion Strategy 627
ABC Conversion Strategy 629
User Documentation 631
Mix of On-Line and Manual Documentation 631
Automated Support Tools for Forgotten Activities 632
Summary 633

PART IV
IMPLEMENTATION AND MAINTENANCE 637

Introduction 637

CHAPTER 15
CHOOSING AN IMPLEMENTATION LANGUAGE 640

Introduction 640
Characteristics of Languages 640
Data Types 640
Data Type Checking 641
Language Constructs 642
Modularization and Memory Management 645
Exception Handling 646
Multiuser Support 646
Nontechnical Language Characteristics 647
Comparison of Languages 650
SQL 650
Focus 656
BASIC 656
COBOL 656
Fortran 657
C 657
Pascal 657
PROLOG 658
Smalltalk 659
Ada 659
Programming Language Evaluation 660
Language Matched to Application Type 660
Language Matched to Methodology 661
Automated Support for Program Development 662
Summary 662

CHAPTER 16
PURCHASING HARDWARE AND SOFTWARE 666

Introduction 666
Request for Proposal Process 667
Develop and Prioritize Requirements 667
Develop Schedule and Cost 667
Develop Request for Proposal 668
Manage Proposal Process 669
Evaluate Proposals and Select Alternatives 670
Informal Procurement 670
Contents of RFP 670
Vendor Summary 670
Required Information 671
Schedule of RFP Process 674
Description of Selection Processes 674
Vendor Response Requirements 675
Standard Contract Terms 677
Hardware 677
Functionality 677
Operational Environment 678
Performance 678
Software 678
Needs 679
Resources 679
Performance 680
Flexibility 680
Operating Characteristics 680
RFP Evaluation 681
General Evaluation Guidelines 681
Automated Support Tools for Evaluation 687
Summary 687

CHAPTER 17
TESTING AND QUALITY ASSURANCE 690

Introduction 690
Testing Terminology 690
Testing Strategies 694
  Black-Box Testing 695
  White-Box Testing 697
  Top-Down Testing 699
  Bottom-Up Testing 702
  Test Cases 702
  Matching the Test Level to the Strategy 704
Test Plan for ABC Video Order Processing 706
  Test Strategy 706
  Unit Testing 710
  Subsystem or Integration Testing 718
  System and Quality Assurance Testing 723
Automated Support Tools for Testing 729
Summary 732

CHAPTER 18
CHANGE MANAGEMENT 735

Introduction 735
Designing for Maintenance 735
  Reusability 735
  Methodology Design Effects 738
  Role of CASE 740
Application Change Management 741
  Importance 741
  Change Management Procedures 742
  Historical Decision Logging 744
  Documentation Change Management 744
Software Management 749
  Introduction 749
  Types of Maintenance 749
  Reengineering 751
Configuration Management 751
  Introduction 751
  Types of Code Management 752
  Configuration Management Procedures 755
Automated Tools for Change Management 756
  Collaborative Work Tools 756
  Documentation Tools 758
  Tools for Reverse Engineering of Software 759
  Tools for Configuration Management 759
Summary 759

CHAPTER 19
SOFTWARE ENGINEERING AS A CAREER 764

Introduction 764
Emerging Career Paths 764
Careers in Information Systems 765
  Level of Experience 765
  Job Type 767
Planning a Career 772
  Decide on Your Objective 773
  Define Duties You Like to Perform 773
  Define Features of the Job 773
  Define Features of the Organization 775
  Define Geographic Location 777
  Define Future-Oriented Job Components 777
  Search for Companies That Fit Your Profile 778
  Assess the Reality of Your Ideal Job and Adjust 778
Maintaining Professional Status 780
  Education 781
  Professional Organizations 781
  User Organizations 783
  Accreditation 785
  Read the Literature 785
Automated Support Tools for Job Search 786
Summary 787

APPENDIX
CASES FOR ASSIGNMENTS 790

Abacus Printing Company 790
AOS Tracking System 791
The Center for Child Development 792
Course Registration System 794
Dr. Patel's Dental Practice System 795
The Eagle Rock Golf League 796
Georgia Bank Automated Teller Machine System 796
Summer's Inc. Sales Tracking System 797
Technical Contracting, Inc. 798
XY University Medical Tracking System 799

Glossary 801
Index 811

PREFACE

As we move toward the 21st century, the techniques, tools, technologies, and subject matter of applications development are changing radically. Globalization of the work place is impacting IS development as well, by pressuring organizations to strive for competitive advantage through automation, among other methods. Strategic IS, reusable designs, downsizing, right-sizing, multimedia databases, and reusable code are all discussed in the same breath. Methodologies are being successfully coupled to computer-aided software engineering environments (CASE); yet object-oriented methodologies, which are being touted as the panacea for all problems, have not yet been fully automated . . . or even fully articulated. Few if any tools, methods or techniques address the needs of artificial intelligence and expert system development, which are currently driven by the program language being used for development. New technologies for true distribution of processing are maturing, and integration across hardware and software platforms is the major IS concern in multiple industries [*Computerworld*, 10/15/90].

IS professionals must be jacks-of-all-trades as never before, but there is also increased demand for domain experts who are intimately familiar with all aspects of a particular business area, such as money transfer in banking. It is difficult for any one person to be both expert and generalist. But there are many systems developers—I call them software engineers—who do possess these attributes. Today's ideal software engineer is familiar with the alternatives, trade-offs and pitfalls of methodologies (notice the plural form), technologies, domains, project life cycles, techniques, tools, CASE environments, hardware, operating systems, databases, data architectures, methods for user involvement in application development, software, design trade-offs for the problem domain, and project personnel skills. Few professionals acquire all these skills without years of experience including both continuing education and variations in project assignments, company type, and problem type. This book attempts to discuss much of what should be the ideal software engineer's project-related knowledge and theoretical background in order to facilitate and speed the process by which novices become experts.

The goal of this book, then, is to discuss project planning, project life cycles, methodologies, technologies, techniques, tools, languages, testing, ancillary technologies (e.g., database), and computer-aided software engineering (CASE). For each topic, alternatives, benefits and disadvantages are discussed.

For methodologies, one major problem is that most writing on methods of development concentrates on *what* the analyst does. It is up to the individual instructor and/or student to develop the *how* knowledge. Yet, the what knowledge is easy and takes very little time to learn. If I say, "The first step in object-oriented methodology is to make a list of objects," that sounds like a simple step. I may understand *what* I'm to do, but not *how* to do it. This book is intended to shed some light on the *how* information. One technique used to facilitate the learning process is to develop the same case problem in each methodology, highlighting the similarities, differences, conceptual activities, decision processes, and physical representations. Another technique is to provide cases in the appendix that can be used throughout the text for many assignments, thus allowing the student to develop a detailed-problem understanding and an understanding of how the problem is expressed in different methodologies and using different techniques.

A related problem in software engineering texts is that little information is available on current research and future directions. Information systems development is a 30-year old activity that is beginning to show some signs of maturity, but is also constantly changing because the type of systems we automate is constantly changing. Research in every

area of software development, from enterprise analysis through reengineering 20-year-old systems, is taking place at an unprecedented rate. Moreover, the landscape of system development will change radically in the next 20 years based on the research taking place today. This text attempts to highlight and synthesize current research to identify future directions.

Many software engineering texts never discuss problems attendant with methodologies. This text attempts to discuss methodologies in the context of their development and how they have evolved to keep pace with new knowledge about system development. Both useful and not-so-useful representation techniques will be identified. The book may be controversial in this regard, but at least the knowledge that there *are* problems with methods should remove some of the prevailing attitudes that there are right and wrong ways to complete everything. Unfortunately, *no* methodology is complete enough to guarantee the same results from two different analysts working independently, so interpretations differ. I try to identify my interpretations and generalizations throughout the text.

The book is case-oriented in several ways. First, a sample project is described, designed, and implemented using each of the techniques discussed. Second, cases for in-class development are provided. Third, cases for homework assignments are also provided. Research on learning has revealed that we learn best through practice, analysis of examples, and more practice. For each topic, an example of both acceptable and unacceptable deliverables is provided, with discussion of the relative merits and demerits of each. Through repeated use of different cases, students will learn both the IS topics and something about problem domains that will carry over into their professional lives.

Finally, this text has a bias toward planning, analysis, and design activities even though the entire life cycle is discussed. This bias is partly due to practical and space limitations; however, it is also because of the realities of changing software engineering work. CASE promises to remove much of the programming from business application development by automating the code generation process. Although languages are discussed, the discussion focuses on how *to choose* the correct language for an application based on language characteristics, rather than on how *to program* in the language.

The audience for this text includes business, computer information systems, and computer science students. The courses for which this text is appropriate include software engineering, advanced system analysis, advanced topics in information systems, and IS project development. Computer software engineering is moving away from a concentration on developing the perfect program to a realization that even perfect programs never work in isolation. Program *connections* are significantly more important than individual program code. Thus, even computer scientists are recognizing a need for methodologies, techniques for system representation, and language selection.

The text was originally planned to accommodate either quarter or semester classes. I have taught this material in both. While the written material is longer than anticipated, I believe the book can be covered in one quarter because there are usually more contact hours with students. One of my goals was a book that did not require much additional outside material to supplement the text; I hope this goal was met. Much of the bulk is explaining the *how* processes in Chapters 7–12, and these should be covered in class to discuss alternatives, possible flaws in my thinking, and so on. If programming is also included in the course, I suggest development of a two-quarter (or semester) sequence that includes software engineering through system design in the first course and the remaining subjects in the second course.

Every school seems to offer courses on "Advanced Topics in Systems Development" or Advanced Systems Analysis" or "IS Development Project" that frequently use no book because nothing covers all the desired topics. This book attempts to provide for these courses. Advanced systems analysis and development courses all tend to concentrate on alternatives during the design process from which decisions must be made. The typical systems analysis course might discuss one technique for each major topic area: enterprise modeling, data modeling, process modeling, program design. That alternatives are available is certainly mentioned, but there is simply not enough time to teach all topics,

nor are students able to assimilate much information about alternatives without becoming hopelessly confused. Advanced courses try to broaden the knowledge base of students with discussions of alternatives in each area. Even in these courses, without a hands-on orientation and concrete examples to use for reference, the number of topics and alternatives is necessarily limited. The use of a single case throughout the text, together with cases for home/school work practice, should broaden the number of topic areas that can be covered adequately in a one-semester course.

ACKNOWLEDGMENTS

No textbook is published without the involvement of many people and I would like to acknowledge those who have helped bring this book to fruition. I am grateful, first, to my husband Dave and my daughter Katie, who have put up with haphazard meals and an absent-minded wife and mother for a long time. Baby-sitters were especially important when I commuted four hours a day. I thank Elaine Black, Lis Nielsen, Sarah Cropley, Louise Shipman, Jacquie Draycott, Ellen Crawford, and Angela Moore.

Also, I wish especially to thank Peter Keen for his unfailingly good advice and uplifting moral support. I have never before worked with someone so free with great ideas. Frank Ruggirello, who actually got me moving and enlisted the supportive and helpful reviewers, played a special part in the project. I want to thank the reviewers, who put up with my typos and grammar long enough to read about the ideas I am attempting to convey. Their comments have materially enhanced the final quality of this book. These reviewers include: Donald R. Chand, Bentley College; Dale D. Gust, Central Michigan University; Lavette Teague, California State Polytechnic University–Pomona; Jon A. Turner, New York University; Douglas Vogel, University of Arizona; Connie E. Wells, Georgia State University; J. Christopher Westland, University of Southern California; and Susan J. Wilkins, California Polytechnic University–Pomona. My thanks for the helpful and supportive comments.

Next, the Wadsworth "family" has been supportive throughout the work, including Kathy Shields, Rhonda Gray, Tamara Huggins, Peggy Mehan, Greg Hubit, and Janet Hansen. Martha Ghent, the copy editor, deserves special mention. Having never worked through the copy process before, I had no idea what was done. Martha was easy to work with and taught me how to improve both my writing and my punctuation.

Friends and colleagues, who have given me anecdotes, support, ideas, and comments, were invaluable. The friends who have materially contributed to this project include Peter Keen, Connie Wells, Judy Wynekoop, Irene Auerbach, Chung Pin Chuang, Karen Loch, Kuldeep Kumar, Scott Owen, Iris Vessey, Nancy Russo, Alex Heslin, Paul Haldeman, Marty Fraser, Eph McLean, Ross Gagliano, Jim Senn, Mike Palley, Dorothy Dologite, Ronnie Wilkes, Jong Kim, Seok Jung Yoon, Dennis Strouble, Mary Alexander, Ted Stohr, and the many student 'guinea pigs' (mine and others) from Georgia State University, Baruch College (CUNY), University of Texas–Arlington, University of Dallas, and New York University. Thank you all.

Finally, I would like to thank you, the reader, for buying this book and taking the trouble to read even a portion of it. If you should disagree with my reasoning or find errors or omissions that should be corrected, I would be grateful for suggestions and correspondence.

Sue Conger
Dallas, Texas

CHAPTER 1

OVERVIEW OF SOFTWARE ENGINEERING

INTRODUCTION

Businesses around the world depend more and more on software in the very basics of their operations. U.S. firms alone have 100 billion lines of program code in use today. This code cost $2 *trillion* to create and costs $30 billion *a year* to maintain. The typical Fortune 1000 company maintains 35 million lines of code. Quality of software design and quality of business service are increasingly linked. We take for granted the everyday convenience we gain from reservation, telephone, automated teller, and credit card authorization applications. We can take these conveniences for granted until they 'crash' or have a 'bug.' Software engineers (SEs) developed those systems. The engineering skills they apply to developing applications go far beyond the writing of good programs. The skills SEs need are to deploy and manage the data, software, hardware, and communications business assets of a corporation. These computer-related assets now account for almost half of all U.S. business investment.

Software engineers are skilled professionals who can make a real difference to business profitability. The word *professional* is key here. Software development is notoriously difficult to manage; software projects are routinely over budget and behind schedule. Computer programmers are legendary for their lack of understanding of, or interest in, business. SEs who are professionals are more likely to manage and deliver a quality project on time and within budget. One goal of this text is to challenge you to set high standards for personal excellence: to become a professional and to make a difference.

This chapter introduces you to the book and the topics to be covered in more detail in later chapters. The objectives of this chapter are to: (1) review what you might already know, (2) give you a vocabulary for discussing applications, and (3) introduce the topics of this text. Use this chapter to learn basic definitions and to begin building a mental picture of how different approaches to software engineering work. You will learn the details in later chapters.

Software engineering is the systematic development, operation, maintenance, and retirement of software. Software engineers (SEs) have a mental 'tool kit' of techniques to use in developing applications. As students of information systems, you know bits and pieces of the tool kit. This text will show you how to use the tools together, and will add to what you already know. For instance, you should already know data flow diagrams (DFDs). DFDs are one of many tools, including new diagrams such as process hierarchies, process dependencies, and object diagrams. No one tool is ideal or complete. The SE knows how to select the tools, understanding their strengths and weaknesses. Most of all, an SE is not limited to a single tool he or she tries to force-fit to all situations.

Software engineering is important because it gives you a foundation on which to develop a career as an information systems development professional. At the end of the course, you will understand a variety of approaches to analyzing, designing, programming, testing, and maintaining information systems in organizations. You will know the alternatives for developing applications, and you will know how and when to select from among them. You will be able to compare and contrast methodology differences and will know the major computer-aided software engineering (CASE) tools that support each methodology. Finally, you will have an appreciation of the roles of software engineers and how they work with project managers in application development.

In the next section, you will learn what it means to be a *software engineer*. Then, a framework for discussing applications will help you categorize characteristics, technologies, and types of *applications* in business organizations. The next several sections guide you through alternatives for overall management of the application development *process*. The last section briefly outlines the remaining chapters of the book. Along the way, major terms are highlighted in bold print and defined so you can begin to form a mental picture of the alternative approaches to software engineering work.

SOFTWARE ENGINEERING

This conversation might be overheard in a manager's office:

Consultant Manager: "All right, Mary, tomorrow you start work on the rental processing application we are developing for ABC's Video Company. Mary, you are the project manager. Are you ready?"

Mary: "Yes, our first job is to find out more about the application. Then, Sam and I will decide our approach to development and the documentation that is needed. ABC's manager, Vic, is willing to provide us with whatever we need. Then, we will complete a feasibility analysis and . . ."

Mary is describing the first steps used by a modern software engineer in the development of a computer-based application. **Software** is the sequences of instructions in one or more programming languages that comprise a computer application to automate some business function. **Engineering** is the use of tools and techniques in problem solving. Putting the two words together, **software engineering** is the systematic application of tools and techniques in the development of computer-based applications.

A **software engineer** is a person who applies a broad range of application development knowledge to the systematic development of application systems for organizations. Software engineers used to think of their job as conscientious development of well-structured computer programs. But, as the field evolved, systems analysis as a task appeared along with systems analysts, the people who perform that task. Now, there is a proliferation of techniques, tools, and technologies to develop applications. Software engineers' jobs have evolved to now include evaluation, selection, and use of specific systematic approaches to the development, operation, maintenance, and retirement of software. **Development** begins with the decision to develop a software product and ends when the product is delivered. **Operations** is the daily processing that takes place. **Maintenance** encompasses the changes made to the logic of the system and programs to fix errors, provide for business changes, or make the software more efficient. **Retirement** is the replacement of the current application with some other method of providing the work, usually a new application.

Fundamental skills of software engineers include

1. How to identify, evaluate, choose, and implement an appropriate methodology[1] and CASE tools
2. How and when to use prototyping
3. How and when to select hardware, software, and languages

1 Technically, the term *methodology* means 'the study of methods.' In information systems work, the term is colloquially accepted to mean a collection of tools and techniques used to represent an application's requirements. We use the Information Systems (IS) form of the term meaning 'collections of tools and techniques.' CASE software automates the use of the tools and techniques.

EXAMPLE 1-1

NEW YORK BANK

In 1970, NY Bank wanted to be first in the New York market with an automated teller machine (ATM) system. The bank contracted with a large computer vendor to build custom ATM software using the vendor's equipment. Because telecommunications technology was in its infancy at the time, and distributed processing did not exist when the system was installed in 1971, the two ATM locations used small, local computers to record transactions. The computers did not communicate with each other. Nor could they check customer balances to verify availability of funds for transactions.

Within one month of the opening of the ATMs, one customer had, in one 24-hour period, withdrawn $200,000 from the two machines. The customer's balance in his checking account was $50. One month, and one similar user later, NY Bank shut its ATM offices, canceled the contract with the vendor, and wrote off $30 million in development costs. Shortly after, NY Bank began another project to develop a "second-generation" ATM system in which balances were checked via communications with a centralized database application.

4. How to manage activities associated with configuration management, planning, and control of the development process
5. How to select computer languages and develop computer programs
6. How and which project testing techniques to apply
7. How to choose and use software maintenance techniques
8. How to evaluate and decide when to retire applications

The **goals of a software engineer** are to produce a high quality product and to enjoy a high quality development process. The *product* of a software engineering effort is a delivered, working computer system, some examples of which include:

- Accounts receivable processing
- Order processing
- Inventory monitoring and maintenance
- Decision support for overnight funds investment
- Collateralized mortgage obligation cost determination
- Insurance reimbursement processing
- Funds transfer processing
- Early warning system for problems with critical success factors
- Query processing for a customer information database

A quality **SE product** is

- on time
- within budget
- functional, i.e., does what it is supposed to do
- friendly to users
- error free
- flexible
- adaptable

In addition to a quality product, quality of process is desirable. The **software engineering process** describes the steps it takes to develop the system. We begin a development project with the notion that there is a problem to be solved via automation. The process is how you get from problem recognition to a working solution. A quality process is desirable because it is more likely to lead to a quality product. The process followed by a project team during the development life cycle of an application should be orderly, goal-oriented, enjoyable, and a learning experience.

That we try to apply engineering discipline to software development does not mean that we have all the answers about how to build applications. On

EXAMPLE 1-2

TUV INSURANCE COMPANY

In 1991, TUV Insurance Company began a restructuring project for an annuity premium processing application. The project team consisted of a manager who had been with the company 20 years and two analysts who were new hires in 1991. The two new people, Jacquie and Ted, both wanted to apply information engineering techniques to the work. They discussed the methodology with the project manager and clients who agreed to try a modified form of the new methodology.

During the first phase of development, an entity-relationship diagram was developed with accompanying data dictionary and process decomposition descriptions. The project team and users were pleased with the results.

When the schedule for development was presented to the user, it was estimated that the entire project would take 18 months using information engineering. The client balked. He said, "The history of this company is that any project over one year never gets done. Therefore, I won't approve this. Just design me a file, like we have always done, and then add on the processing to create and maintain the file. When you revise the schedule to use this approach—file design and its processing—make sure it is under a year."

the contrary, we still build systems that are not useful and thus are not used. For example, New York Bank lost millions of dollars (see Example 1-1) because they used the wrong technology. Part of the reason for continuing problems in application development, like those of NY Bank, is that we are constantly trying to hit a moving target. Both the technology and the type of applications needed by businesses are constantly changing and becoming more complex. Our ability to develop and disseminate knowledge about how to successfully build systems for new technologies and new application types seriously lags behind technological and business changes. This book discusses where the field is now, and where it is likely to be in the 21st century. One thing is certain: The way we build systems in 10 years will be vastly different from the way we build systems today. The existing techniques that we expect to be using into the next century are discussed in this text. There will be other techniques yet to be developed, and you will have to learn to use them, too. One purpose of this text is to provide a foundation for learning to learn software engineering.

Another reason for continuing problems in application development is that we aren't always free to apply the techniques we know work best. Why? you might ask. Organizations may *know* the right things to do, but it is hard to change habits and cultures from the *old way* of doing things, as well as get users to agree with a new sequence of events or an unfamiliar format for documentation. As Example 1-2 shows, compromise is possible. The example illustrates some problems with revolutionary change and how *revolution* can be pared down to *evolution* and made acceptable.

You might ask then, if many organizations don't use good software engineering practices, why should I bother learning them? There are two good answers to this question. First, if you never know the right thing to do, you have no chance of ever using it. Second, organizations will frequently accept evolutionary, small steps of change instead of revolutionary, massive change. You can learn individual techniques that can be applied without complete devotion to one way of developing systems. In this way, software engineers can speed change in their organizations by demonstrating how the tools and techniques enhance the quality of both the product and the process of building a system.

APPLICATIONS

Software engineering is the building of applications. An **application** is the set of programs[2] that automate some business task. Businesses are made up of functions such as marketing, accounting, manufacturing, and personnel. Each function can be divided into work processes for which it is responsible. For instance, marketing is responsible for sales, advertising, and new product development. Each process can be separated into its specific tasks. Sales, for instance, requires maintaining customer relations, order processing, and customer service. Applications could support each task individually. Conversely, one marketing application could perform all tasks, integrating the information they have in common.

All applications have some common and some unique features. One problem is that there is no agreed upon way to discuss these similarities and differences. In this book, we present three dimensions of applications to simplify and clarify this discussion. The dimensions of applications are characteristics, responsiveness, and type. **Characteristics** are common to all applications and include data, processes, constraints, and interfaces. The section on application characteristics is first and should be a review. **Responsiveness** defines the underlying time orientation of the application as batch, on-line, or real-time. By knowing the time orientation of an application, we can define minimal technology required to support the application. **Type** defines the business orientation of the application as transactional, query, decision, or intelligent.

Application Characteristics

This section is about shared characteristics of applications: data, processes, constraints, and interfaces (see Figure 1-1). All applications: (1) act on data and require data input, output, storage and retrieval; (2) imbed commands that transform data from one state to another state based on and constrained by business rules; and (3) have some human interfaces and may have one or more computer interfaces. Application types vary in the extent to which these characteristics are known, defined, and understood. Each of the characteristics is discussed below. Since this is review, if you can define the terms in bold print, you might skip to the next section: Application Responsiveness.

Data

Data are the raw material (numbers and letters) that relate to each other to form fields (attributes), which define entities (see Figure 1-2). An entity is some definable class of people, concrete things, concepts, or events about which an application must maintain data. Examples of each entity type are customers, warehouses, departments, or orders, respectively. Data and entities can be described independently of their processing rules. Examples of data definition aids are entity relationship diagrams (see Figure 1-3) and third normal form linkage diagrams (see Figure 1-4).

Data requirements in applications include input, output, storage, and retrieval.

INPUT. Data **inputs** are data that are outside the computer and must be entered using some input device. Devices used for getting data into the computer include, for example, keyboard,[3] scanner, and transmission from another computer.

OUTPUT. **Output** is the opposite of input; that is, outputs are data generated to some media that is outside the computer. Common output devices include printers, video display screens, other computers, and microform equipment (e.g., microfiche, microfilm).

STORAGE AND RETRIEVAL. Data **storage** describes a physical, machine-readable data format for data, while data **retrieval** describes the means you use to access the data from its storage format. Storage and retrieval go together both conceptually and in software. Storage format and retrieval access

2 A program is composed of instructions that perform some well-defined task. Sometimes there are many tasks, composed of millions of instructions in an application. When there are many tasks, they are split into programs. This **decomposition** into subtasks which relate to programs is one topic in the chapters on application design.

3 Attached to video display or maybe some typewriter-like terminal, touch-tone phone, etc.

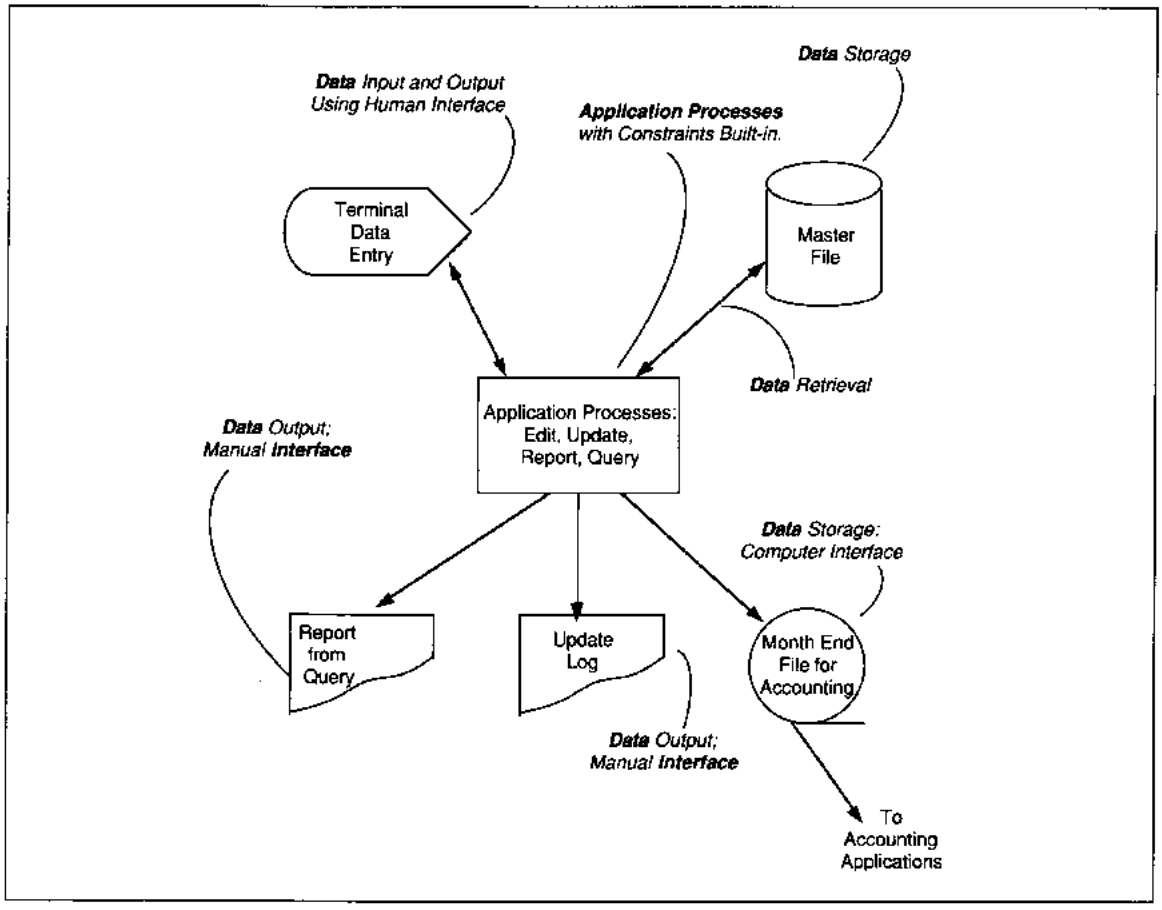


FIGURE 1-1 Application Characteristics

may be defined by your use of purchased software (such as a database management system's method, e.g., Oracle, DB2, or Adabas[4]), or may be defined by an access method provided by a hardware vendor (e.g., IBM's virtual sequential access method—VSAM).

Data storage require two types of data definition: logical and physical. The logical definition of data describes the way a user thinks about data, that is, the **logical data model**. These definitions might be relational, hierarchic networked, or object-oriented. **Relational logical data models** are arranged in tables of rows and columns. **Hierarchic logical data models** define one-to-many relationships in a tree-shaped model that resembles an organization chart. **Network logical data models** define many-to-many relationships.

Object-oriented logical data models (OOLDMs) combine hierarchic and network logical models to form a lattice-structured hierarchy. OOLDMs are more specific in identifying classes and subclasses of objects in a hierarchy. A **class** is a set of data entities that share the defining characteristic. For instance, the class *customer* might have

4 Oracle is a trademark of the Oracle Corporation. DB2 is a trademark of the IBM Corporation. Adabas is a trademark of Software AG, Inc.

123426789SandraJaniceJones21NorthfieldRoadFreeportGA442404042214960
is less meaningful than if it is split into related fields of information:

ENTITY: Person

ATTRIBUTES:	INSTANCE of Person
Social Security Number:	123-42-6789
Name:	Sandra Janice Jones
Address Line:	21 Northfield Road
City:	Freeport
State:	GA
Zip Code:	44240
(Area Code) Telephone:	(404) 221-4960

FIGURE 1-2 Attribute-Entity Example

subclasses for *cash* and *credit* customers. The lattice network arrangement allows relationships to remain unconstrained by a data management software conceptualization.

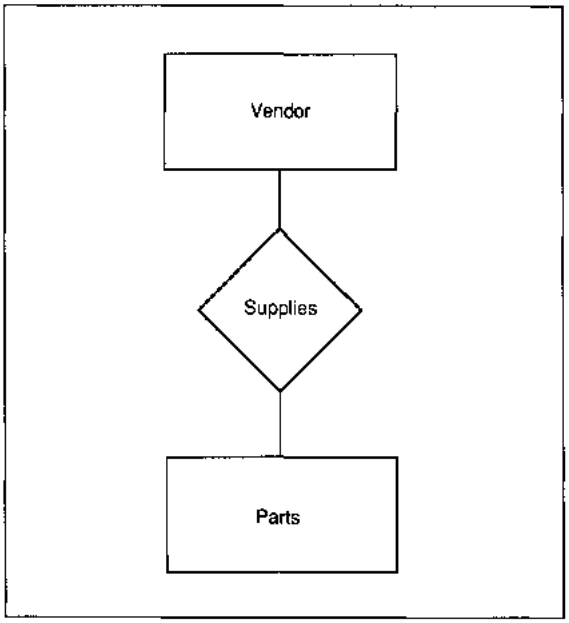


FIGURE 1-3 Entity-Relationship Example

Figures 1-5, 1-6, 1-7, and 1-8 show logical data structured in each of the four ways for vendor-parts information. Notice that the network and relational diagrams are somewhat similar. The relational model uses logical data connections to reflect relationships, while the network model uses physical address pointers imbedded in the data structure to maintain the relationships. For the hierarchic model, you must make a decision about which information is more important within the data context. If both vendors and parts are equally important, then complete redundancy with two hierarchies is required as shown in the diagram.

The physical definition of data, or **physical data model**, describes its layout for a particular hardware device. Physical layout is constrained by intended data use, access method, logical model, and storage device. External storage devices for data include magnetic disk, magnetic diskette, optical disk, compact disk, laser disk, digitally applied tape, and magnetic tape, to name a few. The major differences in devices are the number of times a device can be written to [e.g., as in write-once-read-many (WORM) technology], the cost, the amount of data that can be stored, the portability of devices, and the type of retrievals that can be done on data (e.g., magnetic tape requires front-to-back sequential processing versus direct accessibility to any data).

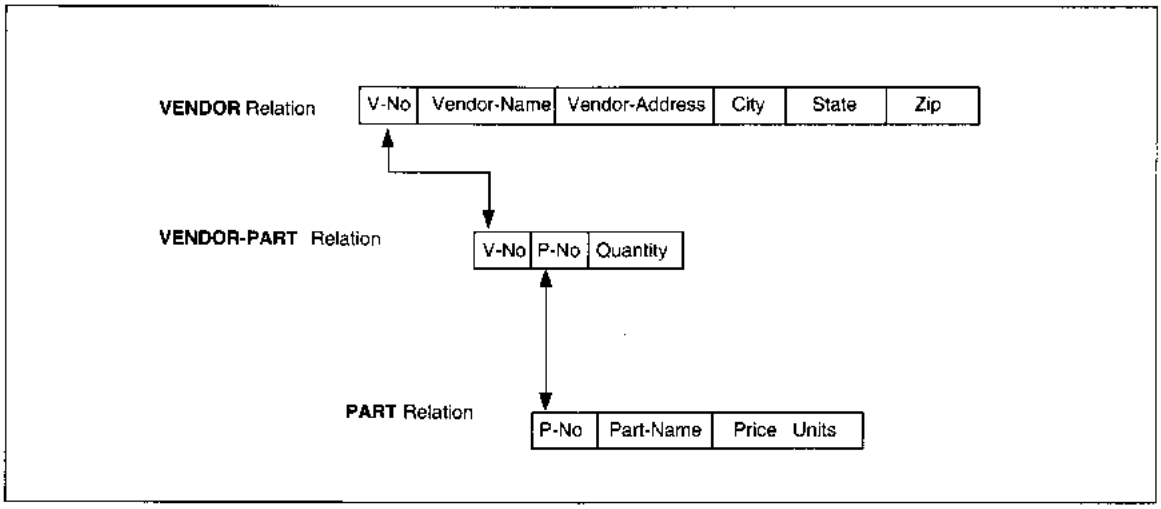


FIGURE 1-4 Third Normal Form Example

VENDOR Relation

V-No	Vendor-Name	Vendor-Address	City	State	Zip
01	ABC Hardware	123 Main St.	Morristown	NJ	07950
03	XYZ Hardware	425 Center St.	Akron	OH	44311
02	QBE Hardware	7290 4th St.	New York	NY	10010

VENDOR-PART Relation

V-No	P-No	Quantity
01	001	750
01	002	2000
02	004	1200
01	004	1000
....		

PART Relation

P-No	Part-Name	Price	Units
001	Screwdrivers	700	Each
002	Nails, #1	125	Gross
004	Nails #3	120	Gross
.			

FIGURE 1-5 Relational Logical Data Model

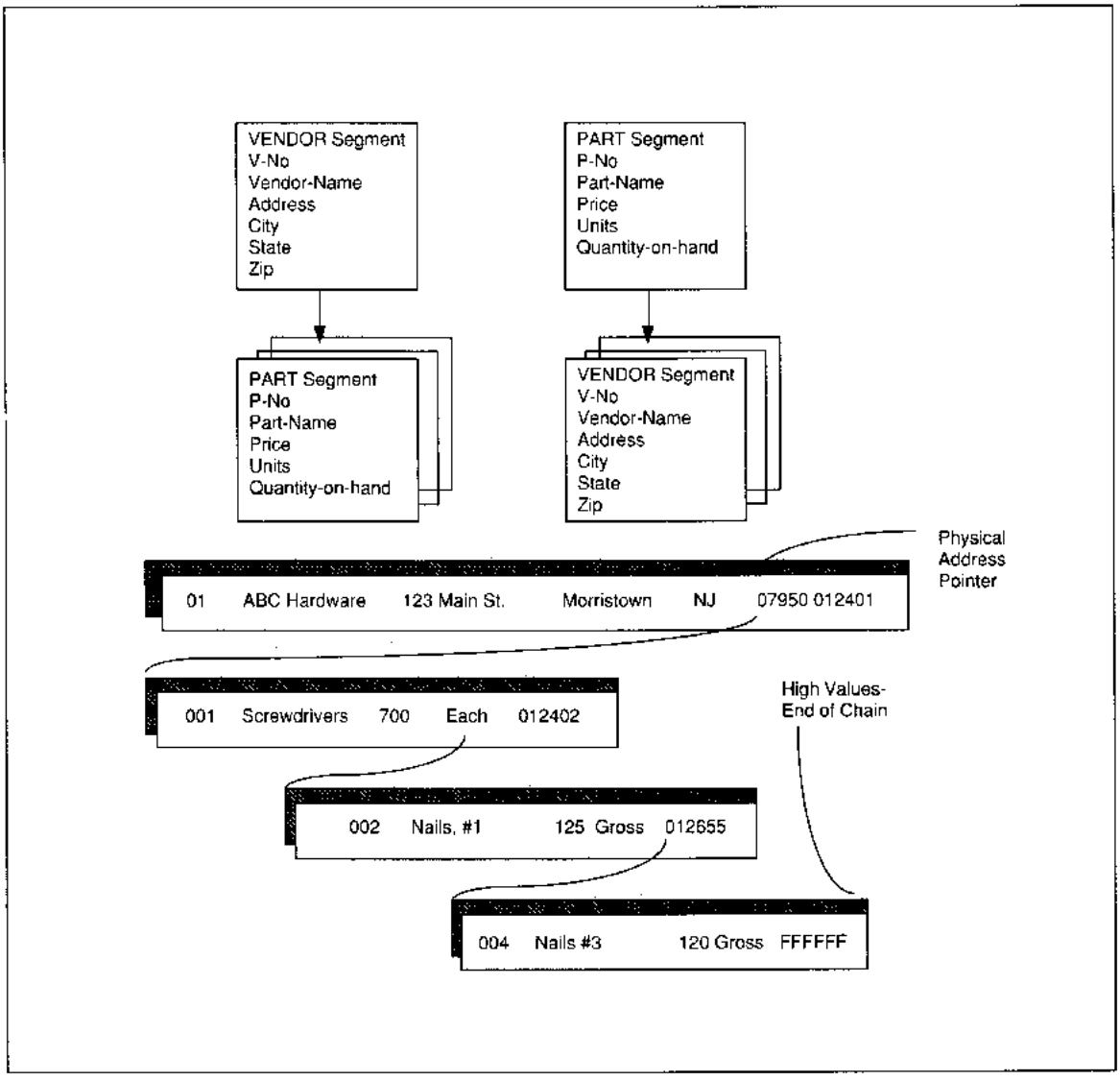


FIGURE 1-6 Hierarchic Logical Data Model

Processes

A **process** is the sequence of instructions or conjunction of events that operate on data. The results of processing include changes to data in a database, identification of data for display at a terminal or printing on paper, generated commands to equipment, generated program commands, or storage of new facts or rules inferred about a situation or entity.

Constraints

Processing is subject to **constraints,** which are limitations on the behavior and/or processing of entities.

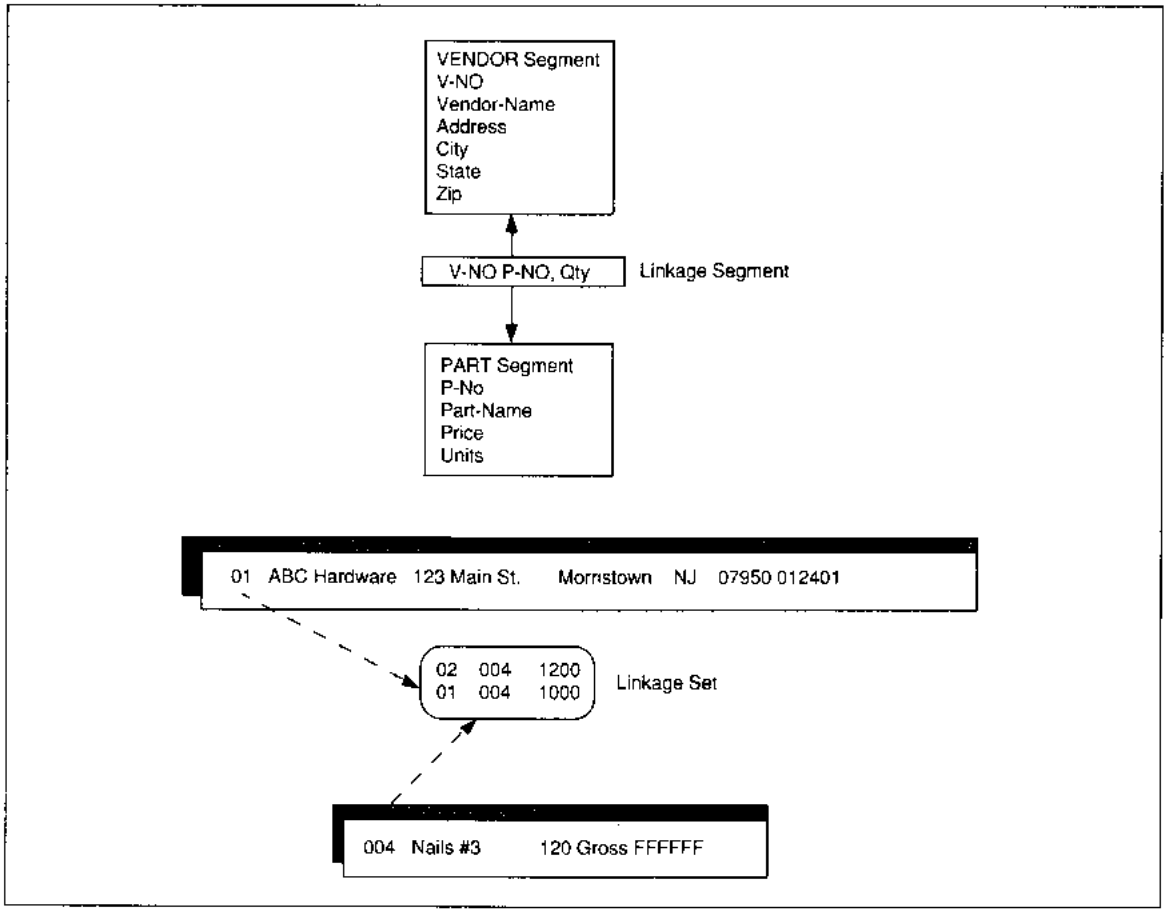


FIGURE 1-7 Network Logical Data Model

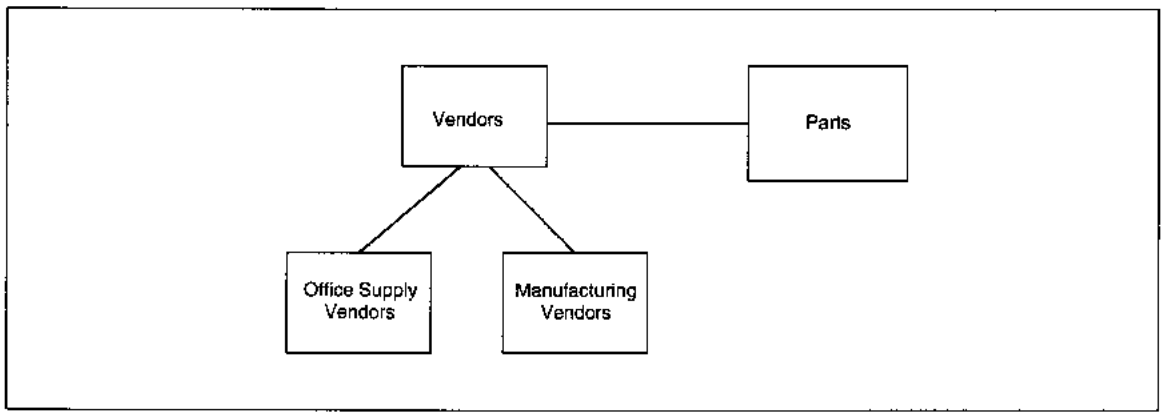


FIGURE 1-8 Object-Oriented Logical Data Model

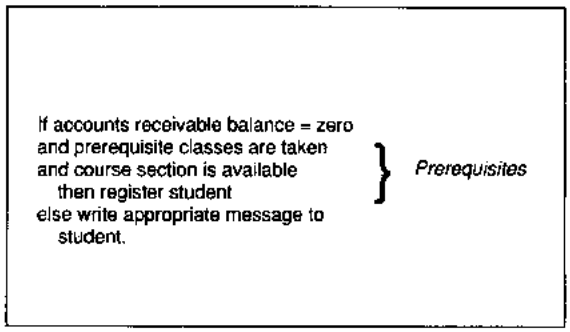


FIGURE 1-9 Prerequisite Constraint Example

Constraint types are prerequisite, postrequisite, time, structure, control, or inferential.

PREREQUISITES. **Prerequisite constraints** are preconditions that must be met for processing to occur. They usually take the form of 'if . . . then . . . else' logic in a program (see Figure 1-9).

POSTREQUISITES. **Postrequisite constraints** are conditions that must be met for the process to complete successfully. They also take the form of 'if . . . then . . . else' logic, but the logic is applied *after* processing is supposedly complete.

TIME. **Time constraints** may relate to one or more of the following:

1. Timing of processing, for instance, all money transfers in New York must be processed by 3 P.M. to meet the NewYork Federal Reserve Bank closing deadline.
2. Time allotted for a process, for instance, time-out of the database when remote site *A*'s expected response is not received within ten seconds.
3. External time requirements, for instance, reports must be delivered to the Controller's office by noon.
4. Synchronous processing, for instance, locations *A* and *B* must both have completed their respective actions successfully for location *C* to perform action *X*.
5. Response time for external interface processing, for instance, the system must respond to the user terminal within two seconds after the enter key is pressed.

STRUCTURE. **Structural constraints** describe the relationships between data, meta-data (knowledge about data), knowledge and meta-knowledge (system generated knowledge) in applications (see Figure 1-10). Customers, for example, might have different processing if they pay by credit or cash. So, there would be a general *class* customer and two *subclasses*, credit-customer and cash-customer. Meta-data about customers includes, for example, the definition of the domain of allowable values for customer identification.

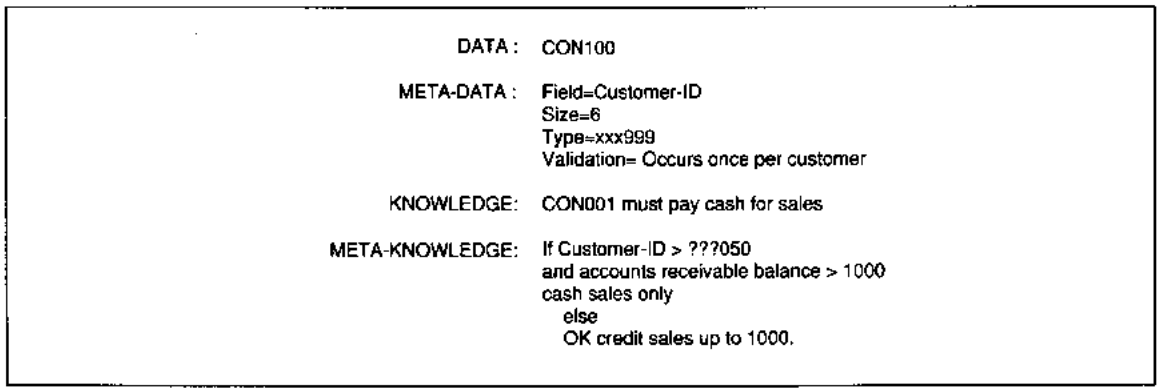


FIGURE 1-10 Structural Constraint Example

Structural constraints determine what type of inputs and outputs may be allowed, how processing is done, and the relationships of processes to each other.

CONTROL. **Control constraints** relate to automated maintenance of data relationships (e.g., the batch total must equal the sum of the transaction amounts).

INFERENCES. The word infer means to conclude by reasoning, or to derive from evidence. **Inferential constraints** are limits on the reasoning ability of the application and its ability to generate new facts from previous facts and relationships. These constraints come in several varieties. First, inferential constraints may relate to the application. For example, you might not want a medical expert system to build itself new knowledge based on new user information unless the "user" is an approved expert who understands what he or she is doing.

Second, inferential constraints may relate to the type of knowledge in the system and limits on that knowledge. For example, CASE tools cannot help you decide what information to actually enter into the system (yet). Rather, you as the user must already know what you want to describe and how to describe it when you use a CASE tool. What CASE *can* do is reason whether the information you entered conforms to its rules for how to represent information.

Third, inferential constraints may relate to the language in which the system is developed. For instance, you might be required to build an expert system in Prolog because that is the only language available. Prolog is a goal-oriented, declarative language with constructs for facts and rules that requires its knowledge (i.e., the data) to be imbedded in the program. Large programs in Prolog are hard to understand and may be ambiguous. Therefore, programmers write smaller, limited reasoning programs. If you have a large, complex knowledge base, you may want to separate the data from the program logic. But the language choice can constrain your ability to do such separation.

Interfaces

There are three types of interfaces: human, manual, and computerized. There are few guidelines in any methodologies for designing any of these interfaces. Each type of interface is discussed briefly in this section, and in more detail later in the text.

HUMAN. **Human interfaces** are the means by which an application communicates to its human users. Human interfaces are arguably the most important of the three types because they are the hardest to design and the most subject to new technologies and fads.

Most often, a human interface is via a video display which might have options for color, size of screen, windows, and so on. Many application developers are tempted to design elaborate screens with the assumption that more is better: more color, more information, and so forth. But a growing body of research combined with graphic design ideas show that this is not the case. Figure 1-11 shows the same information on a well designed screen and on a poorly designed screen. A screen should be organized to enhance readability, to facilitate understanding, and to minimize extraneous information. Few colors, standardized design of top and bottom lines, standardized use of programmable function keys, and easy access to help facilities are the keys to good screen design.

MANUAL. **Manual interfaces** are reports or other human-readable media that show information from the computer. You use manual interfaces whenever you pay electric, telephone, or water bills. Some simple standards for manual interfaces are to mirror screen designs when possible to enhance understanding, to fully identify the interface contents with headers, notes, and footers when needed, and to follow many of the same human interface "rules" for formatting information.

AUTOMATED. An **automated interface** is data that is maintained on computer-readable media for use by another application. Application interfaces tend to be nonstandardized and are defined by the data-sharing organizations. Guidelines for applica-

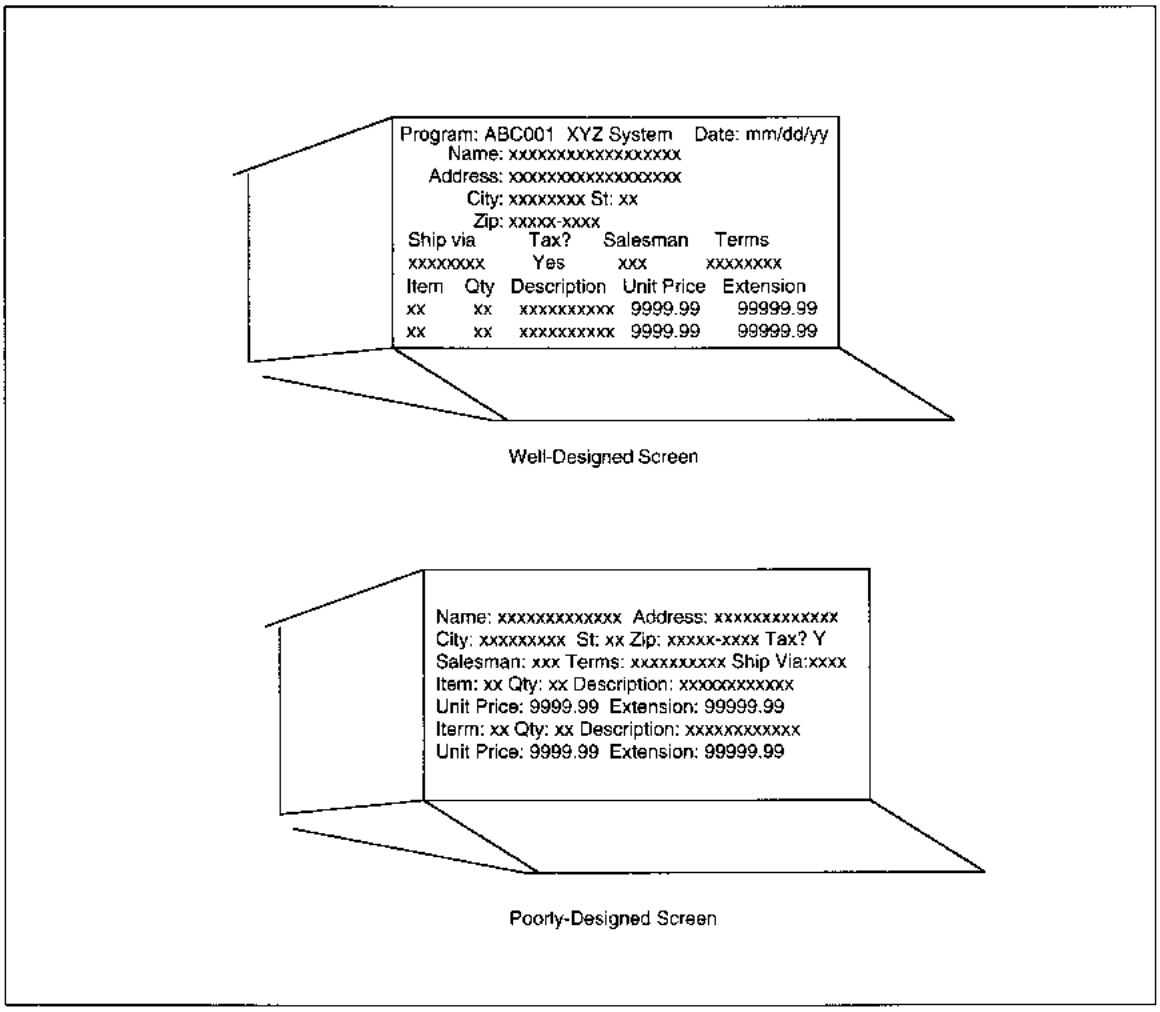


FIGURE 1-11 Good versus Bad Screen Design

tion file interfaces have evolved over the last fifty years to include, for instance, placement of identifying information first and placement of variable length information last. Other interfaces are governed by numerous formal standards, for instance, local area network interface standards are defined by the Institute of Electrical and Electronic Engineers (IEEE) and the open system interface (OSI) standard for inter-computer communication is governed by the International Standards Organization (ISO). Few such standards are currently relevant to an individual business application. Lack of standards, such as for graphics user interfaces (GUIs) slows business acceptance of new innovations. Uncertainty over which 'look' will become the standard, in the case of GUIs, leads to business caution in using new technology.

Application Responsiveness

In this book, application responsiveness is how long it takes the system to act on and respond to user

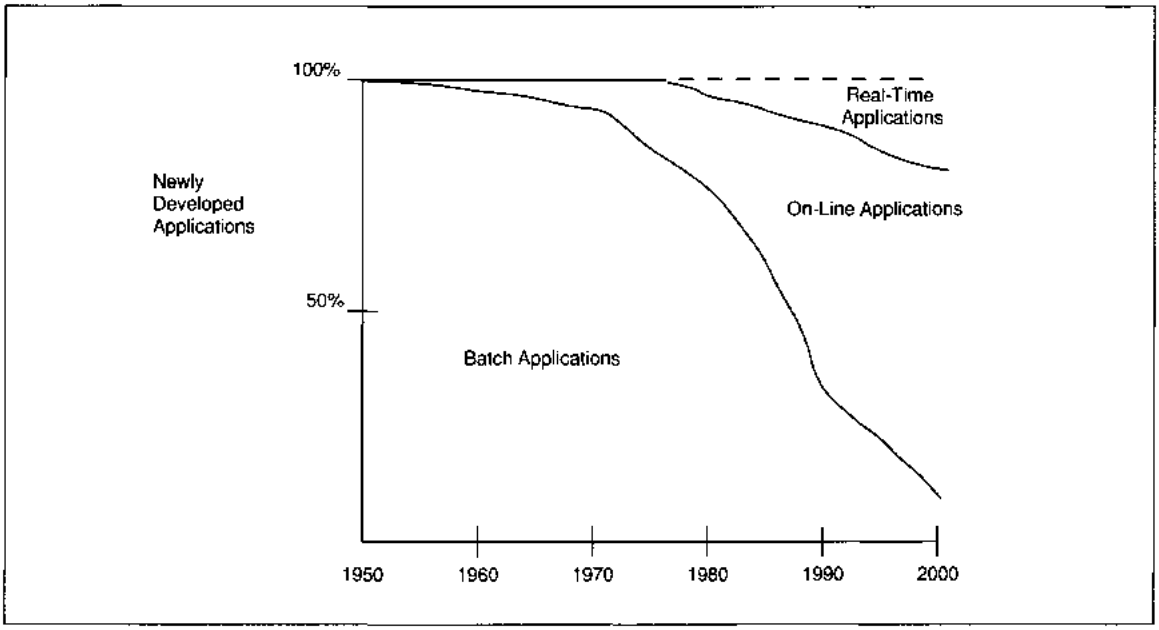


FIGURE 1-12 Application Type Transition

actions. Responsiveness of an application reflects the fundamental design approach as batch-oriented, on-line, or real-time. Each of these approaches is defined in this section. Of course, in the real world, any combination or permutation of these approaches are used in building applications. Most applications designed in the 1990s are on-line with some batch processing. In the 21st century, on-line applications will give way to more real-time applications. Figure 1-12 shows the transition from batch to on-line to real-time processing in the last half of this century. Table 1-1 compares application responsiveness on several categories.

Batch Applications

Batch applications are applications in which transactions are processed in groups. Transactions are gathered over time and stored together. At some predefined time, the batch is closed and processing on the complete batch is done. Transactions are processed in sequence one after the other. A system flow diagram of a typical batch application is shown in Figure 1-13. The *batch of transactions* is edited and applied to a *master file* to create a *new master file* and a printed *log of processing*. In batch applications the requirements relating to the average age and maximum possible age of the master file data determine the timing of processing.[5] In addition to processing transactions, other programs in batch applications use the master file as their major input and process in a specific fixed sequence.

On-Line Applications

On-line applications provide interactive processing with or without immediate file update. **Interactive processing** means there is a two-way dialogue between the user and the application that takes place during the processing. This definition of on-line differs somewhat from the use of on-line terminology in other texts which assume that on-line systems are

5 See Davis, G. and Olson, M., *Management Information Systems: Conceptual Foundations, Structure, and Development*, New York: McGraw-Hill, 1985, for a detailed discussion of batch systems.

TABLE 1-1 Comparison of Application Technologies

Category	Batch Applications	On-Line Applications	Real-Time Applications
Amount of data	Large	Small-Large	Medium
Visual review of inputs	No	Yes	Yes
Ratio of updates to stored data	High	Low-High	High
Inquiry	Batch	On-line	On-line
Reports	Long, formal	Short, informal	Short, informal
Backup/Recover	Copy files to tape	One or more of the following: Copy files to tape transaction log, preimage log, postimage log, mirror image files	One or more of the following: Copy files to tape transaction log, preimage log, postimage log, mirror image files
Cost to build*	Low	Medium	High
Cost to operate*	Low	Medium-High	High
Efficient use of	Computer resources	People time	People time
Difficulty to build*	Simple	Medium	Complex
Speed of processing all transactions	Fast	Slow	Medium
Speed of processing one transaction	Slow	Medium	Fast
Uses DBMS and data communications	May or may not	Probably	Yes
Function integration	Low	Medium	High

*Relative measure

also real-time (see the next section). In this text, on-line processing means that programs may be resident in memory and used sequentially by numerous transactions/events without reloading.

Figure 1-14 shows the difference between an on-line application and a batch application. In an on-line application, small modules perform the function and communicate directly *via data* passed between them. In the batch application, disjoint programs perform the function and indirectly communicate *via permanent changes to file contents* created by one program and interpreted by the next program(s). The on-line programs keep a log of transactions to provide recovery in case of error; this prevents re-entry of data.

On-line programs' dialogue with the user is to ensure entry of syntactically correct data. The error correction dialogue replaces the error portion of the update log. The remainder of the update log to document updates becomes optional and, instead, an acknowledgement of successful processing is displayed to the user.

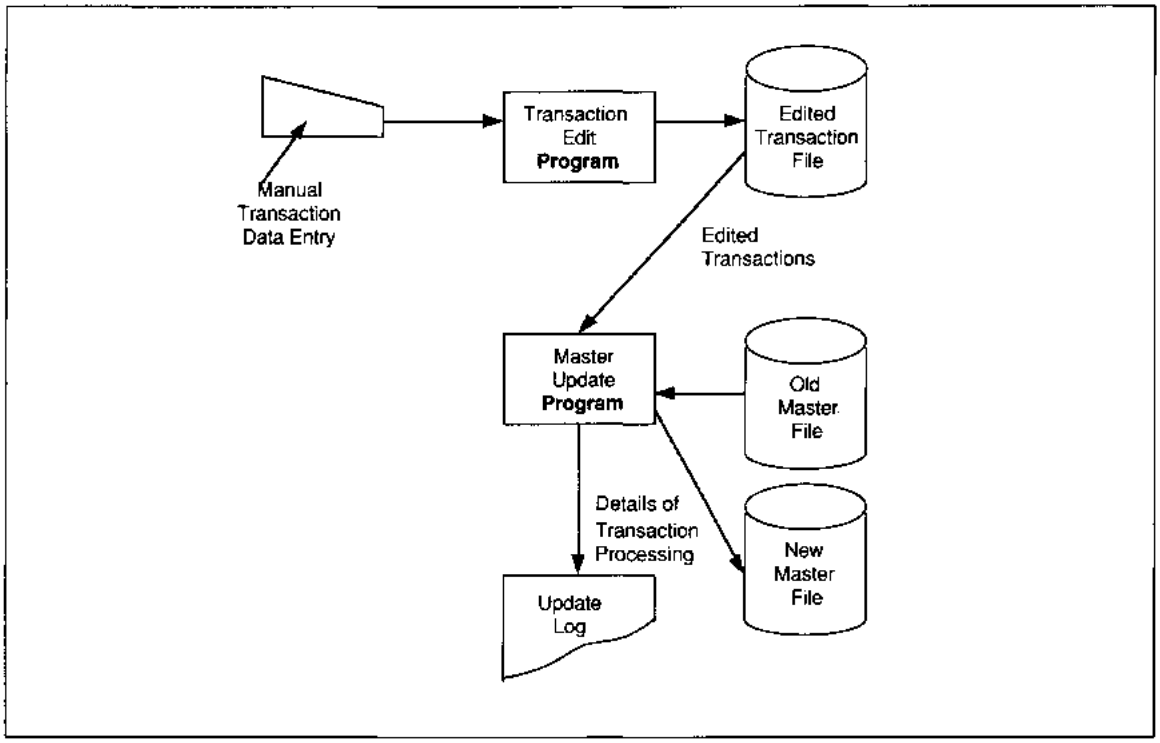


FIGURE 1-13 Batch Application System Flow Diagram

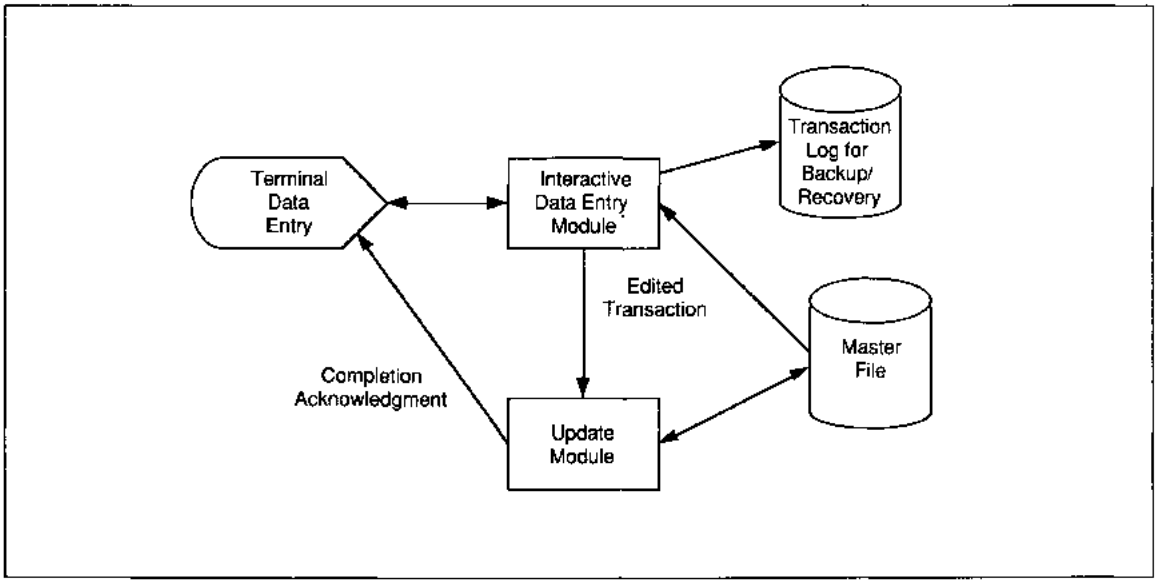


FIGURE 1-14 On-Line Application System Flow Diagram

Real-Time Applications

Real-time applications process transactions and/or events during the actual time that the related physical (real world) process takes place. The results of the computer operation are then available (in real time) to influence or control the physical process (see Figure 1-15). Changes resulting from a real-time process can be refreshed to users viewing prechange data when the change is completed. Real-time programs can process multiple transactions concurrently. In parallel processes, concurrency literally means that many transactions are being worked on at the same time. In sequential processes, concurrency means many transactions are *in process* but only one is actively executing at any one moment.

Database processing is more sophisticated in real-time systems. If an update to a data item takes place, all current users of the item *may* have their screens refreshed with the new data. Examples of real-time applications include automated teller machine (ATM), stock market ticker, and airline reservation processing.

Types of Applications

There are four types of business applications: transaction, data analysis, decision support, and expert applications. Today, all four types are usually online although the application may use any (or all) of the responsiveness types, even on a single application. In addition, a fifth type of application: embedded, is defined briefly to distinguish computer science-software engineering from IS-software engineering.

Transaction-Oriented Applications

Transaction-oriented applications, also known as **transaction processing systems (TPS)**, support the day-to-day operation of a business and include order processing, inventory management, budgeting,

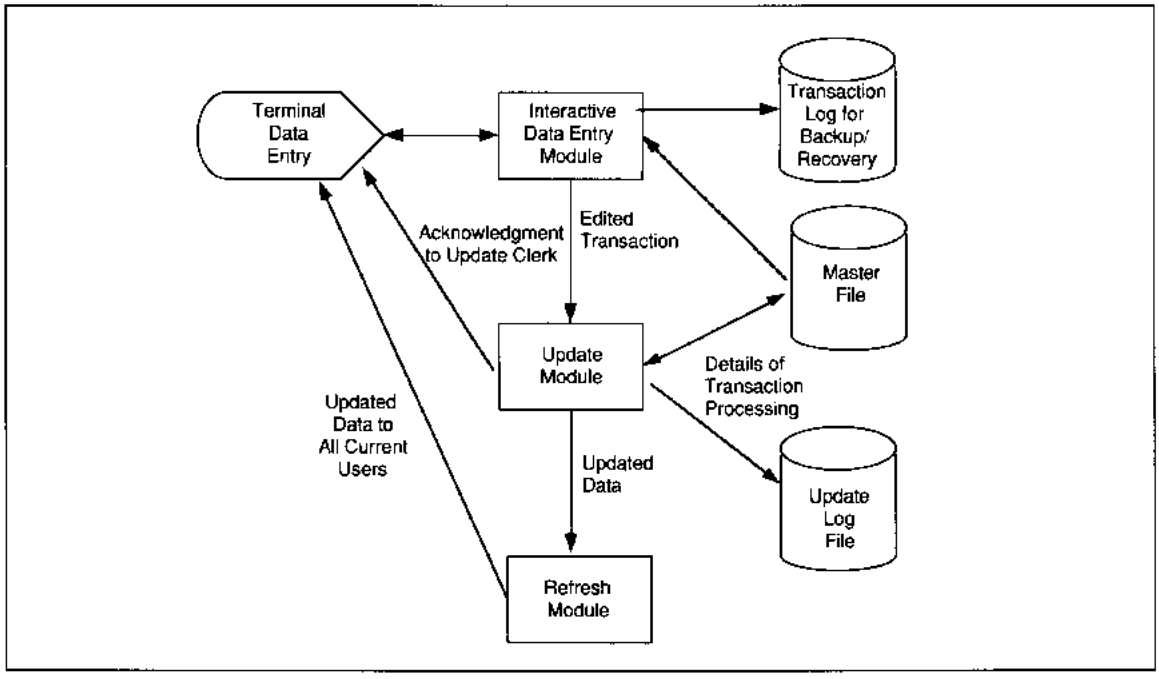


FIGURE 1-15 Real-Time Application System Flow Diagram

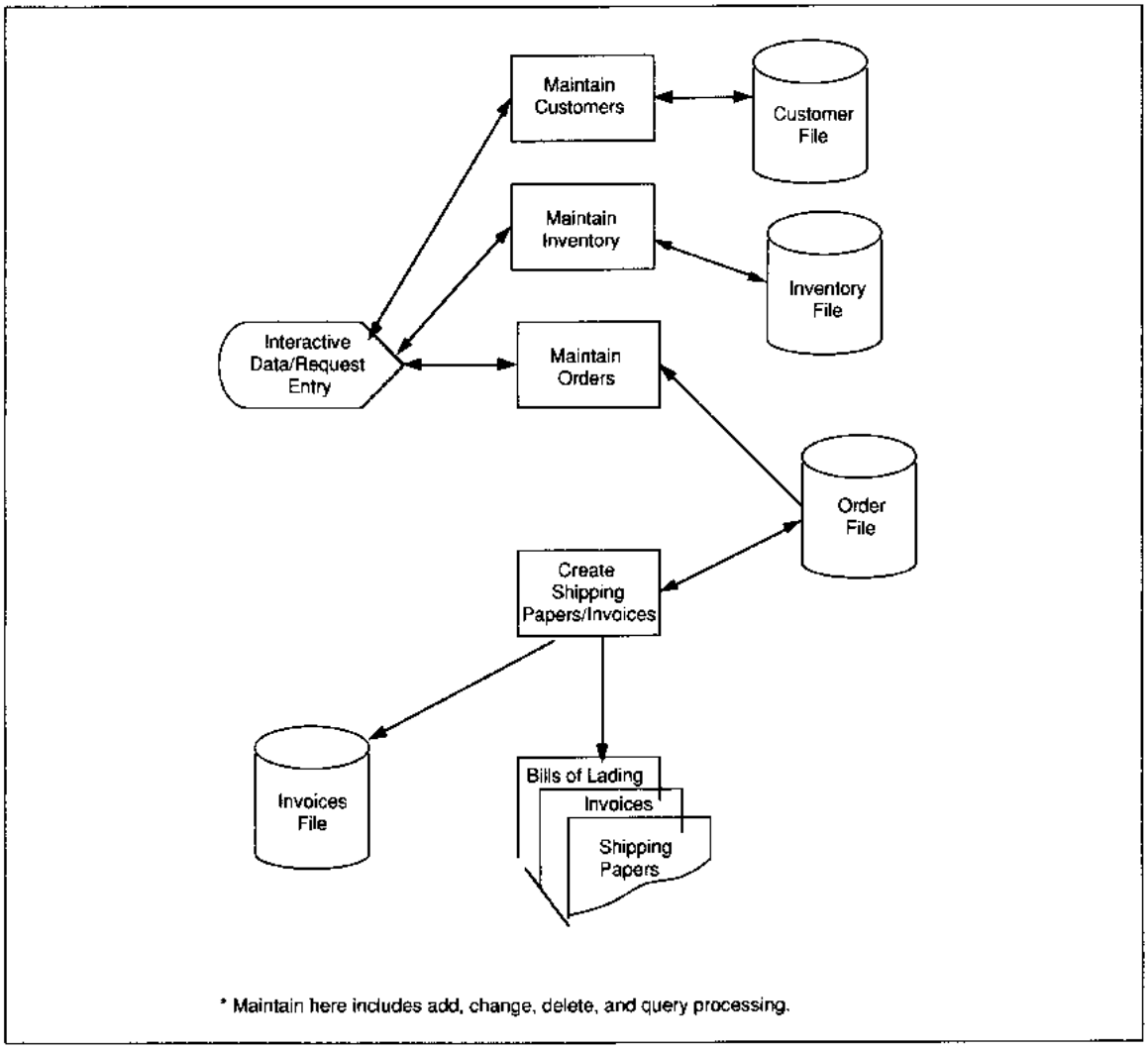


FIGURE 1-16 Order Processing Applications

purchasing, payables, accounting, receivables, payroll, and personnel. They are characterized as applications for which the requirements, the data, and the processing are generally known and well-structured.[6] By known, we mean that the function is repetitious, familiar and unambiguous. By well-structured, we mean that the problem is able to be defined completely and without ambiguity. The requirements are identifiable by a development team.

A transaction application example is order processing (see Figure 1-16). Order processing requires an order file, customer file, and inventory file. The contents of the files differ depending on the level of integration of order processing with accounts receivable, manufacturing, purchasing, and inventory processing. Processing of orders requires add, change,

6 An informative text on transaction processing systems is *On-line Business Computer Applications*, 2nd ed., by Alan Eliaison. Chicago: Science Research Associates, Inc., 1987.

EXAMPLE 1-3

EFFECTIVE INSURANCE COMPANY

In the early 1980s, Effective Insurance realized they were generating 22 feet of paper each month in accounting reports that were sent to about 80 different parts of the organization. Yet, for all this paper, the number of legitimate requests for access to data was mushrooming and had reached about 200.

Rather than try to produce reports for each specific user, the company decided to automate the information and allow users to access their own data to generate their own reports. That way, paper could be reduced and each person would have the data they wanted, formatted the way they wanted it.

The company never anticipated the immense savings in time, money, and, more importantly, the increases in productivity and morale, that this move would produce. By 1989, there were over 2,000 users accessing some or all of the accounting information. Each user had his own terminal and the use of a fourth generation language,* to generate customized information interactively. Reports were created by each user as needed.

*A fourth-generation language is one in which a query language, statistical routines, and data base are integrated for application development by both IS and by non-IS professionals.

delete, and inquiry functions for all files, pricing of items, and creation of shipping papers and invoices. Inquiry functions should allow retrieval of information about orders by date, order number, customer ID, or customer name. The software engineer uses his or her understanding of general order processing to customize the application for a given organization and implementation environment.

Data Analysis Applications

Data analysis applications support problem solving using data that are accessible only in read-only mode. Data analysis applications are also known as **query applications**. A query is some question asked of the data. SQL, the standard query language for database access, poses questions in such a way that the user asks *what* is desired but need not know *how* to get it. The computer software figures out the optimal access and processing methods, and performs the operations it selects. An example of a query asking for the sum of all sales for customers in New York State for the first yearly quarter might look like the following:

```
SELECT CUST_NAME, CUST_ID, AND
SUM(CUST_SALES)
  FROM CUSTOMER
  WHERE CUST_STATE = 'NY' AND
  MONTH IN (1, 2, 3);
```

A language, such as SQL, is a **declarative language**, because you 'declare' *what* to do, not *how* to do it. Declarative languages are relatively easy to learn and use, and are designed for use by noninformation systems professionals.

Queries are one of three varieties:

1. Interactive, one-of-a-kind. These are assumed to be thrown away after use.
2. Stored and named for future modification and re-execution.
3. Stored and named for frequent unchanging execution.

The third type of query frequently replaces reports in transaction applications (see Example 1-3). The data for all query processing must be known in advance and tend to come from transaction applications. Query outputs may use program language for-

matting defaults (as in SQL), or may be formatted for formal visual presentation or fed into other software (e.g., graphical software) for summarizing.

Query applications support an evolving concept called **data warehouse**, a storage scheme based on the notion that most data should be retained on-line for query access. A warehouse stores past versions of major database entries, transaction logs, and historical records.

Decision Support Applications

Decision support applications (DSS) seek to identify and solve problems. The difference between decision support and query applications is that query applications are used by professionals and managers to select and summarize historical data like the example above, while DSSs are used by professionals and managers to perform what-if analysis, identify trends, or perform mathematical/statistical analysis of data to solve unstructured problems. Data for DSSs usually are generated by transaction applications.

Unstructured problems are ones for which not all information is known, and if it is known, the users may not know all of the relationships between data. An example of a structured problem is to answer the question: "What is the cost of a 5% salary increase?" An example of an unstructured problem is "What product mix should we manufacture next year?" The difference between these two kinds of questions is that the **structured problem** requires one query of known data to develop an estimate, while the product mix question requires economic, competitive, historical, and product development information to develop an estimate. Because the information may not all be known, DSS development uses an iterative problem-solving approach, applying mathematical and statistical modeling to the decision process. Corrected and/or supplemental data are fed back into the modeling processes to refine the analysis.

Executive information systems (EIS) are a spin-off from DSS. EIS applications support executive decision making and provide automated environmental scanning capabilities. Top executives deal with future-oriented, partial, inaccurate, and ambiguous information. They scan the economy, industry, and organizational environments to identify and monitor key indicators of business activity that affect their organization. EIS integrate information from external information databases and internal applications to provide an automated scanning and modeling capability. The major difference in EIS from DSS then, is the incompleteness, potential inaccuracy, and ambiguity of the data.

Group decision support systems (GDSS) are a special type of DSS applications. GDSS provide an historical memory of the decision process in support of groups of decision makers who might be geographically dispersed. GDSS focus more on the group interaction processes with little or no data modeling or statistical analyses. Data analysis software in GDSS tend to be less elaborate than DSS software, but may include a spreadsheet and routines to present summaries of participant votes on issues in either numerical or graphical formats. GDSS typically provide such functions as

1. Anonymous recording of ideas
2. Democratic selection of group leaders
3. Progressive rounds of discussion and voting to build group consensus

For all DSS, application development is more formal than query applications, and less formal than transaction applications. The development life cycle tends to be iterative with continuous identification of requirements. DSS software environments are sophisticated and typically include software tools for communications support, statistical modeling, knowledge-base maintenance, and decision process support.

Expert Systems

Expert systems applications (ES) are computer applications that automate the knowledge and reasoning capabilities of one or more experts in a specific domain. ESs analyze characteristics of a situation to give advice, recommend actions, or draw conclusions by following automated reasoning processes. The four major components of an ES are knowledge acquisition subsystem, the knowledge base, the inference engine (or rule base as it is sometimes called), and explanation subsystem. Each of these components are briefly explained here.

EXAMPLE 1-4

MEDICAL ES ETHICAL DILEMMA

A doctor who is not a specialist in rare diseases sees a patient in the emergency room who appears to be in respiratory distress. After a preliminary exam, he consults with an expert system that diagnoses many diseases and recommends a course of treatment. The ES requests all of the symptoms from the doctor who answers the questions to the best of his ability. The ES diagnoses the problem as advanced Legionnaires' disease with a probability of 80%. The ES suggests no other possible diseases. The doctor prescribes the ES's recommended treatment. The patient dies. On investigation, it turns out that the ES contains errors in its rules and that the correct diagnosis, following the exact same set of symptoms, would have led to a different diagnosis with different treatment.

There are ethical issues in every aspect of this problem. Who is responsible for ES accuracy? Is the knowledge engineer who built the ES responsible for ensuring accuracy of information in the system? Or, does his or her responsibility only mean translating the reasoning processes correctly? What is the responsibility of the "expert" who supplies the information in ensuring it is correctly entered into an ES to supply correct reasoning? If a medical ES contains information on thousands of diseases, is it even possible to test it completely? How is consistency of diagnoses checked? What happens when symptoms are entered in different sequences? Is the doctor who uses the ES suggestion ethical? There is no consensus on answers to these questions at present. The lack of consensus highlights the need for discussion of ethical issues in IT applications.

The **knowledge acquisition subsystem** is the means by which the knowledge base is built. In general, the more knowledge, the 'smarter' the system can be. The knowledge acquisition subsystem must provide for initial loading of facts and heuristic rules of thumb, *and* be easy to use in adding knowledge to the knowledge base.

Frequently, we reason without knowing how we arrive at a solution. In fact, reflect how you yourself think when analyzing a problem to develop an application. How do you decide what the processes are? You follow an elaborate, highly internalized process that is difficult to talk about. You are not alone in having this difficulty. *Eliciting the information* about reasoning processes from experts is a major difficulty in building effective ES applications.

The **knowledge base** is the codified automated version of the expert user's knowledge and the rules of thumb (also called heuristics) for applying that knowledge. Designing the knowledge base is as difficult as eliciting the information because no matter how it is designed, it will be limited by the software in which it is implemented. Therefore, special ES programming languages have been designed to allow the most flexibility in defining connections between pieces of information and the way the pieces are used in reasoning.

Just as people reason to develop a most probable outcome to a situation, ESs use reasoning and inference to develop multiple, probable outcomes for a given situation. Several solutions may be generated when there is incomplete information or partial reasoning. Probabilities of accuracy of the solution(s) are frequently developed to assist the human in judging the usefulness of a system-generated outcome. Ethical and moral issues may be more apparent in ESs than the other application types. Example 1-4 describes an ethical dilemma relating to a medical ES.

The last major component of ES is the ability to explain its reasoning to the user. The **explanation subsystem** provides the ability to trace the ES's reasoning. Tracing is important so the user can learn from the experience of using the system, and so he or

she may determine his or her degree of confidence in the ES's results.

These four application types—transaction, query, DSS, and ES—will be referenced throughout the text to tie topics together and to discuss the usefulness of methodologies, languages and approaches to testing, quality assurance, and maintenance for each.

Embedded Systems

Embedded systems are applications that are part of a larger system. For example, a missile guidance application works in conjunction with sensors, explosives, and other equipment within a single missile unit. The application, by itself, is minor; its complexity derives from its analog interfaces, need for complete accuracy, and real-time properties within the missile's limited life span once it is released. Embedded applications development has been the province of computer science educated developers rather than information systems (IS) educated developers.

As business deploys ever more complex equipment in the context of computing environments, the need for embedded systems skills will increase. This implies that IS education must also address real-time, embedded system requirements, and that computer scientists will continue to move into business for application development.

Applications in Business

Applications are most successful when they match the organizations' needs for information. Most information in organizations is generated to allow the managers to control the activities of the organization to reach the company's goals. Goals may be short-term or long-term. Control of activities implies information evaluation and decision making. There are three levels of organizational decision making: operational, managerial, and strategic. Each level has different information needs and, therefore, different application needs.

At the operational level, the organization requires information about the conduct of its business. Decisions deal with daily operations. For instance, the operational level in a retail organization is concerned with sales of products. The main operational level applications would be order processing, inventory control, and accounts receivable. In a manufacturing business, the operational level is concerned with sales and manufacturing. The main operational level applications would be manufacturing planning, manufacturing control, inventory management, order processing, and shipping.

The information at the operational level is current, accurate, detailed, available as generated, and relates to the business of the organization. Operational information is critical to the organization remaining in business. As a critical resource, the data requires careful management and maintenance. The types of applications that support operational level decisions and information are transaction processing applications (see Figure 1-17). Query applications for current operational data are other applications that support operational level decisions.

The information needs for managerial control are mostly internal information, can be detailed or summary, and should be accurate. Decisions made for managerial control concentrate on improving the existing ways of doing business, finding and solving problems, and take a medium-range (e.g., quarter or year) view of the company's business. The types of issues dealt with concern reduction of

- costs by comparing suppliers' prices
- the time to process a single order
- the errors in a process

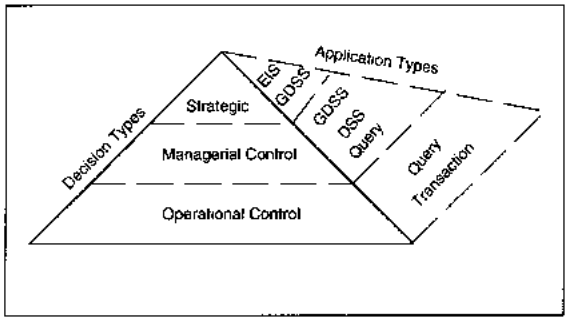


FIGURE 1-17 Application Types and Decision Types

- the number of manual interactions with an order, and so on

The types of applications that support these data needs are data analysis applications, DSS, and GDSS (see Figure 1-17). Each of these application types serves a different role in supporting managerial control decision needs. Data analysis applications can be used to find and solve problems. DSSs can be used to identify trends, analyze critical relationships, or compare different work processes for possible improvements. GDSSs facilitate meetings of people with different motivations and organizational goals, providing a means to reach consensus with a frank discussion of the issues.

At the strategic level, the types of decisions take a broad view of the business and ask, for instance, what businesses should we be in? What products should we produce? How can we improve market share? These questions require external information from many sources to reach a decision. The information is ambiguous, that is, able to be interpreted in many different ways. Because the information is future-oriented, it is likely to be incomplete and only able to be digested at a summary level.

The types of applications that support incomplete, ambiguous, external information needs best are executive information systems (EIS) (see Figure 1-17). EISs are specifically designed to accommodate incomplete, ambiguous information. GDSSs also might be used at the executive level to facilitate discussion of alternative courses for the organization.

PROJECT LIFE CYCLES

There are several different ways to divide the work that takes place in the development of an application. The work breakdown in general comprises the project's life cycle. If you asked five SEs to describe the life cycle of a typical computer application, you would get five overlapping but different answers. Life cycles discussed here are the most common ones: sequential, iterative, and learn-as-you-go.[7]

Sequential Project Life Cycle

You should remember from systems analysis that a **sequential project life cycle (SPLC)** starts when a software product is conceived and ends when the product is no longer in use. Phases in a SPLC include

- initiation
- problem definition
- feasibility
- requirements analysis
- conceptual design
- design
- code/unit test
- testing
- installation and checkout
- operations and maintenance
- retirement

These SPLC phases are more appropriate to business than to military/government applications because, in the government, the first four phases (initiation, definition, feasibility, and functional requirements definition) are usually completed by a different organization than that of the implementers. Government projects are subject to congressional review, approval, and budgeting. So, a government project requiring congressional appropriation is usually defined as beginning at the conceptual design phase and ending with deployment of the system with operational status according to Department of Defense standard #2167a [DOD, 1985]. In contrast, business IS are typically initiated by a user department requesting that a system be built by an MIS department. The need for an IS is typically motivated by some business situation: a change in the method of business, in the legal environment, in the staffing/support environment, or in a strategic goal such as improving market competitiveness.

We call these SPLC phases a 'Waterfall' approach to applications because the output of each phase feeds into the next phase, while phases are modified via feedback produced during the verification and validation processes[8] (see Figure 1-18).

7 Future developments in life cycles are discussed in Chapter 18.

8 Boehm, Barry W., *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

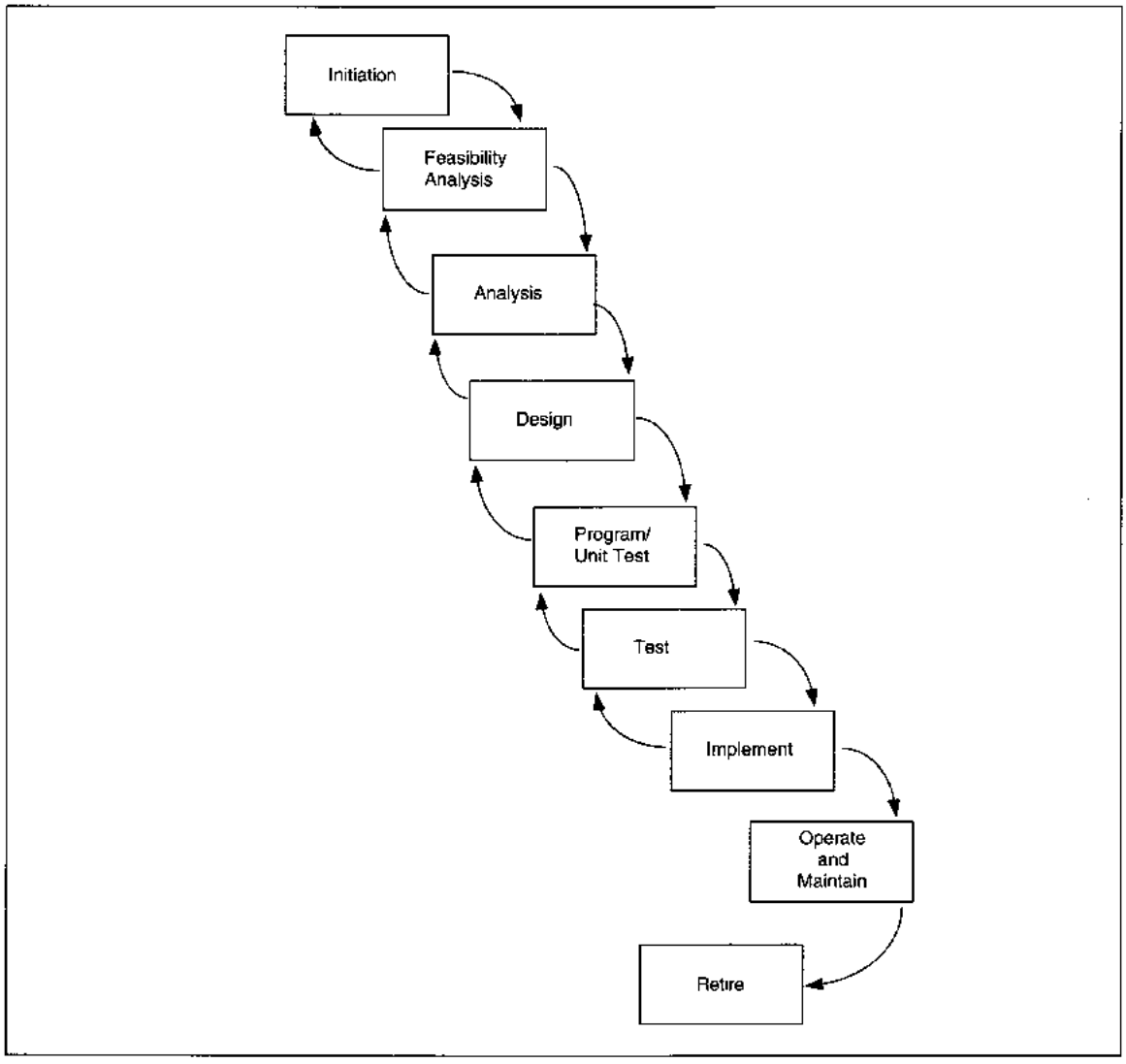


FIGURE 1-18 Sequential Project Life-Cycle Model

Phases in the waterfall definition are defined as discrete even though, in practice, the information is obtained in a nonlinear manner and the phase beginnings and endings are difficult to distinguish. To identify discrete beginnings and endings, most companies use the completion of the major product (i.e., program or document) produced during each phase as signaling the phase end. So, completion of a feasibility report, for instance, identifies the end of the feasibility analysis phase. In the following subsections, each phase of the project life cycle (SPLC) is defined,[9] with the main activities and documents identified.

9 This definition is adapted from work conducted during The Assessment and Development of Software Engineering Tools project sponsored by the U.S. Army Institute for Research in Management Information, Communications, and Computer Sciences (AIRMICS), contract DAKF11-89-C-0014.

SPLC Phases

INITIATION. Project **initiation** is the period of time during which the need for an application is identified and the problem is sufficiently defined to assemble a team to begin problem evaluation. The people and organizations affected by the application, that is, the **stakeholders**, are identified. Participants from each stakeholder organization for the development team are solicited. The outcome of initiation is a memo or formal document requesting automation support and defining the problem and participants.

FEASIBILITY. **Feasibility** is the analysis of risks, costs and benefits relating to economics, technology, and user organizations. The problem to be automated is analyzed in sufficient detail to ensure that all aspects of feasibility are evaluated.

Economic feasibility analysis elaborates costs of special hardware, software, personnel, office space, and so forth for each implementation alternative.

In **technical feasibility** analysis, alternatives for hardware, software, and general design approach are determined to be available, appropriate, and functional. The benefits and risks of alternatives are identified.

Organizational feasibility is an analysis of both the developing and using organizations' readiness for the application. Particular emphasis is placed on skills and training needed in both groups to ensure successful development and use of the application. The decision whether or not to use consultants and the type of role they would play during development is made during organizational feasibility analysis. Organizational decisions include effectiveness of the organization structure and definition of roles of individual jobs in the organization as they will be with the new application.

The feasibility report summarizes

- the problem
- the economic, technical and organizational feasibility
- risks and contingency plans related to the application
- preferred concept for the software product and an explanation of its superiority to alternative concepts
- training needs and tentative schedules
- estimates of project staffing by phase and level of expertise

After feasibility is established, the **Software Development Life Cycle (SDLC)**, a subcycle of the SPLC, begins. This subcycle typically includes phases for analysis, conceptual design, design, implementation, testing, and installation and checkout. SDLC end is signaled by delivery of an operational application.

ANALYSIS. The **analysis** phase has many synonyms: Functional Analysis, Requirements Definition, and Software Requirements Analysis. All of these names represent the time during which the business requirements for a software product are defined and documented. Analysis activities define

1. Functional requirements—"what" the system is supposed to do. The format of the functional requirements definitions depends on the methodology followed during the analysis phase.
2. Performance requirements—terminal, message, or network response time, input/output volumes, process timing requirements (e.g., reports must be available by 10 A.M.).
3. Interface(s) requirements—what data come from and go to other using applications and organizations. The definition includes timing, media, and format of exchanged data.
4. Design requirements—information learned during analysis that may impact design activities. Examples of design requirements are data storage, hardware, testing constraints, conversion requirements, and human-machine interaction requirements (e.g., the application must use pull-down menus).
5. Development standards—the form, format, timing, and general contents of documentation to be produced during the development. Development standards include rules about allowable graphical representations,

documentation, tools, techniques, and aids such as computer-aided software engineering (CASE) tools, or project management scheduling software. Format information includes the content of a data dictionary/repository for design objects, project report contents, and other standards to be followed by the project team when reporting project accomplishments, problems, status and design.

6. The plan for application development is refined.

Analysis documentation summarizes the current method of work, details the proposed system, and how it meets the needs of the required functions. Requirements from the work activities are described in graphics, text, tables, structured English, or some other representation form prescribed by the methodology being used.

CONCEPTUAL DESIGN. Once the proposed logical system is understood and agreed to by the user, conceptual design begins. Other names for conceptual design activity include preliminary design, logical design, external design, or software requirements specifications. The major activity of **conceptual design** is the detailed functional definition of all external elements of the application, including screens, reports, data entry messages, and/or forms. Both contents and layout are included at this level. In addition, the logical data model is transformed into a logical database schema and user views. If distribution or decentralization of the database is anticipated, the analysis and decision are made during conceptual design. The outputs of conceptual design include the detailed definition of the external items described above, plus the normalized and optimized logical database schema.

Not all organizations treat conceptual design separately. Outputs of conceptual design may be in a conceptual design document or might be part of the functional requirements document developed during analysis. Depending on the project manager's personal taste and experience, the conceptual design might be partially completed during logical design and fully completed during physical design. *In this text, the two phases, design and conceptual design, are treated as one.*

DESIGN. **Design** maps "what" the system is supposed to do into "how" the system will do it in a particular hardware/software configuration.[10] The other terms used to describe design activities include detailed design, physical design, internal design, and/or product design.

During the design phase, the software engineering team creates, documents, and verifies:

1. Software architecture—identifies and defines programs, modules, functions, rules, objects, and their relationships. The exact nature of the software architecture depends on the methodology followed during the design phase.
2. Software components and modules—defines detailed contents and functions of software components, including, but not limited to, inputs, outputs, screens, reports, data, files, constraints, and processes.
3. Interfaces—details contents, timing, responsibilities, and design of data exchanged with other applications or organizations.
4. Testing—defines the strategy, responsibilities, and timing for each type of testing to be performed.
5. Data—physically maps "what" to "how" for data. In database terms, this is the definition of the physical layout of data on the devices used, and of the requirements, timing, and responsibility for distribution, replication, and/or duplication of data.

SUBSYSTEM/PROGRAM DESIGN. **Subsystem** and/or **program designs** are sometimes treated as subphases of the design phase. Whether they are separate phases or not, the software engineering team creates, documents, and verifies the following:

1. Application control structure—defines how each program/module is activated and where it returns upon completion.

10 Anyone who has designed a system will tell you that you cannot perform the conceptual design without some knowledge and attention to the implementation environment. So, the "what" and "how" distinctions are generally, but not completely, accurate when described as discrete activities.

2. Data structure and physical implementation scheme—defines physical data layouts with device mapping and data access methods to be used. In a database environment, this activity may include definition of a centralized library of data definitions, calling routines, and buffer definitions for use with a particular DBMS.
3. Sizing—defines any programs and buffers which are expected to be memory-resident for on-line and/or real-time processes.
4. Key algorithms—specifies mathematically correct notation to allow independent verification of formula accuracy.
5. Program component (routine with approximately 100 source procedure instructions)—identifies, names, and lists assumptions of program component design and usage. Assumptions include expectations of, for instance, resident routines and/or data, other routines/modules to be called in the course of processing this module, size of queues, buffers, and so on required for processing.

CODE AND UNIT TEST. During **coding**, the low-level program elements of the software product are created from design documentation and debugged. **Unit testing** is the verification that the program does what it is supposed to do and nothing more. In systems using reusable code, the code is customized for the current application, and checked to ensure that it works accurately in the current environment.

TEST. During **testing**—sometimes called Computer Software Component (CSC) Integration and Testing[11]—the components of a software product are evaluated for correctness of integrated processing. Quality assurance testing may be conducted in the testing phase or may be treated as a separate activity. During **quality assurance** tests, the software product (i.e., software or documentation) is evaluated by a nonmember of the project team to determine whether or not the analysis requirements are satisfied.

IMPLEMENTATION. Also called Installation and Checkout, **implementation** is that period of time during which a software product is integrated into its operational environment and is phased into production use. Implementation includes the completion of data conversion, installation, and training.

At this point in the project life cycle, the software development cycle ends, and the maintenance phase begins. Maintenance and operations continue until the project is retired.

OPERATIONS AND MAINTENANCE. Operations and maintenance is the period in the software life cycle during which a software product is employed in its operational environment, monitored for satisfactory performance, and modified as necessary. Three types of maintenance[12] are

1. **Perfective**—to improve the performance of the application (e.g., make all table indexes binary to minimize translations, change an algorithm to make the software run faster, and so on.)
2. **Corrective**—to remove software defects (i.e., to fix *bugs*)
3. **Adaptive**—to incorporate any changes in the business or related laws in the system (e.g., changes for new IRS rules)

Each type of maintenance requires a mini-analysis and mini-design to determine social, technical, and functional aspects of the change. The current operational versions of software and documentation must be managed to allow identification of errors and to ensure that the correct copy of software is run. One aspect of change management specifically addresses *configuration management* of application programs in support of maintenance activities.

RETIREMENT. Retirement is the period of time in the software life cycle during which support for a software product is terminated. Usually, the functions performed by the product are transferred to a successor system. Another name for this activity is phaseout.

11 This is a term used by DOD standard #2167a, 1985.

12 A detailed discussion of maintenance topics is presented in Lientz and Swanson, 1980.

UNIVERSAL ACTIVITIES. There are two universal activities which are performed during each life-cycle phase: verification and validation, and configuration management.

An integral part of *each life-cycle phase* is the verification and validation that the phase products satisfy their objectives. **Verification** establishes the correctness of correspondence between a software product and its specification. **Validation** establishes the fitness or quality of a software product for its operational purpose.

For instance, the individual code module specifications from design are verified to ensure that they contain accurate and complete information about the functions they perform. The modules are validated against the analysis phase specification to ensure that all required functions have corresponding designs that accurately reflect the requirements.

Configuration management refers to the management of change after an application is operational. A designated *project librarian* maintains the official version of each product. The project librarian is able at any time to provide a definitive version (or *baseline*) of a document or software module. These baselines allow the project manager to control both the software maintenance process and the software products.

History

The sequential life cycle was originally developed and documented in the 1960s to provide defense contractors a life-cycle documentation standard for Department of Defense (DOD) projects. The current DOD Standard #2167a lists all activities and details all documentation required for software development as fulfillment of military contracts. As industry recognized that their own application development projects were out of control, over budget, and unsatisfactory when complete, they modified the standard to eliminate defense/aerospace terminology and replace it with industry specific terminology. Organizations modified the standard to incorporate elements of methodologies, such as structured development, data flow diagrams, and walk-throughs, that were becoming known at the same time. In the late 1960s and early 1970s the waterfall and 2167 documentation standard were used throughout most Fortune 500 companies as cast-in-concrete requirements for building and documenting systems.

Problems

As nonnegotiable documentation requirements, projects frequently produced thousands of pages of documentation that no one except the authors ever read. Information about applications was rarely in any one person's head and communication overhead became a major problem to completing systems successfully. User/management approval to continue with each phase was not based on their knowledge of what the system would do, but on some other criteria. Published studies showed that the typical written application requirements document contained, on average, one-half to one error per page. The conclusion that paper prose is not a good medium for conveying the complex variety of application requirements led to the development of more graphical representation forms.

Eventually, IS managers realized that the waterfall, when applied too stringently, not only did *not* solve the problems of bad systems, it contributed to a new generation of overdocumented bad systems. The result has been a scaling back on required documentation. Standards have become 'guidelines' for experienced project managers to consider and to provide new project managers with review lists of activities whose relevance they should consider. Each project team customizes the documentation and development activities in addition to the tools and techniques they use.

Even with relaxation of required documentation, a sequential life cycle does not recognize the iterative, nonlinear nature of application development, and cannot easily accommodate overlap of phases. Many organizations now use a variant of the waterfall by performing the activities in an overlapped manner, sometimes called the 'pipeline' approach. Finally, the waterfall approach does not recognize that the level of detail necessary to adequately document application functions is significantly different with the use of automated tools, use of diagrams (e.g., DFDs) to replace text, and use of high level, fourth-generation languages (e.g., SQL).

Current Use

The sequential life cycle is still used but rarely in full detail, and mostly for transaction applications. The sequential life cycle and its terminology will be around for many decades to come, but two divergent trends will occur. On the one hand, demarcation of phases will be more relaxed. Three trends leading to phase relaxation are:

- increasing maturity of computer-aided software engineering (CASE) tools
- increasing use of high-level languages
- availability of reusable electronically stored application information

On the other hand, further alteration and customization to accommodate the need for more detail in systems using new, more complex, or otherwise novel hardware/software components will also take place. It is from these novel, groundbreaking applications that our industry frequently develops new techniques to better communicate application characteristics.

Iterative Project Life Cycle

Iterative PLC Description

An **iterative project life cycle** is a cyclic repetition of analysis and design events. Iterative PLC is sometimes called **prototyping** or a spiral approach to development.

Prototyping is the development of a system or system component in a short period of time without formal written specifications. Originally thought of as helpful for proving the usefulness of new technologies, prototyping caught on in the early 1970s as a way to circumvent the overload of documentation from the sequential life cycle. Frequently, prototyping was wrongfully used and led to bad systems. But, as experience with prototyping has grown, there are three specific uses for which prototyping can be very beneficial:

1. Complete iterative development of an application when requirements are not well-understood, e.g., DSS
2. Proof of utility, availability, or appropriateness for technology, software, or hardware
3. Rapid development of part of the system to ease a critical work situation for users, e.g., order entry without edit/validation to ease paper backlog

Some authors describe a completely different life cycle for prototyped applications. The notion that the lifecycle is completely different is not entirely correct. The life cycle depends on the nature of the prototype. If a complete application is built, then the model of the life cycle mirrors that of the waterfall with iteration between analysis-design-programming-testing-implementation as requirements become known (see Figure 1-19). The difference is the level of detail to which analysis and design are performed. Requirements of iteratively-developed applications are generally not well known or understood. They might be ambiguous or incomplete for some time. The prototype provides a base from which users and developers together discover the requirements for the application.

One use of prototyping tests proof of utility, availability, or appropriateness of the hardware, software, or design concept. The prototype development process is a subphase of development that may parallel either feasibility, analysis, or design. There is no significant testing of a 'proof' prototype because it is being used to verify that an activity can be automated in a certain way, or that hardware (or software) can be used as planned. An example of a proof-of-concept prototype is shown in Figure 1-20 as taking place at the same time as the feasibility study. By the end of feasibility analysis, the usefulness of the prototype is decided, and the feasibility report recommends that the tested product (or idea) be abandoned or used.

A third type of prototype is a partial application developed as a stopgap measure for a particular problem until the complete system is available. A partial prototype might be built with its complete life cycle paralleling one phase of the development life cycle as shown in Figure 1-21. The phases of the prototype development cycle mirror those of a normal development life cycle; they differ in that only a small portion of the entire application is developed. These prototypes can omit processing details. For instance, an on-line data entry might not fully validate data. Feedback to the design team would detail

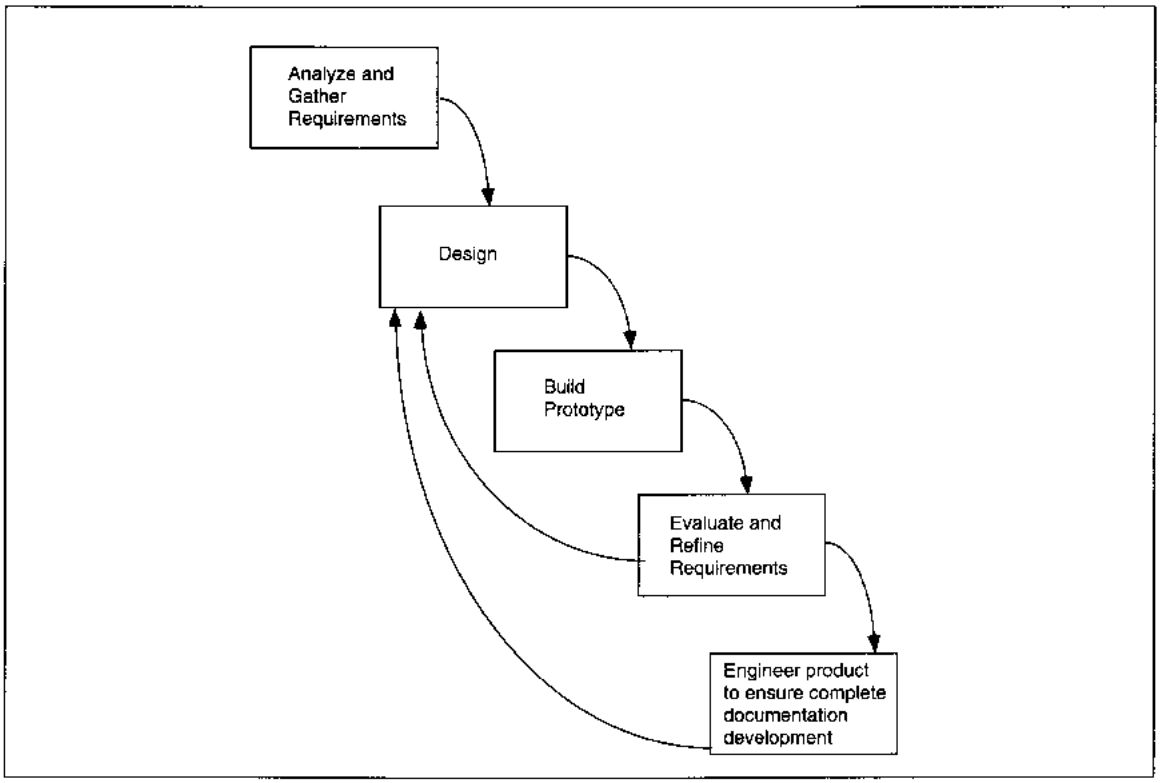


FIGURE 1-19 Full System Prototype Life Cycle

what is and is not in the prototype so that its design and development are completed during the regular application development.

Problems

There are two major problems with prototyping: misuse to circumvent proper analysis and design, and never completing prototypes as proper applications. Prototyping has been used as one way to circumvent rigidities in the sequential life cycle when it is treated as a set of nonnegotiable activities. In this misuse, some authors refer to 'quick analysis' and 'quick design' as if less work is done during those phases. In fact, if done properly, the activities and work are identical to those done in the life cycle, and the effort normally placed on documentation is diverted to building software.

The other major problem with development of a prototype is that the system might never get formalized. Details of processing, for instance, data validation and audit requirements, might be forgotten in the push to get a working prototype into production. While this problem is easily solved, it requires user and management commitment to a completed project. The problems with ensuring this commitment are political, not technical.

Current Use

Although still misused in the development of undocumented, incomplete applications, prototyping for the above intended purposes is also alive and healthy. All forms of query and DSS applications are candidates for iterative life cycles. Some languages (such as Focus, Rbase, Oracle) have easy to learn, short, very high-level programming languages that are naturally amenable to prototyping. A database can be defined, populated with data, and queried in

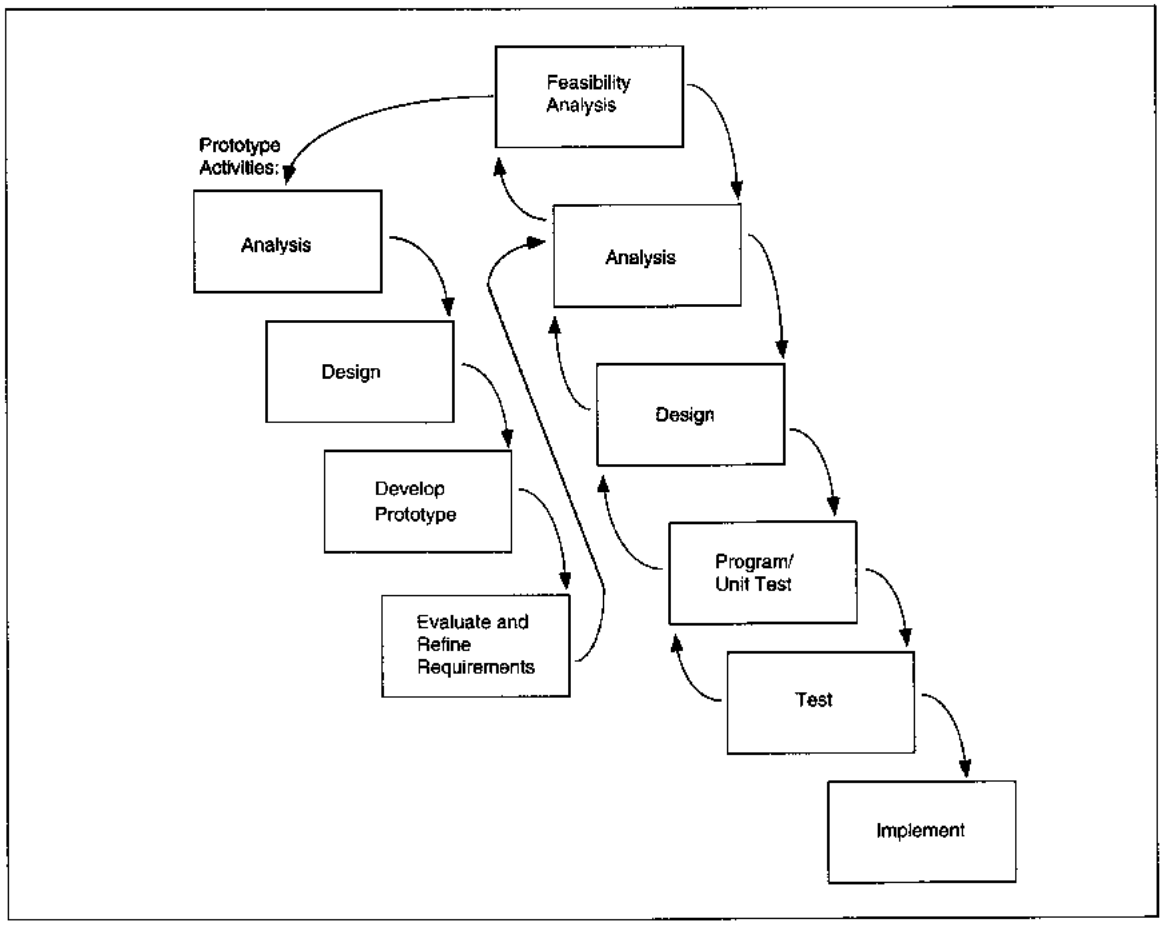


FIGURE 1-20 Proof of Concept Prototype Application Development Activities

under an hour to show capabilities of the languages or discuss requirements of a system. This kind of prototyping builds morale in the IS staff and confidence in the users, and is a great selling tool for in-house application development.

Future Use

Prototyping is appropriate to validate designs, to prove use of new hardware and/or software, or to quickly assist users while building a larger application. For these uses, prototyping is expected to be employed with increasing use of high-level languages to facilitate prototype development. Though there are few automated prototyping tools that also interface to CASE for full application definition, more such integrated tools are becoming available.

Learn-as-You-Go Project Life Cycle

Learn-as-You-Go PLC Description

With all the good news about developments in life cycles, there is a disturbing statistic that about 75% of all companies in the United States do not use any life cycle and/or methodology to guide their development work.[13] The title **learn-as-you-go** could equally well be called trial-and-error, or individual problem solving. The life cycle for the no-life cycle

13 Necco, Charles R., Carl L. Gordon, and Nancy W. Tsai, "Systems analysis and design: Current practices," *MIS Quarterly*, December 1987, pp. 461–476.

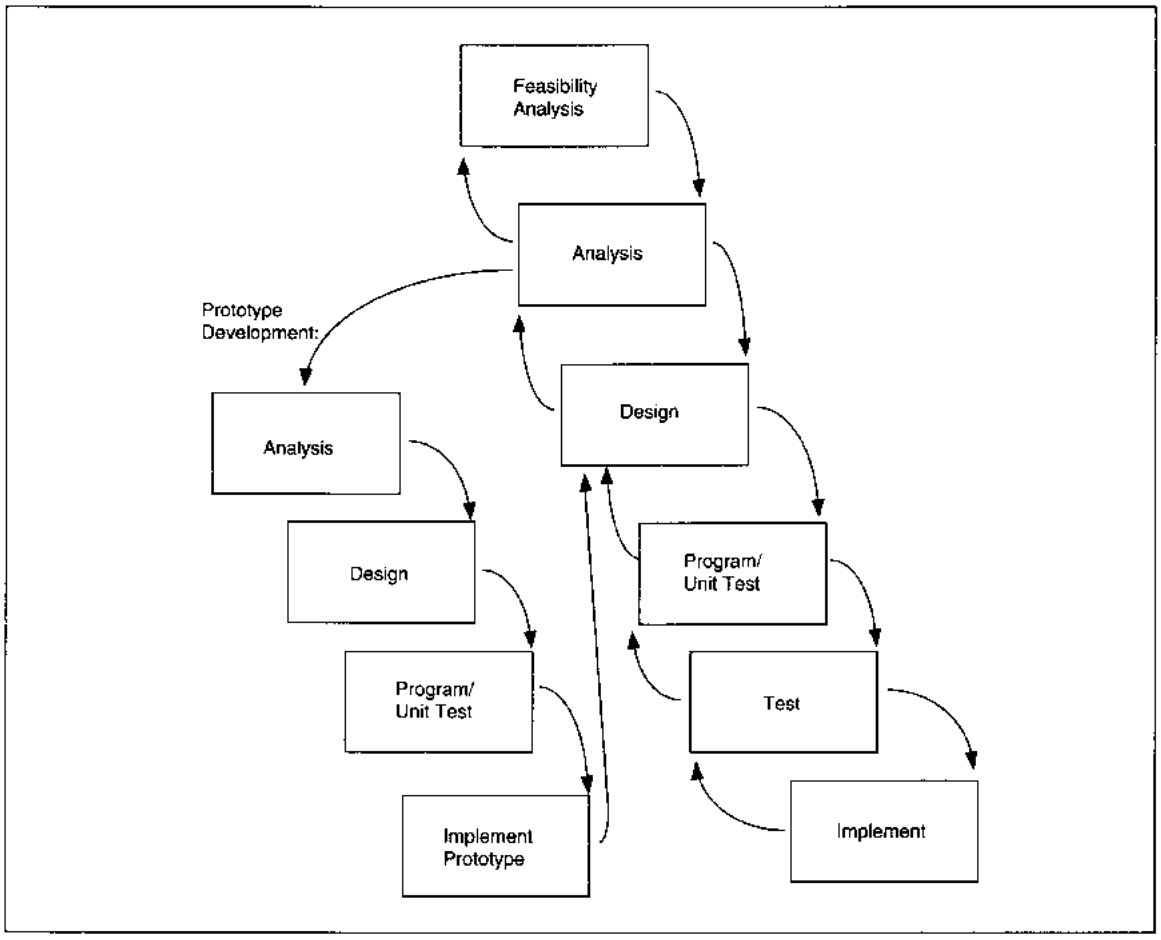


FIGURE 1-21 Partial System Prototype

approach is shown in Figure 1-22, which shows a generic life cycle. The problem is defined. The SE develops the application, which enters operation and maintenance. This approach is not suited to group work, so projects are limited to one person developing small applications. There are two different types of development groups that are in this category: developers of truly unique applications, and developers who do not want too much control or structure in their work.

The first developer view that the problem is unique and cannot easily be molded into a formal life cycle because of its nature is appropriate to applications using emerging technologies and techniques, such as expert systems and artificial intelligence. There is no life cycle that describes building of expert systems, although with a feedback loop between maintenance and definition to indicate iteration, Figure 1-22 is appropriate to these systems. There is no methodology of knowledge engineering; rather, there are several techniques that one might use depending on the nature of the expertise, the personality of the expert, and the complexity of the problem domain. This life-cycle approach is appropriate for such emerging application domains as long as it is a disciplined experimentation loop that includes feedback and documentation.

The second view that all problems are unique, and if understood, do not require significant modeling, documentation, or sequences to the analysis and

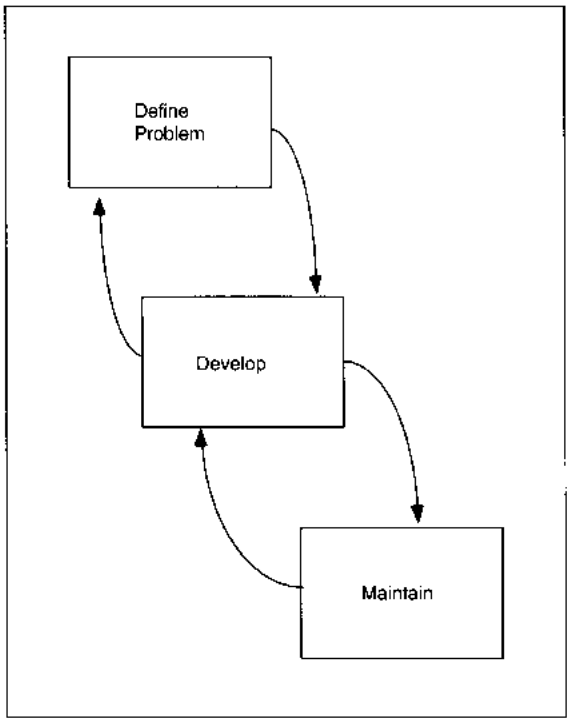


FIGURE 1-22 Generic View of Life-Cycle Development

development events. Since each problem is unique, there is no point in trying to repeat the analysis and design experience. Development is viewed as a *creative* activity that should be unconstrained. There should be no *formal* analysis, design, programming, or testing, even though each of these activities must be performed during the process. This approach denies the need for professional SEs or a profession of software engineering. In fact, it is frequently a cover for ignorance, or an excuse for laziness. This is a hacker's view of the world that is not appropriate to business organizations.

Problems

If building *small* systems (e.g., less than 2,000 lines of code in a 3GL, like Cobol, less than 400 lines of code in a 4GL), the developers, managers, and users may not have problems. Many financial analysis models and small systems in brokerage firms are developed using no life cycle and no methodology. But anything other than small applications are unlikely to perform exactly as desired, may not be completely tested when placed into production, and cannot be integrated easily into existing applications.

A less obvious problem is that this technique relies on individual problem-solving capabilities and knowledge. Studies by IBM and others show individual programmer differences of as much as 16 times in productivity and more than that in accuracy. If the firm using this technique has only the best, top 5% of programmers on its staff, there is little risk. But how many firms actually have these people?

The view that we do not need a disciplined approach to developing applications implies that just anyone can design and build good applications. Yet daily we hear of users who have built complex spreadsheet DSSs only to leave a company with no documentation and no procedures for the next user. We also hear of users (and, regrettably, people with the title software engineer) who are leaders of projects that are canceled after spending millions of dollars, because the pieces just do not work together. For each type of application, there is a price with this view: DSSs without an architecture cannot be extended; ESs without a plan are unreliable and unmaintainable; TPSs without architectures and plans can only ever support one small piece of business; integration across subject data areas is impossible. Even though ES and AI problem solving both use the learn-as-you-go technique, both require a different *kind* of discipline and rigor.

Current Use

As related above, about 75% of all companies in the United States do not use any life cycle or methodology to guide their application development work. With this statistic, it is no wonder that most applications do not perform as intended, are delivered late, overrun the budget, and have unsatisfied users.

Future Use

For emerging technologies, techniques, or conceptualizations of applications, this approach *is* an effective way to nurture development of a field of

knowledge. For these uses, it will remain. Unfortunately, it will also remain for companies who believe that discipline and order cost too much, and who will continue to suffer the risks involved with relying solely on one person's skill and integrity.

In summary, life cycles define a global breakdown of activities in the life of an application. No life cycle prescribes *how* to actually *do* the work within the phases of a PLC. For that definition, we turn to methodologies.

METHODOLOGIES

Methodologies are procedures, techniques, and processes used to direct the activities of each phase of a software life cycle. There are five classes of methodologies: process, data, object, semantic, or none. Each has its own unique view of an application that relates to its historical context, its own shortcomings, problems, and futures. In this section, a brief overview of the classes of methodologies is given with a general list of documents produced by the analysis phase, problems with the methodology, and short analysis of the methodology's current and future use. Much of this material should be review. If it is not review, don't panic. Use this material to learn the terminology for discussing the methods in detail later.

In addition to the methodologies prescribing how to do an analysis and design, a special class of methods advises how to bring users into the process. That class, sometimes called social methodologies, is the last part of this section.

Process Methodology

History

Process methodologies take a structured, top-down approach to evaluating problem processes and the data flows with which they are connected. Process methods developed during the 1970s in response to increasing complexity of application processing, increased complexity of operating system environments (e.g., the IBM 360 generation of hardware), and the introduction of disk file processing with sequential, indexed, and direct access methods. The documentation produced by the process approach[14] includes, for example, context diagrams, data flow diagrams, data store definitions, and structured English process descriptions. In the course of a complete application development, many other types of analysis and design documentations are developed. These additional documents are discussed in the chapters on analysis and design.

Current Use

Individual techniques such as context and data flow diagrams are widely used and also supported in CASE environments. Other techniques have been replaced by newer methods, for example, paper-based data dictionaries have been replaced by CASE repositories or active data dictionaries, file design has been augmented by normalization, entity relationship diagramming, and so on.

Future Use

Process methods as attributed to DeMarco and others will fade as a distinguishable methodology with context and DFDs melded into a collection of techniques that will be used to support methodology customization.

Data Methodology

History

Data methodologies begin analysis activities by first evaluating data and their relationships to determine the underlying data architecture. When the data architecture is defined, outputs are mapped onto inputs to determine processing requirements. The most used data methodology is information engineering (IE) which was described by Finkelstein and Martin.[15] Documentation produced by the data

14 The architects of process methods were Yourdon and Constantine, 1978; DeMarco, 1979; Gane and Sarson, 1979.

15 See Martin, James, *Information Engineering, Book 1: Introduction, Book 2: Planning and Analysis, Book 3: Design and Implementation*, Englewood Cliffs, NJ: Prentice-Hall, 1990; and Finkelstein, C., *Information Engineering*, 1991.

approach discussed in this text is that of information engineering.

As the use of DBMS software became pervasive during the late 1970s and early 1980s, software engineers recognized a need for improved ways of designing data structures. Many methodologies were developed that concentrated strictly on the data aspects of applications with the processing added as an afterthought [cf. Warnier, 1981]. As an attempt to address the entire application development life cycle, Martin and Finkelstein borrowed techniques, packaged them in a new methodology, and integrated them to provide the first 'womb to tomb' methodology. Information engineering, the resulting methodology, begins with enterprise level analysis and proceeds through identification of applications and individual project life cycles. The methodology was not the work of one person; rather it integrates concepts that were thought of as the best at the time including entity-relationship modeling, normalization and other techniques relating to DB design. The enterprise level techniques are adapted and widely used in organizational reengineering.

An example of analysis documentation developed using information engineering includes entity relationship diagrams (ERD), entity hierarchy diagrams, process dependency diagrams, process hierarchy diagrams, and third normal form logical database definition.

Current Use

Information engineering is gaining acceptance in some of the largest U.S. corporations (e.g., Mobil, Texaco) and is used in Australia (where Finkelstein lives) but is not widely used otherwise. Other 'data' methods enjoy regional popularity.[16]

Future Use

Some of information engineering's appeal is its position as the only methodology that represents all levels of organizational analysis from enterprise through application. IE *cannot* easily be altered, at this time, to accommodate object orientation or knowledge engineering. But it will be around for some time with parts of the methodology replaced in a customizing process. Individual techniques such as ERD will gain even more acceptance in the future as data administration increases.

Object-Oriented Methodology

History

Object-oriented methodology is an approach to system life-cycle development that takes a *top-down* view of data objects, their allowable actions, and the underlying communication requirement to define a system architecture. The data and action components are *encapsulated*, that is, they are combined together, to form abstract data types. Encapsulation means that if I know what data I want, I also know the allowable processes against that data. Data are designed as lattice hierarchies of relationships to ensure that top-down, hierarchic inheritance and sideways relationships are accommodated. Encapsulated objects are constrained only to communicate via messages. At a minimum, messages indicate the receiver and action requested. Messages may be more elaborate, including the sender and data to be acted upon.

Object orientation developed during the 1980s and 1990s as producing desirable software attributes (for instance, minimal coupling) espoused since the 1960s. Object-oriented designs can result in software with desirable properties: modularity, information hiding, functional cohesion, and minimal coupling. Like the other methodologies, bad designs lead to bad applications.

Object orientation appears able to support the abstract concepts needed to automate meta-data and meta-meta-data needed for expert, intelligent, and multimedia applications. **Meta-data** gives meaning to data and is information about data. For instance, a name or data type is information about the data in the example (see Figure 1-23). **Meta-meta-data** is information about the meta-data that describes its allowable use to the application. These types of definitions allow you to *plug-in* any hardware

16 Michael Jackson's Jackson Structured Development (JSD) is used in England. Warnier-Orr techniques are used in companies such as AT&T. Chen's entity-relationship approach is used in isolation in many corporations but is also part of information engineering.

Data	Cathrine Ratliff
Meta-Data	Name, Alpha, 16 Characters
Meta-Meta-Data	Type=Data Field, Logical Link = Process, Physical Link, Process, DBMS (EMPL DB)
Data	D01
Meta-Data	Drive Address, Alphanumeric, 3 Characters
Meta-Meta-Data	Type=Disk, Logical Link = I/O Driver Physical Link = SCSI Channel 0
Data	SC01
Meta-Data	Screen ID, 80x20 Alphanumeric Characters
Meta-Meta-Data	Type=3270 Black/White Terminal, Logical Link = I/O Driver, Process Physical Link = SCSI Channel 0

FIGURE 1-23 Object-Oriented Example

device, software, or data to *create* an application environment.

Object orientation is still an immature discipline, undergoing almost daily evolution and change. As such, the details presented for object orientation in this text may be considerably different in five years.

The documentation produced by one object approach for analysis/design includes, for example, a succinct paragraph describing the system, an object list, an object attribute list, an action list, an action attribute list, a message list, and several optional diagrams.

Current Use

Object orientation is the usual approach to developing applications in aerospace and defense organizations, and experiments with its use are occurring in most large companies. Object design appears to be the best suited method for real-time applications, and is useful for on-line applications. It is one of *the* IS buzzwords of the 1990s and appears often in every trade periodical, research journal, and booklist.

Future Use

Keeping in mind that it is neither a complete nor a mature methodology, the current high level of activity implies a future full of object-oriented applications, databases, and CASE tools. When done properly, object orientation appears capable of supporting many complex environments, including: intelligent applications, multimedia applications, and reusable code and reusable design objects. Look for object ori-entation to be around for a long time.

If you only learn one new methodology, this will be a profitable one to learn for the future.

Semantic Methodologies

History

Semantic methodologies are used in the automation of artificial intelligence (AI) applications. AI, like object orientation, is in its infancy. By definition, AI *methodologies* are also in their infancy.

AI applications cover a broad range of intellectual difficulty, ranging from recognizing to reasoning to learning (see Figure 1-24). Most AI applications in business are on the lower end of the AI spectrum, and provide limited reasoning in applications. Businesses are experimenting with more complex uses of AI.

This discussion is about AI applications that reason through problems to achieve expert level competence in a specific area of expertise. These applications are usually called knowledge-based systems (KBS) or expert systems (ES) applications. Most ES contain the reasoning processes of one or more human experts.

Semantic approaches to system life-cycle development automate the *meaning* of objects in the application. For example, a knowledge object might be composed of objects describing a 'legal' hardware configuration. The reasoning process in the ES first asks characteristics of hardware objects that are required for a system (e.g., speed of disk drive, size of disk drive). Then, using the required characteristics as constraints, the ES determines 'legal' configurations that meet the constraints.

At present, data and rules for evaluating data in semantic applications are defined together within the application and not separated as in traditional applications. There is no separation of analysis and design activities per se for semantic applications either. Rather, the task of knowledge engineering encompasses three general tasks: eliciting knowledge from an expert, analyzing it to define the heuristics and data, and automating the information in some logic-based language, such as Prolog.

Current Use

Knowledge-based systems are a growing segment of the applications portfolio in organizations today. This is another class of methodology, along with object orientation, that is in its infancy. Semantic methods are somewhat more well-defined for business use than object methods. But, the extent of special training and expertise required to implement intelligent applications make the knowledge inaccessible to most practicing SEs.

Future Use

There is a significant amount and diversity of research that will result in mature semantic methodologies over the next decade. One major activity in the future will be the addition of expert intelligence to current transaction, query, data analysis, and DSS systems. Semantic method use will continue to be a growth area in IS for the foreseeable future.

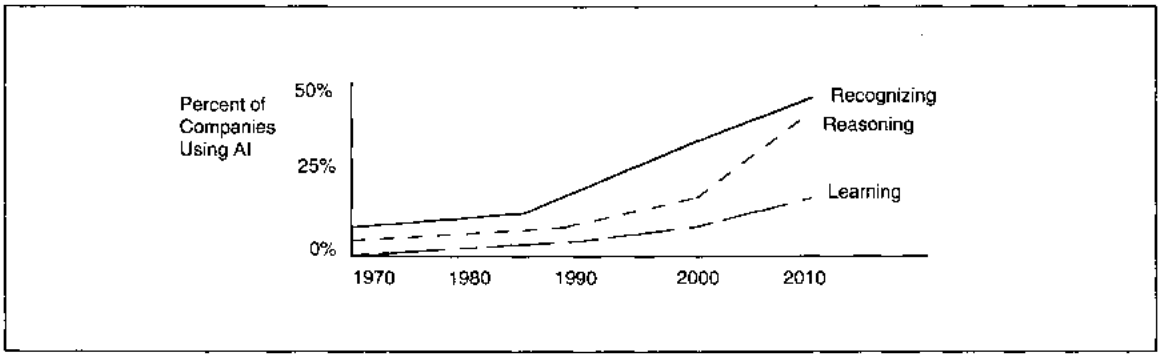


FIGURE 1-24 Range of Artificial Intelligence Applications

EXAMPLE 1-5

STOCK MARKET SELLERS, INC.

Stock Market Sellers, Inc. (SMSI) is a brokerage firm that had a reputation for slow, steady growth and low aggression relative to its industry. In 1988, SMSI embarked on a new, more aggressive position and began introducing new products practically overnight to keep up with its competition.

Automated support for SMSIs new products was the responsibility of Alec Ranier, a young Brit who was a whiz-kid programmer. Alec was promoted several times until, in 1991, he managed a staff of twenty programmers who developed applications to support new products.

When asked about his use of life cycles and methodologies, Alec said, "No, we don't use any of those methodologies or CASE technologies. We don't have time. A broker wants a new product or a new analysis the day after they ask for it, basically."

"Don't programmers have to talk to each other to coordinate their work?" I asked.

He replied, "Not usually. That's how we get away with being so informal."

"What happens when you *do* need to have programmers talk to each other?"

Alec answered, "It is a mess! (laughing) I'll grant that. We redesign, rewrite, and do a lot of code. Another side effect is we reinvent the wheel a lot. We probably have twenty programs that calculate collateralized mortgage obligations and their returns."

I was astonished. "How do you verify their accuracy?"

"Well, we don't because we can't. That is a problem. We're actually trying to design a few key modules to be reusable, but it's a problem because the potential-using programs are all going to need to be rewritten."

I asked, "Do you know any methodologies to help you do that design?"

Alec was honest. "Not really. I'm a good programmer who got promoted. Some day I might learn one but now I just want to 'get product out the door.' "

No Methodology

History

When you develop an application using no methodology, you rely on your own experience and problem-solving ability to automate a solution to a problem. The use of no methodology is implied by the discussion of the learn-as-you-go life cycle. There are no general activities because what is done and how it is done are left strictly to the individual.

Current Use

Most organizations in the United States currently use no methodology. Example 1-5 illustrates the box companies get themselves into when they do not use a methodology. As in the example, companies generally do not recognize any problems. On probing, they realize they have problems but have no idea for getting out of the situation short of rewriting all applications . . . a solution they consider too costly.

Future Use

There are two major reasons why use of no methodology will begin to disappear as a strategy for designing applications. First, trial-and-error is not a productive problem-solving strategy when the requirements for an application *can* be identified. Rather, a lack of methodology indicates laziness,

shoddy work practices, and lack of rigor, usually where it is most needed. Hopefully in the future, more organizations will recognize the need for rigor in developing applications . . . their company's future might well depend on that recognition. Second, in order to use CASE tools and gain *any* of their productivity improvements, *some* methodology is required.

User Involvement in Application Development

Each of the previous methodology discussions approaches the problem of application development as if it were done only by technically oriented personnel. Where in this picture is the user of the application? Ultimately, users must supply information about the business functions and accompanying data that are being automated. In this section, we discuss user involvement in application development so you do *not* think SEs work only with each other. Although early applications *were* frequently built without discussions with users, isolation of SEs from users resulted in systems that might work technically, but often did not meet user needs, and frequently disrupted the workplace.

In the early 1960s, Scandinavians began to voice concerns over the social side effects of applications. Early systems frequently deskilled workers. Socially oriented methodologies of application development were created in response to the concerns about the effects of computerization. **Social methodologies** describe an approach to SDLC that attends to social and job-related needs of individuals who supply, receive, or use data from the application being built. Social methodologies are not really methodologies; rather, they are user involvement techniques. These techniques ignore technology completely and assume that some other approach to the technical aspects of application development is used along with user involvement.

The three main user-involvement techniques are joint application design (JAD), socio-technical systems (STS), and Ethics. The most practical and popular method is **joint application design** (JAD) which requires an off-site meeting of all involved users and systems people, who meet for five to ten days to develop a detailed functional description of application requirements. Daytime meetings are used for new analysis; nighttime meetings document daytime results for review and further refinement the next day.

There are many benefits from user involvement in application development. First, it builds commitment by users who automatically assume ownership of the system. Second, users, who are the real experts at the jobs being automated, are fully represented throughout development. Third, many tasks are performed by users, including design of screens, forms, and reports, development of user documentation, and development and conduct of acceptance tests.

We assume that user involvement is not only desirable, but *mandatory* to truly effective application development product and process. This does not imply that such design *will* result, only that it can. Using a social approach assumes that job enlargement is a desirable by-product of automation.

The most important aspect of user involvement is that it *must be meaningful*. The users must be decision makers and staff who fully understand the impact of their decisions, and who are interested in participating in the development process. Using low-level staff, or assigning 'expendable' managers is *not* the way to have users participate in developing applications. Neither is co-optation of users desired. Co-opting means that you get people to agree with the outcome because they 'participated' in the decision process even though the alternatives are all defined by the application developers.

The goal of user participation is for IS and non-IS people to work together as *business partners* rather than as adversaries. When users participate, *they* make all nontechnical decisions. The SEs explain and shepherd users to make semitechnical decisions, for instance, design of screens. The SEs explain both the impact and reasoning of major technical decisions. If this discussion implies that users call the shots, that is what is meant. User involvement means that users run the project, making the majority of

decisions and having final say on *all* major decisions. The SEs and other Management Information Systems (MIS) staff act as service-oriented technicians, as they are.

In many organizations, the social aspects of work are specifically felt *not* to be within the scope of responsibility of software developers. If the development staff are only technical in their orientation, this is probably true. Then it is the responsibility of the project manager to educate user and IS management about the need to design the organization and jobs as well as the system.

In the United States, high levels of user involvement are still unlikely and usually at the discretion of the project manager. In many cases of 'user involvement', the reality is that users are not involved. Even in companies that have user project managers, IS staff can ignore user desires and build the systems *they* want to build.

SEs and users who have participated in user-involved application development tend to be fully committed to user involvement as a requirement in application development. Hopefully, the days of application development by technicians who never consult with users are gone, or soon will be. Future generations of computer-literate users will demand a say in how their systems are developed. The prognosis, then, is for user involvement to continue slow growth of use in the United States.

Overview of the Book

In this chapter so far, we have prefaced and introduced the major topics of the book. In addition to identifying specifically how the above topics will be used later in the book, there are many more topics that you will also learn that we briefly outline here.

Applications

Applications are the underlying topic of all we discuss in this text. You should already have a fairly good understanding of what an application is. We will not discuss that topic further.

What we will discuss throughout the text is how application types relate to each of the topics. You will get answers to questions such as: Which life cycles and methods are most appropriate to which application types? When do application characteristics and technologies affect the choice of life cycle and/or methodology?

Project Life Cycles

Project life cycles should also have been mostly review. PLCs, per se, are not mentioned again. Rather, the phases of feasibility, analysis, design, testing, language selection, and testing each have their own chapters. One difference between this text and most other texts is that multiple methodologies and deviations from the standard PLC are discussed in the context of each phase.

Part I: Preparation for Software Engineering

Part I prepares you for the tasks of developing and implementing an application. The chapters in this section introduce you to

- research on learning and software engineering to highlight an effective means of studying and practicing this work
- the ABC Video case used throughout the text
- the roles of project manager and software engineers
- methods of gathering information about the task to be automated
- proper behavior during application development

Part II: Project Initiation

After you know how to elicit information, we begin talking about project development. Part II first discusses organizational level re-engineering, a method to developing application plans. Then, feasibility analysis is detailed in the next chapter. These discussions are separated from those about the methodologies because these tasks are assumed by most methodologies. For each chapter, the theories

underlying the concepts are introduced, a method of performing the tasks is described, and examples are provided from ABC to help make the information concrete.

Part III: Analysis and Design

Part III is devoted to analysis and design activities that each take about 20% of application development time. During analysis, the SE concentrates on defining *what* the application will do. During design, the requirements are translated to define *how* the application will operate in its specific hardware and software environment. One representative methodology from each broad class of methodologies is discussed in detail in Chapters 7 through 12. Chapters 7 and 8 discuss analysis and design, respectively, for process methodologies. Chapters 9 and 10 relate to data-oriented methodologies. Chapters 11 and 12 present object-oriented methodologies. Based on ABC's rental processing application, we will discuss what each methodology can and cannot do for you during logical definition of application requirements. For each methodology, the theories underlying its development are described and representative CASE tools available to support application development are provided.

At the conclusion of the methodology discussion, Chapter 13 recaps the graphical representations and thinking processes used in each methodology. The methodologies are compared and contrasted on several sets of criteria. In addition, future developments in technology and applications and their impact on methodologies are developed.

Some tasks are performed during analysis and design, but are not addressed by most methodologies. These forgotten activities are included in this section and discussed in Chapter 14.

Part IV: Implementation and Operations

Many tasks remain to complete an application development, including programming, testing, maintenance, and change management. Each of these topics is related to application and methodology types in Chapters 15 through 18. For every chapter, applicable automated support tools are identified. Chapter 15 discusses the selection of a target language for an application. Code for applications will be increasingly generated by the CASE tool. As CASE use increases, the need to code, then, is replaced with a need to choose an appropriate language.

Similarly, many applications now use purchased software rather than customized code. Chapter 16 discusses the selection and purchasing of hardware, software, or consulting services for application development.

Testing is required of all applications developers at present whether a machine generates the code or not. Chapter 17 discusses different types of testing, testing techniques, and the development of test plans for an application.

Change is a way of life in application development. Chapter 18 deals with the management of change for documents and software. The section on software maintenance describes re-engineering as it applies to deciding whether or not to replace or maintain code. Several replacement options are presented.

Finally, the last chapter discusses careers in software engineering. Keeping current in a profession that constantly changes is a daunting task. In Chapter 19, you will receive tips on the type of reading you should do and the types of professional organizations you might join to enhance your ability to stay current. In addition, you will learn the types of jobs available to you as a novice software engineer and an approach for deciding on a starting job.

SUMMARY

This chapter prefaces and summarizes the contents of the text. Software engineering was defined as a systematic approach to the development, operation, maintenance, and retirement of software. A software engineer is a person who has a broad knowledge of methodologies, life cycles, languages, and all aspects of software development, and who applies that knowledge to the systematic development of application systems. The two main goals of software engineering are to build a quality *product* through a quality *process*.

Next we defined applications characteristics, responsiveness, and types. An application is a set of related programs that perform some business function. The characteristics that all applications have in common are data, processes, and constraints. Application responsiveness reflects whether the application is batch, on-line, or real-time. Finally, application types include transaction processing, query, DSS, and expert systems.

Project life cycle is the breakdown of work for initiation, development, maintenance, and retirement of an application. Alternative project life cycles include sequential, iterative, and the learn-as-you-go. The sequential life cycle includes a series of phases for initiation, feasibility, analysis, conceptual design, design, programming/unit testing, testing, implementation and checkout, maintenance, and retirement.

Methodologies are policies, techniques, and tools that guide the activities of each phase of a software project life cycle. The five classes of methodologies in this text are process, data, object, social, and semantic. Process and data methodologies are fairly mature guidelines for developing applications. Object and semantic are emerging methodologies that help us build systems using artificial intelligence and new technologies. Social methods are really techniques for involving users and assume the use of one of the other four methodology classes as well.

REFERENCES

Boehm, Barry W., *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

Booch, Grady, *Software Engineering with Ada*, 2nd ed. Menlo Park, CA: Benjamin-Cummings, 1987.

Booch, Grady, *Object Oriented Design with Applications*. Redwood City, CA: Benjamin-Cummings, 1991.

Bostrom, Robert P., and J. Stephen Heinen, "MIS problems and failures: A socio-technical perspective," Part I, *MIS Quarterly*, September 1977, pp. 17–28.

Chen, P. P-S. "The entity-relationship model—Toward a unified view of data," *ACM Transactions on Data Structures*, Vol. 1, March 1976, pp. 9–36.

Davis, Gordon, and Margrethe Olson, *Management Information Systems: Conceptual Foundations, Structure and Development*, 2nd ed. New York: McGraw-Hill, 1985.

Department of Defense, *Standard for Application Development, Guideline #2167a*. Washington, DC: US Government Printing Office, 1985.

DeMarco, Tom, *Structured Analysis*. New York: Yourdon Press, 1979.

Eliason, Alan L., *Online Business Computer Applications*, 2nd ed. Chicago, IL: Science Research Associates, 1987.

Feigenbaum, E., P. McCorduck, and H. P. Nii, *The Rise of the Expert Company*. New York: Vintage Books, 1989.

Gane, C., and T. Sarson, *Structured Systems Analysis: Tools and Techniques*. Englewood Cliffs, NJ: Prentice-Hall, 1979.

Gane, Chris, *Computer-Aided Software Engineering: The Methodology, The Products and the Future*. Englewood Cliffs, NJ: Prentice-Hall, 1990.

IEEE, *IEEE Software Engineering Dictionary*. Piscataway, NJ: IEEE Press, 1983.

Lientz, B. P., and E. B. Swanson, *Software Maintenance Management: A Study of Maintenance of Computer Application Software in 487 Data Processing Organizations*. Reading, MA: Addison-Wesley, 1980.

McClure, Carma, *CASE is Software Automation*. Englewood Cliffs, NJ: Prentice-Hall, 1990.

Martin, James, *Information Engineering, Book 1: Introduction, Book 2: Planning and Analysis, Book 3: Design and Implementation*. Englewood Cliffs, NJ: Prentice-Hall, 1990.

Necco, Charles R., Carl L. Gordon, and Nancy W. Tsai, "Systems analysis and design: current practices," *MIS Quarterly*, December 1987, pp. 461–476.

Parnas, D. L., "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, Vol. 15, #12, 1972, pp. 1053–1058.

Sprague, Ralph H., Jr., and Hugh J. Watson, *Decision Support Systems: Putting Theory into Practice*. Englewood Cliffs, NJ: Prentice-Hall, 1986.

Swanson, E. B., *Information System Implementation: Bridging the Gap between Design and Utilization*. Homewood, IL: R. D. Irwin, 1988.

Turban, Efraim, *Decision Support and Expert Systems: Management Support Systems*. New York: Macmillan Publishing Company, 1990.

Yourdon, Edward, and Larry L. Constantine, *Structured Design*. New York: Yourdon Press, 1978.

KEY TERMS

adaptive maintenance
analysis
application characteristics
application responsiveness
application type
automated interface
batch applications
class
coding
computer-aided software engineering (CASE)
conceptual design
configuration management
constraint
control constraint
corrective maintenance
data
data analysis applications
data methodology
data warehouse
decision support applications
declarative language
design
development
economic feasibility
embedded system engineering
executive information system (EIS)
expert systems (ES)
feasibility
goals of SE
group decision support systems (GDSS)
hierarchic logical data model
human interface
implementation
inferential constraint
initiation
input
interactive processing
iterative project life cycle
joint application design (JAD)
knowledge acquisition subsystem
knowledge base
learn-as-you-go project life cycle
logical data model
maintenance
manual interface
meta-data
meta-meta-data
methodology
network logical data model
object-oriented logical data model
object-orientation
on-line application
operations
organizational feasibility
output
perfective maintenance
physical data model
postrequisite constraint
prerequisite constraint
process
process methodology
product
program design
prototyping
quality assurance
query application
real-time application
relational logical data model
retirement
retrieval
SE process
SE product
semantic methodology
sequential project life cycle
social methodology
software
Software Development Life Cycle (SDLC)
software engineer
software engineering
spiral application development
storage
structural constraint
structured problem
subsystem design
technical feasibility
testing
time constraint
transaction-oriented application
Transaction Processing System (TPS)
unit testing
unstructured problem
validation
verification

EXERCISES

1. Develop a table of application characteristics down the rows in the first column, and the application responsiveness levels across the columns. How does each application characteristic differ for each level of responsiveness?
2. Develop a table of application characteristics down the rows in the first column, and the methodology classes across the columns. Begin to develop a comparative table of the way each methodology prescribes documenting the requirements for each application characteristic. You will not be able to complete the table at this point.

STUDY QUESTIONS

1. Define the following terms:

   application characteristics
   batch application
   constraint
   data methodology
   meta-data
   object
   on-line application
   project life cycle
   prototyping
   real-time application
   semantic methodology
   time constraint
   unstructured problem
   validation

2. Define how each methodology's history is affected by technology.
3. What are the four application types and how do they differ?
4. What are the subtypes of decision support systems? How do they differ?
5. What is computer-aided software engineering?

6. What is an application?
7. How do real-time and on-line applications differ?
8. What is the range of artificial intelligence applications? What area do most expert systems cover today?
9. What is the starting point for analysis in a process methodology? for a data methodology?
10. Why is it important to know the orientation of a methodology?
11. If most companies do not use methodologies, why should you learn how to use them?
12. Is some methodology better than none? Is some life cycle better than none? Discuss the pros and cons of using and not using methodologies and life cycles.
13. What are the components of a feasibility study? What type of analysis is performed for each?
14. What are the phases of a sequential development life cycle? How do they vary when you use prototyping?
15. What are the five types of constraints? Give an example of each.
16. What are the four application types? Give an example of each.
17. How do on-line and real-time applications differ?
18. Draw a diagram showing the operation of a typical batch application. Then draw a diagram showing the operation of a typical on-line application. Discuss how they are similar and how they are different.
19. What is the difference between a semantic methodology and an object-oriented methodology?
20. What is quality assurance and when is it performed?
21. What is *meaningful* user involvement?
22. List the three uses of prototyping.
23. What are the dangers in using prototyping?
24. What is wrong with a learn-as-you-go life cycle?
25. What is dangerous about using no methodology and no life cycle?

★ EXTRA-CREDIT QUESTIONS

1. Develop the pros and cons of the ethical issues described in Example 1-5. What is your opinion? How can the open questions be resolved?
2. What can be done to further the involvement of users in applications development? Should this be done? How can it be done in an ethical way?
3. Are methodologies as you know them at this point culture free? How can culture get in the way of their use in a multinational organization?
4. Think beyond this text to the development of applications in a multinational organization. What are cultural and ethical issues in building applications that will be used in many countries of unequal computer resources?

PART I

PREPARATION FOR SOFTWARE ENGINEERING

The four chapters in this section prepare you for the actual work of software engineering. Chapter 2 serves two purposes: First, research on learning and software engineering are summarized to give you some ideas about how to organize the text's material. Good mental maps of the information ease your learning and help you keep the different methodologies distinct. Second, a case describing an application to be built is introduced: ABC Video rental processing. The application is developed in each of the methodologies we will discuss.

Project managers and software engineers perform different duties and are usually different individuals on a project team. In Chapter 3 you will learn the roles of project managers and software engineers and how they complement each other. The kinds of questions we will answer are: What does a project manager do? How does it differ from a software engineer? Why is knowledge of management important to a software engineer?

Last, in preparation for developing systems, Chapter 4 defines techniques for gathering the information you need to analyze and design a system. Then, we will discuss how you should act and how to evaluate what you are told during information gathering. Sample dialogues between ABC managers and the software engineering team illustrate the information presented in Chapter 2.

CHAPTER 2

LEARNING APPLICATION DEVELOPMENT

INTRODUCTION

There is rarely one 'right' solution application in software engineering. Just as in Chapter 1, we said there is rarely one 'right' way of getting a solution for an application. Despite this ambiguity of the software engineering process and product, there *are* approaches to problem solving in software engineering that are more successful than others. Your gaining experience to know those approaches is one goal of this text. To assist you, this chapter discusses how we learn, how we evolve from novice to expert, and how you can apply this knowledge to mastering the material in this book. In the second section, the case study we follow throughout the text is introduced. The case is related to learning approaches suggested in the first section, and to the review in Chapter 1. First, let us turn to learning and the development of expertise.

HOW WE DEVELOP KNOWLEDGE AND EXPERTISE

Learning

There are two basic stages of skill development in learning that we call the declarative and procedural knowledge development stages. In the **declarative**, or *what* stage, we learn basic skills, rules, and activity sequences. We learn declarative knowledge before process knowledge. During the **process**, or *how* stage, we imbed the *what* knowledge into a process. We learn *how* to perform the activity sequences, and *how* to integrate the different rules. In the last part of *how* learning, we internalize both the declarative and process knowledge so they become part of our automatic memory.[1]

The internalization of declarative and process knowledge occurs through

- experiencing real life
- doing classroom exercises
- reading cases and solutions
- developing practice problems with feedback
- studying both good and bad examples

Cognitive psychology and artificial intelligence research describe human thinking as case-based reasoning. A **case** is a predetermined representation of event sequences in a particular setting.[2] During

1 For a complete discussion of declarative and process knowledge, see Chi, Glaser, & Rees, 1982.

2 Kintsch & Mannes, 1987, discuss case-based reasoning. Schank & Abelson, 1977, also writing about artificial intelligence call case-based reasoning "script" based reasoning. The two terms—case and script—are essentially the same.

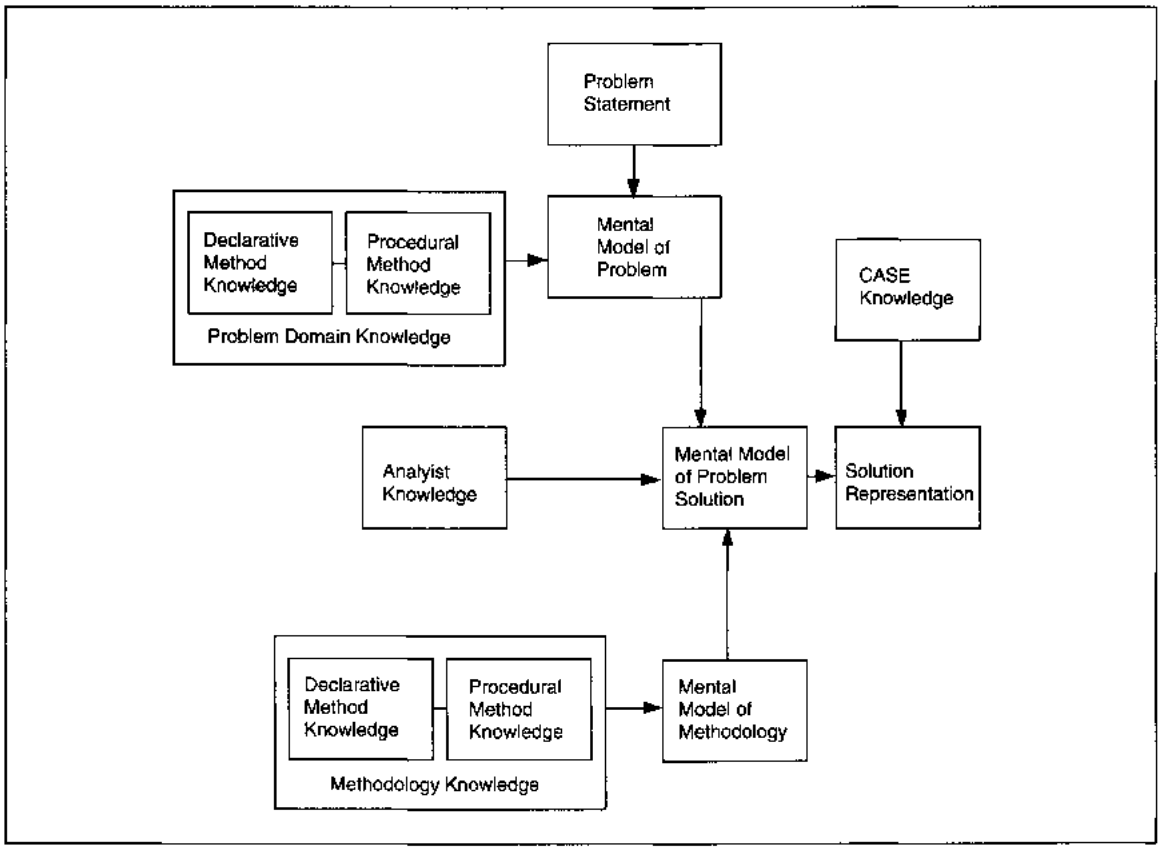


FIGURE 2-1 Interaction of Knowledge Types in Systems Analysis (adapted from Vessey & Conger, 1993)

learning, we recognize patterns of alternatives, expected actions, and decisions that work. After reaching a detailed level of understanding of the patterns, we internalize a *case*, imbedding the patterns, actions, and decisions into our knowledge structure.

In systems analysis, two different types of cases might be appropriate: analysis task and problem task. Figure 2-1 illustrates the information used in analysis and how they interact. The **analysis domain** case is the declarative and process knowledge of actions needed to do the analysis task. We can divide analysis tasks further into subjective and objective activities. Subjective analysis activities are subproblems in application development that accompany all methodologies. Some representative analyst knowledge includes knowing

- what life cycle is appropriate
- what data-gathering technique is likely to be most effective
- when data gathering is complete enough
- when we should iterate through earlier stages of the process

During objective analysis activities, we describe the functioning and design of a proposed application. We may further subdivide objective activities into techniques used, such as methodology or computer-aided software engineering (CASE) tools. When we do not follow a methodology, we rely on our own problem-solving ability and knowledge.

The second type of knowledge required to develop an application is problem task case

knowledge. Problem task knowledge is the declarative and process knowledge of the **problem domain** being automated. For example, order entry–inventory control processing describes a general problem task domain. If we add that the system is for a retail business, it is less general. If we add that the system is for Sears and Roebuck, for instance, it is less general again. During the automation process, we apply our knowledge of how to do analysis to the problem domain. We use analysis knowledge both to describe the current system and to develop the functions of the new system.

Use of Learned Information

Case-based reasoning relies on our recall of past similar experiences, that is, analogous events. **Analogies** are similar experiences that we use to

- classify problems
- plan a course of action
- suggest explanations
- suggest means of recovery from failures

When the analogy matches the current situation, we use it to predict what will happen based on the analogous event. When the analogy *does not* fit, we look for similarities between current and past experiences from which we can generalize to build new analogies.

During the learning process, we build our own examples to help us learn new information. We recognize similarities between different episodes, compile the similar, generalized events, and form a new memory case. This **generalization** process *is* learning. Learning calls for failure of an analogous expectation to work for the current case, followed by explanation of the failure which we make sense of and fit into our own memory as a new case.

Why is the use of analogy so important? System analysis is work that requires judgment and adjustment. System analysis has nonoptimal solutions (i.e., relies on satisficing), and takes place within a bounded knowledge base. Analogical reasoning is better for systems analysis than reasoning by understanding because *analogical reasoning relies on experience* to generate cases while *understanding relies on experimental trial and error.* When analysts have applicable analogous experience, we try to fit that knowledge to the current situation to serve several purposes: understanding of situational dynamics, generating options, and calculating the chance of success of an application option.

In systems analysis tasks, there are frequently one or more aspects of a problem that are unfamiliar to the analyst. In unfamiliar situations, analysts first rely on aspects of the work with which we are familiar, then enlarge and broaden the applicability of our analogical knowledge. But what happens when we do not have the experience to use analogies or our analogies do not appear applicable? Then, we turn to expert/novice differences in problem solving for general tasks to see what happens.

Expert/Novice Differences in Problem Solving

The differences between experts and novices are difficult to pin down. **Expert** analysts are considered to have an extensive, internalized knowledge upon which they draw to apply analogous problem domains and problem-solving techniques to a current analysis task. They work quickly, knowing what they know and what they don't know, and are able to determine at least one workable solution quickly, sometimes within minutes. A **novice**, on the other hand, is slow and unsure, exhibiting some, but not all expert behaviors, and making mistakes throughout the process. Experts and novices differ considerably in their approaches to solving problems. For instance, novices

- develop **local mental models** of problem parts, that is, work on bits of small problems rather than on integrating the bits into a whole. For example, novices concentrate on adding customers instead of concentrating on customer maintenance, including add, change, delete, and retrieval processing.
- use **undirected search** in a trial and error manner (for example, to determine the utility of a new technology). The undirected way is to look through several magazines to see if

they have articles on the technology, instead of looking through a subject index at a library.
- analyze **surface features** (for example, think of control statuses and their allowable values instead of the implications for processing that relate to each value)
- simulate design entities in isolation (for example, simulate video rental processing without paying attention to how it works with return processing)
- misconceive actions (for example, never analyze the complete rental/return cycle)
- fail to integrate the chunked local models into a whole global problem solution (for example, fail to integrate history processing into the rental/return cycle)

Novice problem-solving strategies include satisficing and conservatism. **Satisficing** means to knowingly elect a nonoptimal solution.[3] Novices search for *any* solution; experts search for the *best* solution. **Conservatism** is minimal change of a solution; it means the problem solver takes the first solution rather than testing alternatives. Novices search for alternatives only when the existing method fails, but they cannot always tell that the existing method is failing. So, in becoming conservative, novices use their first conceptualization of a problem. In contrast, experts use optimizing and alternative evaluation in analysis and design. Because of conservatism, novices suffer **breakdowns**—errors in the problem-solving process. Since the process is both constrained and directed by a methodology, the breakdowns relate to the analyst's *mental model* of the problem and *use of a methodology* to develop a mental model of the solution.

Conversely, experts do

- **categorize** problems (for instance, ABC Video Rental processing is a simple form of an order entry problem)
- develop **global mental models** of the problem that they 'see' or visualize the entire problem solution
- use **directed searches** in problem expansion and identification of similar problems
- analyze **deep structures**, not just define terms but analyze their meaning, fit, and the political and technical implications
- use **goals** and **plans** to determine what steps to take in finding a solution
- perform skilled sequences of actions including mental simulation and top-down expansion of the problem

Experts use knowledge of the application development process to direct actions independently from the problem. For instance, regardless of the problem or methodology, you always begin with a definition of the scope of the activity. This abstract knowledge about structuring of a problem, procedures, and process uses internalized cases and plans, and relies on experience. Problem analysis and design involve decomposition of a problem into subproblems, relying on substrategies of analogy and understanding to guide decomposition in a top-down manner. When the problem domain is new and the problem type is new, expansion progresses breadth-first. But, for problem solving in familiar domains, experts prioritize areas on which to focus, using a depth-first strategy for each new area.

With methodology training, practice, and feedback, novice software engineers can display many expert behaviors in a short time, i.e., after analyzing and designing as few as three case problems.[4] Methodologies sequence events, and constrain and direct the actual analytical process. Guidelines and heuristics about what to analyze and how to analyze it are supplied by the method with comments supplied by the text and instructor. Relationships are identified to link each deliverable within a method, associating the thought processes used to develop the deliverables. All of these directed activities speed and simplify both the development of expert behavior and the internalization of methodologies.

Research on whether there are differences between methodologies for facilitating expert

3 See Simon [1960] for a more complete discussion of satisficing and decision making.

4 See Vessey & Conger, 1993, for an example of this type of study.

behaviors is in its infancy. Several laboratory studies by the author and others identify process methods as easier to learn, with no noticeable difference between methodologies in the delivered quality of the resulting proposed logical system. One thing we *do* know is that not all methodologies work equally well for all problems. This information will be discussed in Chapter 13.

How to Ease Your Learning Process

In this text, we assume that you want to go beyond knowing the basics of systems analysis and design, but that you *do know* the basics. We assume you have a working knowledge of structured systems analysis and design, data base, and programming. Most systems analysis and design courses practice developing data flow diagrams. In this text, we will discuss DFDs and compare and contrast them with other methods, building on your current state of knowledge. If you don't feel confident about your ability to draw data flow diagrams, there are exercises at the end of Chapter 6 for practice. For database knowledge you should know and understand the value of normalization, and you should be familiar with SQL and at least one database package. For programming, you should have practice with some procedural language (e.g., Cobol) writing and debugging programs that read sequential files to generate reports. Knowledge of data structures, files, and a structured language, such as Pascal, is helpful but not necessary to using this book successfully.

Application development is essentially a problem-solving exercise which is unique because there is rarely one *right* or *best* answer to an automation problem. Practitioners and professors of application development will both tell you that the best way to learn software engineering is to "Do it!" A quote to support this idea comes from Confucius:

> I see and I forget,
> I hear and I remember,
> I do and I understand.

In doing, you *will* make mistakes, get confused, and think you are completely wrong. Don't give up. Ask questions. Since we learn declarative knowledge first, try to remember as much of the procedural *what* knowledge as you can while you read the text.

Try to think like an expert. Try to develop a global picture of the problem, methodology, or other subject in your mind and develop a plan of attack for your work session. Try to categorize problems both that you are working on and that you are having with the work. Analyze your thought processes to develop a better understanding of your problem-solving approach. See if you can mentally simulate your application design, asking yourself how complete it is and how well it solves the problem. Attempt to analyze the 'deep structures' by asking what each term *means* and what it *implies*. Talk about all of these thought processes both with your instructor and with other students.

Practice your reasoning *process* by reviewing the example in the text, by working through problems at the end of each chapter, and by talking to other students about the reasoning you used to develop your representations. Try different ways of doing the same thing. When you find mistakes, try to learn *why* what you did was not the best, and *how* you could have reasoned to develop a better answer. Through these processes, you will learn valuable problem-solving skills that will be useful throughout your career in IS.

APPLICATION DEVELOPMENT CASE

Now, we are going to switch gears, away from the theoretical to the realistic. In this section, we present the case used throughout the text. The setting, a video store, is used for two reasons. First, it is a simple business that should allow you to build an accurate, complete mental model. A complete mental model is crucial to developing an accurate solution in *any* methodology. Second, most of us rent videos and have analogous knowledge that we can practice using. As you read the cases, try to apply the ideas discussed in the previous section. Ask yourself,

What is the 'big' picture? Do I understand this problem? Use analogies from your experiences as a video store customer (or clerk) to the way Vic wants to run his business.

The case—ABC Video Rental Processing—is representative of the class of order processing/inventory control problems. Through its processing, customer, inventory, and order files are maintained. In addition, ABC Video Rental Processing also is unique in that the video rental business is different from other businesses, and ABC's video rental processing is distinct from other video rental businesses.

ABC Video rental processing similarities and differences from other types of order processing applications highlight the importance of knowing how to learn. The similarities allow you to use analogy to determine the general requirements of the application. For instance, all order entry applications require *customer*, *order*, and *inventory* databases. Conversely, each company does its own detailed processing for order fulfillment. In ABC's case, it is a *rental* company, not a *sales* company, and rentals are not handled the same as sales. So even if you already know *order processing*, only a portion of the knowledge will be applicable to the rental situation. Keep this in mind when you discuss your own video store experiences. Each store has its own 'brand' of processing that might differ from ABC's. You must constantly evaluate the applicability of your past experience to the current situation, trying to use everything possible without forcing inappropriate past knowledge on the new client's application. Next, the context of the industry is described.

History of the Video Rental Business

The video rental industry experienced phenomenal growth during the 1980s. The cost of entry into the industry was low, every mom-and-pop store, supermarket, and small time entrepreneur entered the market. There was no stability in the market and competition was fierce. For instance, some businesses required "membership fees," others did not. Some businesses charged one price for all rentals, usually about $2.00 per videotape per day. Some businesses offered promotions, such as "Two-Fer-Tuesdays," for which two tapes were the same price as one.

Soon businesses recognized that 80% of their videos were rented within 20 days of a tape's release into the market. With this recognition, video stores introduced a two-tiered pricing system, charging a *new-release* price and an *old-release* price. The market began to destabilize and small store owners, for whom the business was a sideline, were forced to decide if they wanted to devote the floor space to videos which soon became obsolete, or if they would abandon the business. They abandoned the business in droves and the video rental industry went through a period of consolidation.

The business today is stable, but is becoming monopolized by large chains: RKO and Blockbuster, for instance. ABC is an anomaly in this market because it is still a one-person, one-store operation. Vic, ABC's owner, would like to offer unique and useful services with a minimum of 'bureaucracy' in the process, and to eventually franchise his business expertise. With these goals in mind, we turn to his business requirements for defining the video order processing application Vic wants to build.

ABC Video Order Processing Task

ABC Video rents video cassettes to customers. Since this business is becoming more competitive, Vic, the owner, wants to automate rental processing, inventory maintenance, and an expert system to speed and simplify the rental process. Vic prepared information for the consulting team to begin work. Vic tried to separate what he wanted from what he needed. So, the application business requirements are listed. Then, Vic's 'vision' of the application is presented.

General Requirements (Excerpted from a memo from Vic to consultants)

. . . ABC Video currently owns two PC ATs and can get IBM compatible PCs cheaply. I would like all the

machines hooked together somehow to share the information and have some equipment backup in case a PC breaks down. Each PC will have a printer for two-part forms. If the customer wants a copy of an order, he or she takes the top copy and signs the bottom. I need a signed copy to legally charge for unreturned tapes.

I want to minimize typing throughout all the processing. Bar code readers are cheap. Can we use that technology for keeping track of rentals?

There are three to six clerks doing rentals at any one time, sharing machines. Rental/return processing is about 90% of the business. Machines should be allowed to do any processing, but should stay set at rental/return processing once there. I want to be able to know where every tape in the store is—out on rental, on the shelf, or waiting reconditioning.

Business requirements relate to customer, videos, rentals, and history information. Each of these requirements are listed below.

Customer Requirements

Customers are people who desire to rent videos for one or more days.

1. All customers must be 'registered'. This means they must have an *easy to remember* identification code, plus their phone number, name, address, credit card number, credit card type, and credit card expiration date on record before they may rent videos.
2. All members of a household should be able to share the same identification number.
3. Customers are required to pay rentals in advance and settle late fees before any new rentals are allowed.
4. Customers can return tapes in three ways:
   - Drop off through a slot in the door
   - Drop off at the desk as they walk in to get new videos
   - Drop off as they take out new rentals
5. Customers who fail to return tapes or damage tapes are charged for the video on their credit cards. Their customer record must be marked 'bad credit risk' and they will not be allowed to rent videos.
6. Retrieval of customer information must be allowed by identification, phone, name, address, zip, or credit card number.
7. All fields must be allowed to be changed as required.
8. Reports on number of new customers by month, by year, 'bad credit risk' customers, late returning customers, expired credit card numbers must all be allowed.
9. Deleting of customers must be allowed by the manager (Vic) only.

Video Requirements

Videos are taped movies, sports, or music events that are rented to customers.

1. All videos received in the store must be 'registered' and tracked. Minimum information is identification number of copies, title, vendor, code, and date received. Video registration should use some technology (a bar code reader?) that does not require typing.
2. Individual copies of videos should be identifiable for rental/return processing.
3. All copies of a title must be identifiable to track rental trends.
4. Counts of the number of rentals by copy and by title should be available for reporting.
5. Retrieval of video information for reporting must be allowed on any single or multiple criteria. Common reports needed will be for maintenance (based on how many rentals), number of tapes and rentals by type (e.g., musical, horror, drama, comedy), and for tapes that have not rented in the last *x* days.
6. I don't know how hard or expensive this is, but I would like some history information, such as
   - rentals by copy by title
   - days rented by month by year by copy by title
   - rentals by customer so I can warn them when they try to re-rent a title
7. Future provisions should allow for
   - tracking the number of days of rentals by copy by title or by dates of rentals
   - multiple rental products (such as VCRs, camcorders, CDs, video games, Nintendo game sets, and so on)
   - automatic debit card or credit card payments
   - variable rental charges based on promotions, date of receipt, and so on

Rental Processing

1. First, NO BUREAUCRACY! Second, the process MUST BE EASY. The rental process must not require customers to carry a card, must not require clerks to type much, and must be easy to learn. Return processing must also be simple and flexible.
2. To take out tapes, customer ID and video IDs are entered. All other information should be pulled from the computer.
3. The system should compute total charges, include late fees, and compute change for money entered.
4. The computer must be hooked to a cash drawer or cash register that unlocks when the money is entered.
5. A printed copy of orders must be kept and signed by customers. These go to accounting and are reconciled at the end of the day.
6. End of day totals for the cash registers must show a total number of tapes out, cash paid, tapes in, on-time tapes, late tapes, late fees, and a total amount of money in for the day.

Vic's Vision of Rental/Return Processing

Customers choose videos for rental either by taking the empty box from a shelf in the store or by telling the clerk the video name(s). The clerk retrieves the tape(s), which are filed alphabetically by name. The clerk enters customer identification (could this be phone number?) into the system to retrieve the customer's record and to create an order. Any late fees from previous rentals must be settled before a new rental can occur.

The clerk uses a bar code reader (or other scanner) to scan the video identifier and enter videotape identification into the system. For each video bar code entered, the system completes the rental detail line on the screen with today's date, videotape identification, video name, and rental price. When bar code IDs for all videotapes to be rented have been entered, the system computes the total fee, automatically computing and adding in sales tax. Late fees may be added to the total if any are outstanding. The customer is told the total amount and the money is paid.

When the clerk enters the money amount into the system and puts the cash into the cash register, the system reduces the amount paid by the total fee amount to obtain the amount of change due to the customer. The amount due to ABC for the rental is reduced to zero on the order. The customer signs a copy of the order form as it is printed on a printer and takes the video(s) home.

On return of tapes, the clerk scans the bar code IDs of the videos. The system should retrieve and display the order with the return date and any late fees added to the detail line. If either there are no late fees or late fees are settled upon return of the video, the order is deleted from the system and the history of use information for the tape is updated. Late fees, and the order information about tape(s) rented that caused the late fee(s), remain on file until they are paid.

Trend analysis should include query capabilities with statistics built in. This should be available on an ad hoc basis without having to anticipate all queries and/or types of analysis in advance. Part of the analysis is used to determine how many tapes of each film to purchase. Trends might be based on sequential nights of rental, number of nights rented within the first 20 days, number of nights rented within the second 20 days, and so on. Each individual tape, even though it might be the nth copy of m copies of the same film, should be identifiable for this analysis. These requirements are not included with the description of required file information above, because you should determine the best way to supply this information.

Discussion

Let's stop here a moment and think about the ABC Rental Processing case. First, get a global mental model of the problem. The problem is to automate rental/return, customer, and video inventory processing, including totaling of orders, computing change, monitoring of late returns, and creation of historical information. This sounds like a complete statement of problem scope, and it could be used for that purpose. In this case, the problem is small enough to hold most of the functions in mind at once.

Do you know enough to automate the problem? No, you do not, not if you want to do it properly. The processes, in terms of how a customer will interact with ABC personnel, are fairly simple. Rental processing has fairly well-defined data requirements and business requirements about how to do the process steps. The flow of processes for rentals still

needs elaboration, but is complete enough for understanding the general problem.

What don't you know? The kinds of questions we will ask will be details of what we already know: How many? How often? What about variations on the process? Questions will also elaborate on constraints and determine if there are interfaces. Some examples of specific questions include: How many videos are there in the store? How many new ones arrive each month, week, day? How many customers are there? How many rentals per day? What kind of security is needed? Does Vic already have software in mind for this application?

There are many more questions we will ask as we move through the text, and the type of questions varies with the methodology. Even with many questions, we *do* know quite a bit about the overall process and Vic's ideas for how the process should work. We know much less about specific details of the operation that we need to fully understand the problem and devise a workable solution. We will get more details as we progress through the text.

In terms of the Chapter 1 discussion on types of applications, rental processing will be on-line with interactive processing. It is a transaction processing application with some query processing. The rental application transaction portion automates the paperwork of rentals, returns, and payments for rentals. The query and reporting part of the rental application uses predefined data in a read-only manner, and has predefined reporting requirements as well as ad hoc reporting requirements. The rental processing case is used throughout this text to reason through each methodology.

SUMMARY

In this chapter we explained the nature of learning and experience. Declarative knowledge is knowledge about *what* actions, procedures, or steps are taken to perform some task. Declarative knowledge is a required but incomplete learning. Process knowledge is knowledge about *how* to perform, reason, and integrate the steps we know from declarative learning. While we learn, we form analogies or cases that form patterns of experiences. When we match a pattern from experience with some current problem, we use analogical thinking. When a past experience does not match some current problem, we analyze the differences to develop a new case based on the new situation. The internalization of cases in our memory is learning.

Novices differ from experts in their problem-solving approach. Novices make mistakes because they do not have a global view of a problem, cannot mentally simulate a solution to the problem, and do not see connections and meaning in problem parts. Experts are able to analyze novel problems because they use analogies from their experience to develop a global view of the problem, can take a top-down view of what they know and do not know, can simulate their solutions mentally, and understand connections and meaning in problem parts. Several tips for practicing software engineering were provided to speed and simplify your learning.

The case company, ABC Video, and its role in the video rental business was described, rental-order processing details were developed.

REFERENCES

Adelson, B., and E. Soloway, "The Role of Domain Experience in Software Design," *IEEE Transactions on Software Engineering, SE-11*, Vol. 11, 1985, pp. 1351–1360.

Jeffries, R., A. A. Turner, P. G. Polson, and M. E. Atwood, "The Processes Involved in Designing Software," in *Cognitive Skills and Their Acquisition* (J. R. Anderson, ed.). Hillsdale, NJ: Lawrence Erlbaum Associates, 1987, pp. 255–283.

Kintsch, W., and S. M. Mannes, "Generating Scripts from Memory," in *Knowledge Aided Information Processing* (E. van der Meer and J. Hoffman, eds.). NY: Elsevier Science Publishing Co., Inc., 1987, pp. 61–80.

Klein, G. A., and R. Calderwood, "How do People Use Analogies to Make Decisions?," in *Proceedings of Case Based Reasoning Workshop* (J. Kolodner, ed.), DARPA/ISTO, Clearwater Beach, FL, May, 1988, pp. 209–218.

Littman, D. C., J. Pinto, S. Lechovshy, and E. Soloway, "Mental Models and Software Maintenance," in *Empirical Studies of Programmers—1st Workshop*

(E. Soloway and S. Iyengar, eds.). Norwood, NJ: Ablex Publishing Co., July 5–6, 1986, pp. 80–98.

Schank, R. C., and R. P. Abelson, *Scripts, Plans, Goals and Understanding*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1977.

Schank, R. C., "Explanation: A First Pass," in *Experience, Memory and Reasoning*, (J. L. Kolodner and C. K. Riesbeck, eds.). Hillsdale, NJ: Lawrence Erlbaum Associates, 1986, pp. 139–166.

Shemer, I., "Systems analysis: A systemic analysis of a conceptual model," *Communication of the ACM*, Vol. 30, #6, June, 1987, pp. 506–512.

Simon, H., *The New Science of Management Decision*. NY: Harper and Row, 1960.

Vessey, I., and S. A. Conger, "Requirements specification: Learning object, process, and data methodologies," *Communications of the ACM*, accepted for publication, 1993.

Wand, Y., and R. Weber, "A unified model of software and data decomposition," in *Proceeding of the 12th International Conference on Information Systems* (J. I. DeGross, I. Benbasat, F. DeSanctis, and C. M. Beath, eds.). NY: SIGBDP, Association for Computing Machinery, 1991.

KEY TERMS

analogy
analysis domain
breakdown
case
case-based reasoning
categorize problems
conservation
declarative knowledge
deep structures
directed search
expert
generalization
global mental model
goal
local mental model
novice
plan
problem domain
process knowledge
satisficing
surface features
undirected search

EXERCISES

1. Develop pseudo-code for ABC Video's rental processing system. Identify and discuss what the essential portions of rental processing are. Discuss which procedures could be either included or omitted without changing the essence of the problem. (Note to Instructor: This is a useful exercise to ensure that all students have a good understanding of the problem.)
2. Describe a work situation you have experienced. Discuss the organization: the structure of the organization, its goals, its strategies for meeting its goals, its culture, its managers' style, the social life at work.
   A. Describe your job and how your job contributed to the organization's goals. Describe the computer applications, if any, you used in your job. Analyze what you did on your job and recommend computer applications that could have streamlined, enhanced, or broadened your job. Do you have the 'big picture' of the company and your job's role? If not, how would you go about developing a global view?
   B. Describe some area of the organization (you may or may not have worked there) that could use an application to speed its work, make its work more accurate, enhance jobs, provide better information to workers, or simplify work life. Describe the application and how it would meet its goals.

STUDY QUESTIONS

1. Define the following terms:
   analogy
   conservatism
   declarative knowledge
   global mental model
   problem domain
   satisficing
   surface features
2. Which comes first—declarative knowledge or process knowledge? Why? How does learning work?
3. Why and how do we use analogies?
4. Why are analogies better used in systems analysis and design than a trial-and-error method of problem solving?
5. Describe the details of what it means to rent a tape at ABC. How do the manual processes translate into computer processes? Use analogies from your own experience to discuss rentals.
6. Make a list of questions you have about ABC order processing that still need to be answered.

Use analogies from your own video rental experience to identify issues that still need to be resolved.

7. Describe the details of what it means to return a tape. How do the manual processes translate into computer processes? Identify subprocedures for which you have choices about when and how they are performed.
8. How do you *develop* a global mental model of some problem? How do you know *if you have* a global mental model of some problem? How do you validate your mental model?
9. What does it mean to create historical information? When does history get created? In the ABC case, is history created at video rental time? or at video return time? or at some other time? How do you know when you have the correct answer to this type of question?

★ EXTRA-CREDIT QUESTIONS

1. Write a one page analysis of some work experience you know about. Describe some function and how it contributed to the organization's goals. Describe the computer applications, if any, used in the function. Analyze the job and recommend computer applications that could streamline, enhance, or broaden the function. Make a list of questions you need answered to gain a complete understanding of the problem areas.
2. Draw a diagram or verbally describe (in pseudocode or your own words) how ABC Video performs order processing. Make a list of questions you have about ABC order processing that still need to be answered. Describe how your experience as a video store customer helps you understand what ABC is trying to do. Describe, from your experience as a video store customer, how you think a video store should be automated. How does it differ from Vic's desires? What should you do about those differences? What are Vic's goals for the application in addition to processing rental/returns? What features might you consider for the application to meet those goals? List three functions you can put in the system to help meet Vic's goal of "no bureaucracy."

CHAPTER 3

PROJECT MANAGEMENT

INTRODUCTION

The role of the software engineer (SE) differs from the project manager in that the SE provides technical expertise, while the project manager provides organizational expertise. Depending on the size of an organization and project team, one person might perform both roles. Small project teams (i.e., less than five people) and organizations with limited software development staff (i.e., less than 10 people) expect one person to assume both software engineer and project manager roles. The larger the organization, the more likely the functions are split and the more extensive each person's experience is expected to be.

The project manager and software engineer are responsible for tasks that include both complementary and supplementary skills. In general, the *software engineer* is solely responsible for management of the life cycle, including the following areas detailed in Chapters 4 through 14:

- Management and conduct of development process
- Development of all documentation
- Selection and use of computer-aided software engineering (CASE) tools
- Elicitation of user requirements
- Technical guidance of less skilled staff
- Assurance that representation techniques, such as data flow diagrams, are correct, consistent, and validated
- Oversight of technical decisions
- Assurance that constraints (e.g., two-second response time) are identified and planned as part of the application

Complementary activities are activities that are performed jointly but with different emphasis depending on the role. Complementary activities include planning the project, assigning staff to tasks, and selecting from among different application alternatives.

The project manager (PM) is solely responsible for organization liaison, project staff management, and project monitoring and control. These major responsibilities are discussed in this chapter.

When one person or another is identified as solely responsible for some activity, it does not mean that they alone *do* the work. The SE and PM are team leaders who work together in all aspects of development. The SE may have project management experience. Sole responsibility means that when a disagreement occurs, responsibility for the final decision rests with the responsible person. Different management styles determine how open a manager is to suggestion and discussion of alternatives.

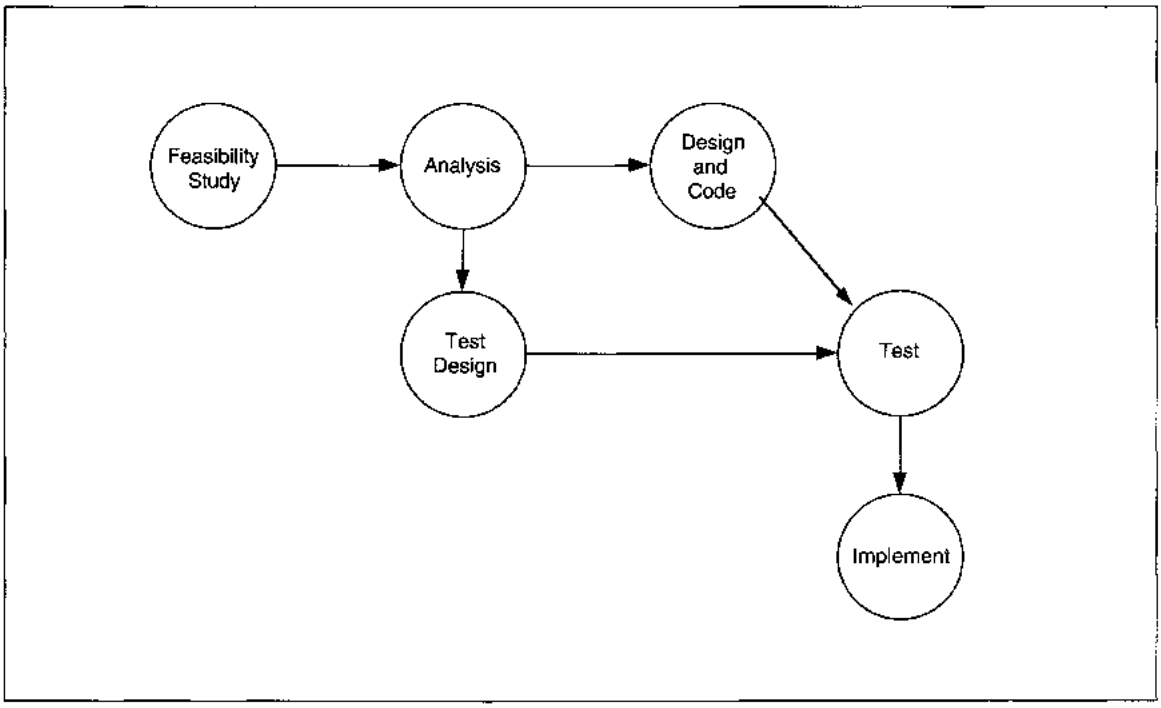


FIGURE 3-1 Example of Too General a Plan

A short discussion of appropriate behaviors for project managers is also included in this section. These behaviors are the project manager's responsibility toward the project.

First we discuss the joint SE and PM activities. Then we discuss activities for which the project manager is solely responsible. Management styles and a brief discussion of project manager responsibilities to the project team are included in the section on personnel management. The last section lists computer-aided support tools for project management.

COMPLEMENTARY ACTIVITIES

Joint activities of the software engineer and project manager include project planning and control, assigning staff to tasks, and selecting from among different alternatives for the application.

Project Planning

To plan the project, the project manager works with the SE to determine human, computer, and organizational resources required to develop the application. While a detailed discussion of planning is included in Chapter 6, the aspects of special interest to the project manager are in this section.

A **project plan** is a map of tasks, times, and their interrelationships. It can be very general (see Figure 3-1) or very specific (see Figure 3-2). Neither extreme of plan is very useful although some plan is better than none. A rule of thumb for level of detail is to define activities for which a weekly review of progress allows the SE and project manager to know whether the schedule is being met. Figure 3-3 shows an example of a well-defined plan.

The general methodology of planning is as follows:

1. List tasks. Include application development tasks, project specific tasks, interface organi-

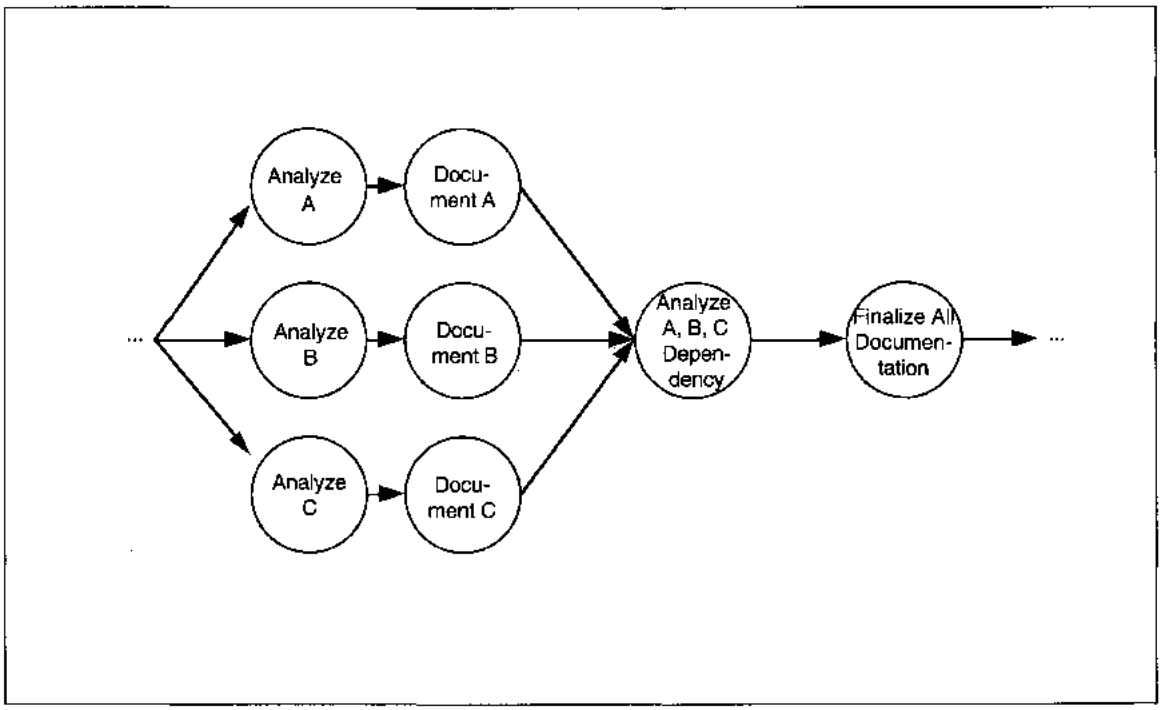


FIGURE 3-2 Example of Too Detailed a Plan

zation tasks, and review and approval tasks.
2. Identify dependencies between tasks.
3. Assign personnel either by name or by skill and experience level.
4. Assign completion times to tasks; compute the most likely time for each.
5. Identify the critical path.

The project manager and SE share responsibility for developing the plan. The *SE's responsibility* is to know all of the tasks relating to the application being developed; the *project manager's responsibility* is to ensure that all organizationally related tasks are included in the list. (The application tasks are discussed in Chapter 6.) Organization tasks include the following:

1. Review documents for completeness, content, consistency, and accuracy.
2. Negotiate, agree, and commit to start and end dates for work.

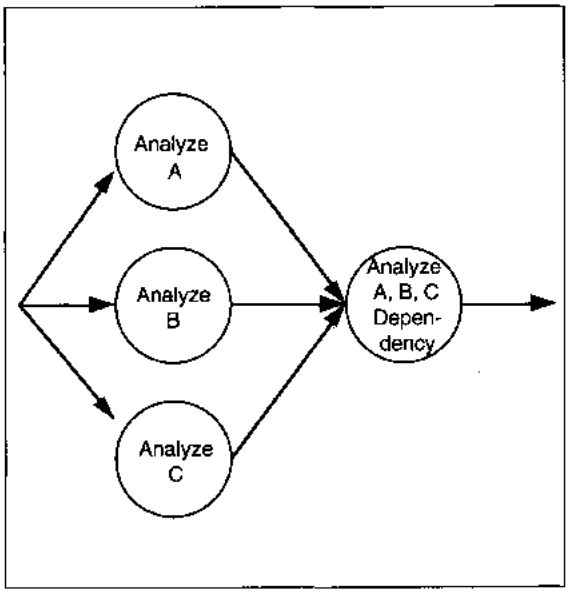


FIGURE 3-3 Example of Acceptable Level of Detail

3. Define necessary application interfaces; plan for detailed interface design work.

All documentation, plans, and design work of the project team is subject to review by at least the user/sponsor. Many other departments or organizations might also be required to review some or all of the work. These organizations might include managers of IS, users, quality assurance, legal, audit, operations, other application groups, government regulators, industry regulators, or others. Each organization applies its specialized knowledge to the application documents to assess their adequacy.

The second task is to obtain agreement and commitments from outside agencies or departments. Frequently, resources and work are provided by other departments. Clerical support, for example, might be from an Administrative Services Department. Operations departments supply support in terms of computer time, memory, disk space, terminals, logon IDs, access to software environments, access to data bases, and so on as necessary to develop and test the application. Auditors frequently want to comment on auditing plans and change the design based on their findings. Quality assurance departments might review documents to find inconsistencies and errors that require correction. Vendors might need to install hardware, software, or related applications that need liaison from the project team and testing once installed. All of these activities need to be scheduled and planned. Since dates for commitments might not be known when the plan is developed, the plan contains the dates at which contact should be initiated and dates by which the commitment must be made in order not to impact the delivery date.

Third, the project manager obtains requirements for application interfaces from other application areas. An **interface** is data that is sent or received between applications. The interface application areas might be in the same company, but might also be an industry group or a government organization. The plan reflects dates by which contact should be initiated and by which the information is required.

If a make-or-buy decision will be made, the project manager and SE work together to develop the subplan for this decision. Subactivities relating to acquisitions include creating and submitting requests for a proposal (RFP), obtaining vendor quotes, evaluating vendor quotes, selecting and obtaining management approval for a vendor, negotiating contract and delivery dates, and planning and testing of the acquired item.

When all of the items are identified, they are related to each other. Tasks that are related are drawn on a **task dependency diagram** showing the sequences of dependencies. Sequences may be interdependent (see Figure 3-4). When all sequences of tasks are on the diagram, independent tasks are added. Milestones, such as the completion of a feasibility analysis document, are shown and are visually obvious because the preceding sets of tasks all feed into that task. Task sequencing can vary depending on the methodology used. (See Chapter 6 for more on this topic.)

Sequencing tasks is the first step to identifying the critical path of tasks for the application's development. The **critical path** is the sequence of dependent tasks that together take the most development time. If any of the tasks in the critical path are delayed, the project is also delayed. So, the critical path tasks are the greatest source of risk for project completion.

The next step is to estimate the amount of work. For this discussion, we assume the project manager and SE assign times to tasks based on their experience (i.e., reasoning by analogy). Other methods are discussed in Chapter 6. Times are assigned to each task based on its complexity and amount of work. Three times should be estimated: an optimistic time, a realistic time, and a pessimistic time. The formula used to compute the most likely time is shown in Figure 3-5. The figure weights the most likely, realistic time by a factor of two in relation to the other estimates.

While times are being assigned, the skill sets and experience levels of a person to do this task should be defined. The list of skill sets and experience levels is used to determine how many people and what type of people are required on the project for each phase. Other assumptions will surface, and a list of them should be kept, as shown in Table 3-1. The assumptions become part of the planning document.

When resource requirements and timing are complete, several activities take place. The SE develops

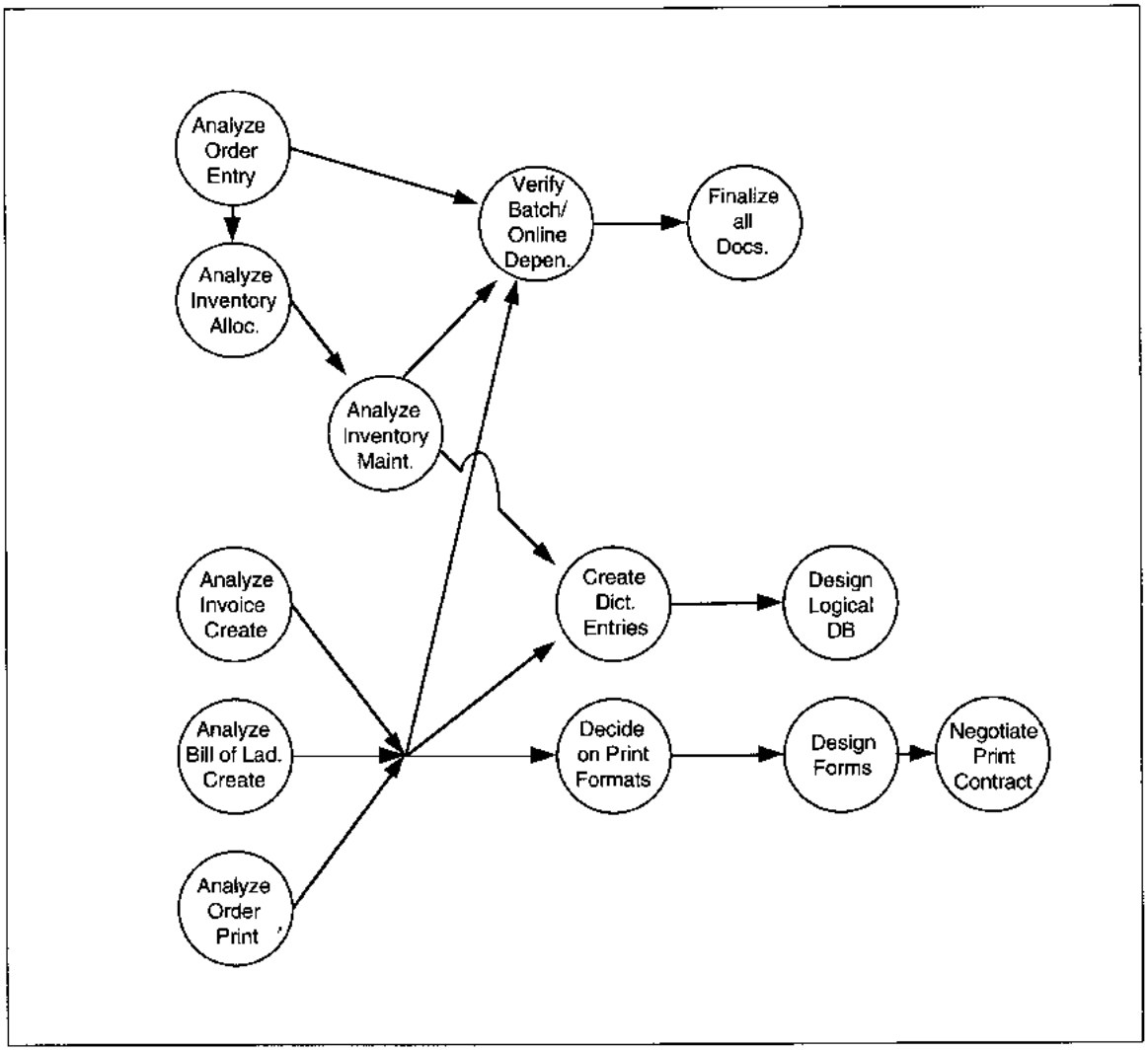


FIGURE 3-4 Example of Interdependent Sequences of Tasks

(O + 2R + P) / 4 = Most Likely Time Estimate

Legend:

O—Optimistic Time Estimate
R—Realistic Time Estimate
P—Pessimistic Time Estimate

FIGURE 3-5 Formula for Determining Schedule Time

a schedule; the project manager develops a budget. They both identify the critical path and discuss it in terms of potential problems and how to minimize their likelihood. Task definitions are made more detailed for critical tasks, to allow more control and monitoring.

When complete, the plan, schedule and budget are submitted to the user and IS managers for comment and approval. Work begins, if it hasn't already, with the plan used to guide project work. The plan is used by the project team to see where their work

TABLE 3-1 Project Assumptions

Type Assumption	Example
Availability of configuration, component of mainframe, special hardware, programmer support equipment, tools, time	Programmers will gain access to IEW by September 10, 1994.
User time involvement. This may be expressed in time per day for a number of days, or may be in number of days.	A middle manager representative from Accounts Payable will be available in a Joint Application Design session scheduled for June 1–5, 1994.
Need for services from audit, law, vendors, quality assurance, or other support groups	The Audit Department will be able to review and comment on the adequacy of audit controls within 7 business days of receiving the review document.
Software performance	The Database Management Software will be able to process 10,000 transactions per day.
Test time, terminal time, or test shot availability	Batch programs can be tested simultaneously with on-line programs.
	Batch programs will be able to average three test runs per day with an average turnaround of less than 2.5 hours.
	Batch programs will be less than 160K and will require no more than two tape mounts each.
Disk space	Operations will make available 100 cylinders of IBM 3390 disk space for the project beginning 9/10/94. An additional 50 cyl. will be added for test databases by 10/30/94. An additional 250 cyl. will be added for production database conversion by 11/30/94.
Memory, CPU time, tape mounts, imaging access, or other mainframe resources	For testing, 30 CPU minutes per day plus 75 hours of terminal access time will be required beginning 10/30/94.
Personnel	Two senior programmer/analysts with 2–3 years of Focus experience and 2–3 years of on-line, multiuser, application development experience is required by 6/30/94.
	Four programmers with 1–2 years of Focus experience and one year of VM/CMS experience is required by 7/15/94.
Hardware/software availability	Imaging equipment will be available for application testing no later than 9/10/94.
	15 PCs or IBM 3279 terminals will be available for access and testing use no later than 9/10/94.

fits in the whole project, and it is used to monitor progress toward project completion.

The plan should *never* be cast in concrete. Plans should change when the tasks are wrong, times are underestimated, or there are changes in project scope that alter the activities performed in some way.

Assigning Staff to Tasks

Task assignment is fairly straightforward. The major tasks are to define the tasks and skills needed, list skills and availability of potential project members, and match people to tasks. The project manager and

SE actually begin discussing possible project staff when they are planning the project and tentatively assigning people to tasks. Then the project manager's real work begins.

The hard part of an assignment is the judgment required to match people whose skills are not an exact match for those needed; this is the usual case. For instance, you might want two programmer analysts with the following list of skills:

- design and programming experience on a similar application
- three to five years experience in the operational environment
- one to two years of experience with the database software
- managerial experience for two to four people
- known for high quality work
- known as an easy-going personality

Suppose your manager gives you a junior programmer right out of a training program, an analyst who does not program and who has no operational environment, database, or managerial experience, and a senior programmer who does no design, is known to be difficult, and sometimes does high quality work.

The good news is that you have three people instead of two. The bad news is no one of them has all of the qualifications you want. What do you do? This is what management is all about.

The project manager should get to know the team members well. This means assessing their position with the company, expectations on the project, specific role desired for the person, possible start and end dates for work, and personality or personal issues that might affect their work. Much of this information can be got from previous performance reviews. But nothing substitutes for discussing the information with the person.

The project manager has responsibilities to his or her manager, the client sponsor, and to the rest of the project team to get the best, most qualified people possible. In these capacities, the project manager honestly discusses previous problems with the person, any personal problems that might detract the person's attention from work, and any outside jobs, school, or other commitments that might also hinder their commitment. The person and the project manager both should be given an opportunity to accept or reject the possibility of work. Even when there is no choice, it is also the responsibility of the project manager to make his or her expectations of quality and quantity of work clear. If the person will not report directly to the project manager, the person she or he will report to should also be at the meeting. In this way, everyone knows exactly what was said and what commitments were (or were not) made.

The answer to the task assignment problem above is to assign the tasks to best fit the skills. Assign the senior person responsibility for the work of the junior one, and provide motivation and incentives for quality work (see the following section on motivation). You also alter the schedule, if needed, to more closely mirror the actual skills of the team.

The heuristics, or rules of thumb, for personnel assignment are as follows:

1. Assign the best people to the most complex tasks from the critical path. Assign all critical path tasks. As the experience and skill levels of people decrease, assign less complex and smaller tasks. Do not give new, junior, or unqualified staff any tasks on the critical path. Assignment of senior people to critical tasks minimizes the risk of missing the target date.
2. Define a sequence of work for each person to stay on the project for as long as their skill set is needed. Try to assign tasks that provide each person some skill development.
3. Do not overcommit any person by assigning more tasks than they have time. Make sure each person will be busy, but allow time to finish one task before beginning another.
4. Allow some idle time (2–5%) as a contingency for each person. Do not allow more than eight sequential hours (i.e., one day) of idle time for any person.
5. Do not schedule any overtime. Scheduled overtime places unfair stress on people's professional and personal commitment and is a regular enough occurrence in development that it should not be scheduled at the outset.

The project manager is also responsible for coordinating movement from another assignment to the

current development project. This coordination is done with the other project manager(s) involved and possibly the personnel department. New hires should be assigned a 'buddy' to help them get familiar with the company, its facilities, the computer environment, policies, and procedures. Senior staff should be assigned to mentor junior staff, encouraging the learning of new skills on the job.

Finally, the project manager must ensure that each person understands the expectations and duties assigned to him or her. All staff should have a copy of their job description. They should know the extent of their user interaction, extent of their intraproj-ect responsibility and communication, and policies about chain-of-command on who to go to with problems, project errors found, or problems with work assignments.

Ideally, the team should be given an overview of the application, a chance to review the schedule, and an opportunity to comment on their ability to meet the deadlines assigned. If they cannot meet the deadlines and have reasonable explanations, the plan, schedule, and budget should be changed. In addition, any training or learning on-the-job that is required should result in a lengthening of the schedule. If the team members agree to the schedule, then they are committed to getting the work done within the time allowed and should be held accountable for that as part of their work assignment.

Selecting from Among Different Alternatives

Applications all have alternatives for implementation strategy, methodology, life cycle, and implementation environment. The project manager and SE together sort out the options, develop pros and cons, and decide the best strategies for the application.

Implementation Strategy

Implementation strategy is some mix of batch, on-line, and real-time programming. The decision is based on timing requirements of users for data accuracy, volume of transactions each day, and number of people working on the application at any one time. All of these numbers are estimates at the planning stage of an application, and are subject to change. The strategy decision might also change. In general, though, a decision can be made at the feasibility stage to provide some direction for data gathering.

As Table 3-2 shows, the timing of data accuracy drives the decision between batch and on-line. Keep

TABLE 3-2 Decision Table for Implementation Strategy Selection

Timing of Data Currency							
< 1 hour	N	N	N	N	Y	Y	Y
< 4 hours	N	Y	Y	Y	—	—	—
< 24 hours	Y	—	—	—	—	—	—
Peak Transaction Volume/Number of People Entering Data							
< 10	—	Y	—	—	Y	—	—
10–59	—	N	Y	—	N	Y	—
> 59	—	N	N	Y	N	N	Y
Options							
Batch application	X	X					
On-line application		X	X	X	X	X	
Real-time application			X	X		X	X

in mind that these are rules of thumb and need to be used in an organizational context. If data can be accurate as of some prior period, a batch application might be developed. If data must be accurate as of some time of the business day, either on-line or real-time strategies would be successful.

If the volume of transactions divided by the number of people is very high (over 60 per minute), then a high-performance application, with many concurrent processes, that is, a real-time application, might be warranted.

If the volume of transactions divided by the number of people is low (less than 25 per minute), but the timing requires on-line processing, an on-line application is best.

The gap in transactions per minute from 10 to 60 requires more information, specific to the project, for a decision. Answers to several questions are needed. For instance, how complex is a transaction? How was the number of workers arrived at, and can the number change? Is management willing to fund the difference in cost for a real-time application over an on-line one? Are there other factors (e.g., specific database software to be used) to consider in the decision? These questions are all context specific and the resulting decision would be determined by their answers.

Implementation Environment

The **implementation environment** includes the hardware, language, software, and computer-aided support tools to be used in developing and deploying the application. The decision is not final at the feasibility and planning stage, rather the alternatives and a potential decision are identified. The issues to be resolved for a final decision are then identified.

Frequently there is no choice of implementation environment. The organization has one environment and there are no alternatives; all development uses one mainframe and one language (for instance, COBOL). More often, as personal computers and local area networks become more prevalent, the alternatives are mainframe or network with PCs as the workstation in the chosen environment.

The decision is based frequently on the experience of the project manager, SE, and potential team members. People tend to use what they know and not use what they do not know. Ideally, the implementation environment should be selected *to fit the application, not the skills of the developers.*

TABLE 3-3 Decision Table for Implementation Environment

CPU Bound	N	N	Y	Y
I/O Bound	Y	Y	N	N
< 100,000 Trans/ Day	Y	—	Y	—
> 100,000 Trans/ Day	—	Y	—	Y
Hardware				
Mainframe	X	X	X	X
LAN	X			
LAN + Mainframe network	X	X	X	

For instance, if a real-time application is being built for a Sun workstation environment under Unix operating system, C++ or Ada are probably the languages of choice. Certainly, Cobol is not a choice.

Guidance in implementation environment selection comes from the user. Do they have equipment they want to use? How is it configured? What other software or applications are on the equipment? How amenable is the user to changing the configuration to fit the new application?

Then, with this information, the decision table in Table 3-3 can be used as a guideline for selecting the implementation environment.

In general, whenever there is a specific requirement, it tends to drive the remaining decisions. Whenever there are general requirements, the decision can remain open for a longer time. Some direction—either toward a mainframe solution or a PC/LAN solution—should be tentatively decided during feasibility and planning. During this process, the project manager should identify the issues for further information needed in making a final decision.

Methodology and Project Life Cycle

The final issue to be tentatively decided is which methodology and how streamlined the life cycle

TABLE 3-4 Decision Table for Methodology and Life Cycle Selection

Source of Complexity								
Process	Y	—	—	—	Y	—	—	—
Data	—	Y	—	—	—	Y	—	—
Knowledge representation	—	—	Y	—	—	—	Y	—
Balanced	—	—	—	Y	—	—	—	Y
Novel problem	N	N	N	N	Y	Y	Y	Y
Methodology								
Process	X				X			
Data		X		X		X		X
Object	X			X	X	X	X	X
Semantic			X				X	

will be. Frequently, there is no choice about these decisions, either. The organization supports one methodology and one life cycle and there is no discussion allowed. Equally frequently, enlightened managers know that not all projects are the same, therefore the development of the projects should also not be the same.

Methodology choices are process, data, object, social, semantic, or some hybrid of them (see Chapter 1). Life cycle choices are the sequential waterfall, iterative prototyping, or learn-as-you-go (see Chapter 1). These decisions are *not* completely separated from those of implementation environment in the previous section, because any fixed implementation requirements can alter both the methodology and the life cycle choices.

Assuming no special implementation requirements, the application itself should be the basis for deciding the methodology. In a business environment, the rule of thumb is to choose the methodology that addresses the complexity of the application best. If the complexity is procedural, a process method is best. If the complexity is data related, a data methodology is best. If the problem is easily broken into a series of small problems, an object method might work best. If the project is to automate expert behavior or includes reasoning, a semantic methodology is best. A decision table summarizing heuristics on deciding methodology and life cycle is shown as Table 3-4.

Life cycle choice also requires some decision about what type and how much involvement there is of users. If some intensive, accelerated requirements or analysis technique is used [see joint requirements planning (JRP) and joint application design (JAD), Part II Introduction], either a streamlined sequential life cycle or an iterative approach can be used. Very large, complex applications with known requirements usually follow a sequential waterfall life cycle. If some portion of the application—requirements, software, language—is new and untested, prototyping should be used. Object orientation assumes prototyping and iteration. If the problem is a unique, one-of problem that has never been automated before, either a learn-as-you-go prototyping or an iterative life cycle would be appropriate.

In the next sections, the activities for which the project manager has sole responsibility are detailed.

These activities include liaison, personnel management, and project monitoring and reporting.

LIAISON

The project manager is a buffer between the technical staff and outside organizations. In this liaison role, the project manager communicates and negotiates with agents who are not part of the project team. A **liaison** is a person who provides communications between two departments. Examples of outside agents include the project sponsor (who may or may not be the user), IS managers, vendors, operations managers, other project managers, and other departments such as quality assurance (for validation and testing), law (for contracts), and administration (for clerical and secretarial support).

For each type of liaison, status reports are an important means of communication (see sample in Figure 3-6). Status reports document progress, identify problems and their resolution, and identify changes of plans to all interested parties. In addition, many other communications of different types are described for each type of liaison. The guidelines here are just that—guidelines. They are developed assuming that open communications between concerned parties is desired, but the guidelines require judgment and knowledge of the situation to separate a good action from a less good action.

Project Sponsor

The **sponsor** pays for the project and acts as its champion. A **champion** is one who actively supports and sells the goals of the application to others in the organization. A champion is the 'cheerleader' for the project.

The goals of liaison with the champion are to ensure that he or she knows the status of the project, understands and knows his or her role in dealing with politics relating to the project, and knows the major problems still requiring resolution.

The major duty of the champion is to deal with the political issues surrounding the project that the project manager cannot deal with. Politics are in every organization, and politics relate to organizational power. Power usually is defined as the ability of a person to influence some outcome. One source of power comes from controlling organizational resources, including money, people, information, manufacturing resources, or computer resources.

Political issues of application development do not relate to the project, but to what the project *represents*. Applications represent change. Changes can be to the organization, reporting structure, work flow, information flow, access to data, and extent of organizational understanding of its user constituency. When changes such as these occur, someone's status changes. When status changes, the people who perceive their status as decreasing will rebel.

The rebellion may be in the form of lies told to analysts, refusal to work with project members, complaints about the competence of the project team, or any number of ways that hinder the change. If the person causing trouble is successful, the project will fail and his or her status will, at worst, be unchanged. Politics, left unattended, will lower the chances of meeting the scheduled delivery date and raise the risk of implementing incorrect requirements. The project manager usually tries to deal with the political issues first, keeping the sponsor informed of the situation. If unsuccessful, the sponsor becomes involved to resolve the problem.

In some organizations, the project manager communicates to the sponsor only through his or her manager. In others, the project manager handles all project communications. In general, treat the sponsor like your boss. Tell him or her anything that will cause a problem, anything they should know, and anything that will cause the project delays.

User

The **user** is the person(s) responsible for providing the detailed information about procedures, processes, and data that are required during the analysis of the application. They also work with the SE and project manager in performing the feasibility analysis, developing the financial and organizational assessments of user departments for the feasibility study.

ICIA Industries—Interoffice Memo

DATE: October 10, 1994

TO: Ms. S. A. Cameron

FROM: J. B. Berns

SUBJECT: Order Entry and Inventory Control Project Status

Progress

We have resolved the testing problems between batch and on-line by going to a two-shift programming environment. The on-line programmers are working from 6 A.M. to 2 P.M. and the batch programmers are working from 2 P.M. to 10 P.M. This is not an ideal situation, but it is working at the moment.

We are still two weeks behind the schedule for programming progress, and we may not be able to make up the time, but we should not lose any more time.

The on-line screen navigation test began two days ago and is going smoothly. Several minor spelling problems have been found, but no logic problems have been found. George Lucas should complete the user acceptance of the screen navigation and screen designs within three days if no other problems surface.

Problems

The decode table for warehouse location, due 5/12/94 from George Lucas, is still not delivered yet. This is going to delay testing of the on-line inventory allocation programs beginning in ten days if we do not have it. Is there another person we can contact to get this information?

Operations found what appears to be a bug in one of the CICS modules. When a screen call is made, two bytes of the information are lost. We are double-checking all modules to ensure that it is not an application problem. Jim Connelly is calling IBM today to see if they have a fix for the problem. At the moment, this is not causing any delays to testing. But it will cause delays beginning next week if the problem is not resolved. The delays will be to all on-line modules calling screens and will amount to the time per module to code a workaround for the unresolved problem. This should be about one hour each for a total of 120 hours. We hope this delay can be avoided; everyone possible is working on the problem, including two experts from our company whom we called in last night as a free service to ICIA.

FIGURE 3-6 Sample Status Memo and Report

Project manager–user communication includes both planned and unplanned status meetings, written communications for status, analysis, interview results, documentation, and walk-throughs of application requirements as specified by the project team. Timing of user communications differs with the type of communication, but is most often daily until the application begins programming and testing. Then, a minimum of weekly personal contact should maintain the relationship.

In general, tell the user everything that might affect them, the project, or the schedule negatively; do not tell them anything else.

IS Management

IS managers, like most managers, want to know progress, problems *and* their solutions, warnings of lateness, and political issues. They do not want to handle all problems for their managers, nor do they appreciate finding out a project will be late the week before it is due. Tell your manager anything that might get him or her in trouble, that they need to know, or that might impact the project negatively. Always expect to propose solutions and argue if you think your solution is better than their's. Always accept their solution if it is mandated, unless it is unethical or illegal.

Technical Staff

Technical staff here means the project team. Always be open with them. Keep them current on progress, problems and resolutions, and any information that affects their ability to do their job. Praise quality work. Practice team building using common sense, like having small victory parties at the end of phases, sharing birthdays, or announcing promotions.

Operations

Operations affect the project differently depending on the phase. In early phases, word processing and PCs must be available for documentation. Computer-aided software engineering tool access might be required. Timing, type, and needs of access should be planned and negotiated well in advance. The kinds of problems a team might suffer from no access may delay documentation but does not delay the work of analysis. In the worst case, the work can be done manually.

During design, the database administrator must have access and resources allocated for the definition and population of a test database. This must also be negotiated well in advance.

During implementation, old data must be converted to the new format and environment, programs must be placed in production, and users begin using the application. At this time, the operations department assumes responsibility for running the application. This responsibility must also be planned and negotiated in advance.

When programming and testing begin, all project members need access to compilers, test database, editors, and, possibly, testing tools to work on their programs. Absence of resources at this time can severely delay project completion. For each day of person-time lost, there can be one day of project delivery time lost. Timing, type, and volume of access are all negotiated items. Advance negotiation should begin at least one month prior to the need. Most operations managers will tell you they want to know about a demand for their resources as soon as you can identify the demand and the date needed. Most operations managers will also tell you they want all requirements at once. So you should be prepared to discuss analysis, design, and implementation needs before much work takes place.

In general, operations managers need to know what the project needs from them and when. They also should be sent progress reports and told of any problems that affect the use of their resources.

Vendors

A **vendor** is any company, not your own, from which you obtain hardware, software, services, or information. If the application is installed in an existing environment, probably no vendor contacts are needed. If, however, acquisition of software, hardware, or both is planned, there are three types of contact with the vendor that take place. The first is proposal communication, the second is for negotiations, and the last is customer support.

A **Request for Proposal** (RFP) (see Chapter 16) is a document developed by the PM and SE to solicit bids from potential vendors. Vendors are asked to respond with an estimate of service and price within some number of days (e.g., 30). All bids received by the cut-off date are reviewed. Proposal communications are usually limited to information about the proposal. RFPs are accepted and responded to by vendor marketing staff with some technical assistance. Project manager contact is with the marketer.

Part of the RFP process is the development of a list of required features for the item being bid upon. This list should have priorities and weights assigned to it during the proposal stage for use during the analysis. Bids are rated on the requirements then compared to see which vendor most closely meets the needs of the application.

When a vendor is selected, a contract must be negotiated. Negotiation may be with the marketer, but might also be with a financial person or with the marketer's manager. Similarly, the project manager might do all or some of the negotiation with assistance from a financial person or his or her manager. Negotiations deal with price, time period of the contract, number of sites, number of users, type of license, guarantees in case the vendor goes out of business, warrantees, and so on. There is no one way to negotiate, and most often, all negotiations are turned over to legal staff for completion of contract terms. It is important never to commit to any terms until they are seen and approved by some manager in the organization. Frequently, contracts have far-reaching implications that an individual project manager may not know.

Other Project Teams and Departments

Other IS organizations that might need project communications include a database administration group, other project teams, and a quality assurance group. Other departments might include law, or audit. In all cases, the communication is similar. These groups need to know what their relationship to your project is, how soon and what type of support you need, who to contact for questions and information, and project status that might change any of these requirements.

In addition, you also have needs of these teams. If any of the organizations is performing work you need to complete your project, then you need the same things from them that they need from you. You need to know exactly what they will do for you and how it will be transmitted to your project, whom to contact, and task status that might affect your schedule.

To summarize, many other groups and departments in the organization need to have liaison activities with a project. It is the project manager's job to provide that liaison with communications tailored to the needs of the other organization.

PERSONNEL MANAGEMENT

For **personnel management**, the project manager hires, fires, coaches, motivates, plans, trains, and evaluates project team members.

Hiring

Hiring is usually coordinated through a personnel office that oversees all IS hiring, not just one project. Newspaper advertisements can be more cost-effective, general, and get a better response when coordinated for all projects. The personnel office receives the responses and filters obviously unqualified applications out from the pool of applicants. Then, working with the project manager, the personnel department screens the applicants and arranges project interviews.

As in most things, timing is important. Ads take from one to two weeks to get approved and placed. Receipt of resumes usually takes the same amount of time. Interviewing is time consuming and can take another one to two weeks for each hire. Then, offers are made and salary negotiations completed. The elapsed time to hire someone might be seven weeks or longer.

In addition, scheduling interviews may mean early-morning, evening, or lunch-time work. People searching for a job who already have one may not

want to take vacation time for an interview. If the person appears qualified, the project manager is expected to shift his or her schedule to fit the needs of the applicant.

Firing

You may not agree, but keeping a person in a job for which they are unsuited does more damage to the manager, the person, and the project than you might think. Project managers are damaged because they think of little else and agonize over the decision much longer than necessary. People usually know if they are going to be terminated because they did not complete their specified tasks. They should have been told, in writing, before the termination date.

Prolonging a termination is damaging to the person being fired because it gives them a false sense of hope, makes them lose confidence in the person not following through on their described actions, and also allows them to influence other project members negatively.

Finally, procrastination on firing is damaging to the project because the longer the termination is delayed, the more likely the person being terminated will begin talking of his or her situation to other project members and disrupting work. As more people find out, more time is spent speculating on the situation. Less work gets done and the staff eventually loses confidence in the project manager.

No one gets into trouble overnight. Usually there is a period during which a problem is known, but it might be corrected before any real problems arise. It is at this time that the project manager should sit down with the person and talk about the situation. Legally, everyone in this situation is entitled to at least one warning letter which is also placed in their personnel file. This is followed by a letter of reprimand stating that performance is substandard with reasons for that judgment. The letter also states that the person is on probation and will be terminated by a specified date unless some actions are taken. The actions are then listed. If the person does the assigned work satisfactorily, they are off probation. All of these communications are in writing, monitored and approved by personnel and the IS manager, and are the basis for any future legal action by the employee.

If the work is performed satisfactorily, probation ends. If not, the person is terminated. Termination from a project does not necessarily require termination from a company. If a person is ill-suited to a particular project, she or he might still be a valuable employee. A good project manager will first try to place the person somewhere else in the organization. If the person is terminated from the company, the company can try to help them find another job through an out-placement service or by providing company resources (a desk and phone away from the project) until a job is found. If the person is terminated for antisocial behavior, an addiction, or for some other nontechnical problem, the project manager might help them seek professional help.

Motivating

Motivation has personal and professional aspects. Professional motivation arises from a desire to do a good job. People are motivated to do a good job when they are treated like a professional and given meaningful, interesting work that includes some discretionary decision making and some creative design.

Personal motivation arises from a desire to improve one's position in life. Position in life is defined individually and may mean earning more money, buying a bigger house, becoming an analyst, or becoming a manager, and so on.

Project management style is the determining factor of personal motivation. A project manager who facilitates participation, fosters controlled risk-taking, and allows people to grow as individuals will gain undying loyalty from his or her staff. A project manager who treats the staff as stupid, lazy, and unmotivated might obtain desired behaviors from them, but it will be through intimidation and coercion.

The project manager needs to know the project team members individually in order to tailor reward systems and assignments to help them reach their goals. Project manager commitment to helping team members reach personal goals determines

how professionally motivated the team members will be.

There are three aspects to motivation. First, the project work itself can be used to further professional goals that include doing novel work and advancing to new levels of seniority, experience, or responsibility. Second, the project manager must be careful to tailor reward and punishment systems to fit the tasks, being unbiased in terms of importance of individual contributions to the work. Third, the individual professional must make a commitment to doing something extra to gain the reward, either on-the-job or on his or her own time.

Take, for instance, a mainframe Cobol programmer who wants to move to a personal computer LAN environment using C++. The project has relaxed deadlines and the project manager might be able to help the person, but some commitment from the programmer is needed. The project manager recommends that the person find, attend, and pass a C++ course for which the company will pay. Then, the person will be assigned a task in the desired environment. If the task is successful, more tasks will follow. If the task is not successful, the situation will be reassessed.

Professional motivation might also come from fostering development of association ties outside of work. Meetings or user groups of vendors,[1] professional associations,[2] or other professional groups related to work duties might be paid for by the company to foster professional motivation.

Motivation also has a negative side. The actions that would be taken should the person fail to do their job competently must also be known. There should be company policies about quality and quantity of work that are also included as part of job descriptions. In the absence of company policy, the project manager should adopt rules, with the knowledge and consent of their manager, about punishments for failure to meet work requirements. These should also be made known to everyone on the project.

Career Path Planning

Motivating is an immediate activity of the project manager, but all employees and managers should be encouraged to develop longer range aspirations, as well. The project manager should help plan, with each individual, the tasks from this project that can be used to further his or her career.

The project manager should discuss goals and career paths at the beginning of the project and at least annually during performance reviews after that. The discussion should include a frank assessment of current perceptions of the individual's verbal, organizational, and professional skills, as well as helping the person plan courses, assignments, or opportunities to improve his or her performance. There should be direct ties from performance to rewards. Any time an individual does something significant enough to be mentioned on an appraisal, he or she should be told and either praised or counseled to change.

Training

The purpose of project training is to specifically address weaknesses of staff in techniques, technology, or tools used on the project. The SE and any project leaders are directly responsible for identifying training needs. The project manager is responsible for obtaining the training for the individual(s) who need it. A senior mentor for the trained skill should be assigned to monitor progress in the development of the skill, once training is complete.

Nonrelated training, as discussed above, may also be authorized by the project manager depending on employee need, rewards, and fit with employee goals.

Evaluating

Evaluations are annual assessments of the person from both professional and organizational perspectives. Evaluations are written and usually are signed

1 Guide and Share are IBM mainframe user groups with over 10,000 members each. DECus is the Digital Equipment users group. In these huge groups, there are subgroups with interests in every software package, language, and development environment offered by the vendor.

2 The Association for Computing Machinery (ACM) is one example.

by the reviewed person and the reviewer. Quality and quantity of work assignment are the professional assessments and are the most important aspects of junior level work. Junior staff, having no business experience, are monitored most closely for their ability to do their work. Competence for the assigned jobs is determined, and the more competent, the faster the person is promoted.

As people become more senior, quality and quantity of assigned work becomes assumed and organizing, motivating, communications, and interpersonal skills become more important. The nontask specific skills are viewed from an organizational perspective. More emphasis is placed on the ability to persuade, manage, motivate, and communicate with others, thus describing a good manager.

Promotion for most senior people is to the managerial ranks. In some companies, the importance of very senior, technical experts, is recognized. In those companies, equal emphasis is placed on the professional and organizational assessments. Technical staff can aspire to the senior technical positions without having to sacrifice their technical expertise in the bargain.

The usual performance evaluation contains sections for assignments, communications and interpersonal relations, absences, planning and organization, supervision, delegation, motivation, training, and special considerations. Each of these is described briefly.

The assignments section contains a brief description of four or five major assignments with expectations on quality and quantity of work for each as well as a brief paragraph assessing the extent to which the assignment was met. Quality and quantity of work are intangible and frequently subjective assessments, but there are always expectations of the amount of work a person should do, and of the extent to which reworking is needed. In addition, the individual's job description should give guidance on expectations for work quality and quantity. Finally, the extent to which the person needs to be monitored and assisted is an indicator of the extent to which they can work independently and competently at their job. The discussion of quality and quantity should be presented in terms of job description, manager expectations, and extent to which expectations are met. Specific examples are required to demonstrate very high and very low quality work.

Project managers evaluate communications and human relations. Assessments of both relating verbal and written communication skills are developed. Communication skills are related to specific project assignments and to other project activities, such as walk-throughs, that are not major assignments. Communication evaluation includes grammar, speed, persuasiveness, clarity, and brevity. The person's ability to develop and deliver a presentation, and actual experiences doing these are described.

Another area of assessment is interpersonal relationships with project manager, senior staff members, peers, others in the department, and users. Additional comments might discuss specific incidents that vary from the general assessment and that might highlight a need for improvement, or identify a particular skill. For instance, a person with good negotiating skills might be identified by their arbitration of a disagreement between two other project members.

Work absences are mentioned in terms of total days missed, number of absences, and type of absence. If there are company policies about absences and they are exceeded, a comment about the extent to which absences affected work might be added. The ability of the person to meet deadlines, maintain an accurate status of the project, and need special communications due to absences are all described. Extraordinary situations causing a long absence, such as emergency surgery, are included.

For planning and organization, accuracy, detail, independence of work, and cooperation with other affected groups are all assessed. In addition, the person's adherence to their own plans is discussed. Do they use it properly as a road map, or is it a rigid rule from which no straying is allowed, or is it ignored and treated as a task done for management?

Delegation is the extent to which the work is shifted from the manager to subordinates. Issues rated are how well work assignments match people's skills, allow monitoring to ensure completion, provide for personal and career improvement of subordinates.

Managerial style is assessed in terms of group motivation. Does the project manager obtain

commitment from staff with enthusiasm, discomfort, unhappiness, or anger? Does the manager ask or command? How successful is the strategy and what must the manager do to change unsuccessful strategies? Are tactics altered to fit the person being managed, or is everyone treated the same way? Are people treated fairly or is favoritism prevalent?

Can the manager motivate others to learn new skills? To what extent does the manager provide needy staff with training, either formal or informal, on techniques, technology, and tools? If formal training is given by the person being rated, summaries of student ratings of quality and quantity of training should be presented. The person's ability to provide mentoring and quality of mentoring might be addressed.

Finally, there is usually a section for the project manager to recommend future assignments, training, or other professional activities for further development of the individual.

MONITOR AND CONTROL

Status Monitoring and Reporting

The rationale of the planned application development is that you monitor the plan to communicate activity status and interim checkpoints to clients. The overall goal—meeting the project installation date—is the end point of a lengthy complex set of processes. Without the plan, knowing whether or not the installation date will be met is difficult. **Status monitoring** is the comparison of planned and actual work to identify problems. **Project control** is the decisions and actions taken based on the project's status.

In a planned approach, project team members report time spent on each activity for some period. The sample time sheet (see Figure 3-7), allows breakdowns for several tasks listed across the top of the form and hours worked on the task reported by day of the month. Totals by day of the month and by task over the period are tallied by row and column totals. This type of reporting allows the project manager to easily see for each person weekend work, how many hours are spent on each activity over a period, and how many effective work hours there are per day.

In addition, each person should write a short progress report. The report summarizes progress in qualitative terms, identifies problems, issues, errors, or other conflicts that might delay the work. If a task will be later than its schedule date, the reason for lateness must be explained. The project manager and SE both review the reports and time sheets to decide if problems need further action. A sample progress memo is shown as Figure 3-8.

The SE and project manager map actual progress of each person against the planned times. When progress looks slow, the project manager asks the person specifically if there are problems, if there are enough resources, for example, test shots, and if the person thinks they can meet the deadline. If the task appears to have been underestimated, the schedule is checked to see if changing the time allotted will cause completion delays. Similar tasks are checked to see if they are also underestimated. The cumulative effect of changes is checked to see if completion is in jeopardy. If it is, the project manager discusses the problem with his or her manager and they decide on the proper course of action.

The best policy is to address potential problems early, before they become big problems. If a person cannot finish work because of too many assignments, then reassign some of the work to another person. If they have not got enough testing time, arrange for more time. Active management prevents many problems.

Problem follow-up includes determining the severity and impact, planning an alternative course of action, modifying the plan as required, and continuing to monitor the problem until it is resolved or no longer has an impact on the delivery date.

Tell the client about problems that may not be solved so they are prepared for delays if they become inevitable. When changes become needed, tell the client about changes to planned dates even when they do not change the completion date.

Project: _____ Month: _____

Name: _____

Activities

Day of Month							Total for Day
1/16							
2/17							
3/18							
4/19							
5/20							
6/21							
7/22							
8/23							
9/24							
10/25							
11/26							
12/27							
13/28							
14/29							
15/30							
31							
Total							

FIGURE 3-7 Time Sheet

ICIA Industries—Interoffice Memo

DATE: October 10, 1994

TO: J. B. Berns

FROM: M. Vogt

SUBJECT: Order Entry and Status

Progress

We completed our screen design and navigation testing 10/7/94 and turned the modules over to George Lucas for user acceptance. He requested changes to several items:

1. The location of the total at the bottom of the screen is moved left five spaces.
2. The PF key assignment for PF3, which we were using to END any process. He would like END to be PF24. We explained that this is not a good design because the operator needs more key strokes (and hence is more likely to err) for PF24. Also, this is a very time-consuming change, about 10 hours, and that he should have mentioned his preference during the reviews. He decided to think about it and talk to some real operators before making a firm decision.

The other testing is progressing well. I am almost done testing the entire order process, except for inventory allocation. I need the warehouse codes from George by next week if I am to continue testing the programs.

Problems

The warehouse codes which were promised some months ago are getting to be on the critical path. If I do not have them by next week, I cannot continue to test the inventory allocation portion of the application. I can assign my own code scheme, then change it to the real one if I have to, but I would like to avoid the double work.

FIGURE 3-8 Sample Progress Report

The kinds of problems that occur and the activities the project manager monitors change over the course of the development. For instance, during the definition of the project scope, the project manager monitors the following:

Is the client cooperative?
Are all the stockholders identified and involved?
Are users being interviewed giving accurate, complete information?
Are users participating as expected?
Are there any apparent political issues to be addressed?
Does the scope look right? That is, does the current definition appear to include relevant activities?

By analysis, the project manager knows most users and how they work, should have identified potential political problems and dealt with them, and should be comfortable that the project scope is correct. The activities monitored turn toward the project team, and include the following:

Do all analysts know the scope of activity and work within it?
Is the analysts' work emphasis on what and not how?
Are users participating as expected?
Are all project members pulling their weight?
Is everyone interested and happy in their job?
Is there any friction between team members, or between team members and users?
Does everyone know what they and all others are doing?
Is there constant feedback-correction with users on interview results?
Are team members beginning to understand the users' business and situation? Are the team members objective and not trying to force their own ideas on the users?
Are walk-throughs finding errors and are they getting resolved?
Are documents created looking complete? Does the user agree?
Is the analysis accurately addressing the problems of the user? Are team members analyzing and describing exactly what is needed without embellishment?
Is typing turnaround, printing of word-processed documents, copying, or other clerical support acceptable?
Does communication between teams and between teams and users appear to be satisfactory?
Is the project on time? What is the status of critical path tasks? Has the critical path changed because of tasks that finished early?
Where are the biggest problems right now? How can we alleviate the problems?
What do we not know that might hurt us in design?

The functional requirements that result from analysis should describe what the application will do. The project manager is constantly vigilant that the requirements are the users. One problem many projects have is that the user wants a plain functional application but the analysts design a high-priced application with the user functions, but with many unnecessary features, or 'bells and whistles,' as well. This problem, if it occurs, must be dealt with before analysis ends or extraneous functions will be in the resulting application. When over-design problems surface, it is important to try to trace them to specific analysts for retraining in providing their services.

In design, the emphasis shifts to monitoring the rate, type, and scope of changes from the users. If the business is volatile, requirements change may become a constant problem. Change management procedures should be developed and used. At this point, the project manager's worries include the following:

Do the analysts know the application?
Is the translation to operational environment correct and complete?
Are walk-throughs finding errors? Are errors being resolved?
Are users participating as expected? Are users properly involved with screen design, test design, acceptance criteria definition?
Are all project members pulling their weight?
Is everyone interested and happy in their job? Is there any friction between team members, or between team members and users?
Does everyone know what they and all others are doing?
Are all team members aware of their changing responsibilities, and are they comfortable with and able to do design tasks?
Does communication between teams and between teams and users appear to be satisfactory?
Is the project on time? What is the status of critical path tasks? Has the critical path changed because of tasks that finished early?
Where are the biggest problems right now? How can we alleviate the problems?
What do we not know that might hurt us in programming? Is the implementation environment suitable for this application?

Can the database management software accommodate this application?

The number of project team members usually increases for programming to do parallel development as much as possible. The communication overhead necessary to know everyone's status and for them to know the project status increases. The problems in the programming and unit testing stage tend to focus on communications and programmer performance.

Does everyone understand how their work fits into the project? Does everyone know their critical-path status? Are all current project members pulling their weight? Does everyone know what they and all others are doing?

Is testing time sufficient? Is terminal access sufficient?

Does everyone know the technologies they are using sufficiently to perform independently?

Are junior staff paired with senior mentors?

Are users requesting further changes?

Are users participating as expected in test design, user documentation development, conversion, and training?

Is there constant feedback-correction with users on suspected errors?

Are prototypes being used as much as possible to demonstrate how the application will work?

Are walk-throughs productive, finding errors? Are errors getting resolved?

While programming and unit testing are proceeding, tests for integration and system level concerns are being developed. The database is being established and checked out. The operational environment is being prepared. Concern shifts from getting the application expressed in code to getting it working correctly. The kinds of questions a project manager might have are the following:

Are all current project members pulling their weight? Does everyone know what they and all others are doing?

Is testing time sufficient? Is terminal access sufficient?

Are users requesting further changes? Are users participating as expected in testing?

Is there constant feedback-correction with users on suspected errors?

Are walk-throughs productive, finding errors? Are errors getting resolved?

Does the system level test really prove that the functions are all accounted for?

Does the integration test verify all interconnections? How can it be leveraged to prove the reliability of the interconnections during the system test?

What do we not know about the operational environment that might hurt the project?

Is the database software working properly? Are back-up and recovery procedures adequate for testing?

How can we use the integration and system tests to develop a regression test package?

Is documentation being finalized? Is everyone working to capacity? Should we start letting programmers go to other projects? If we let a key person go, who can take their place when a problem occurs?

Finally, testing is complete, the application appears ready, and the user is ready to work. There should have been a plan for actually implementing the operational application that eases the user into use without too much trauma. The easing-in period gives the project team some time to fix errors found in production without excessive pressure. The issues now center on getting the application to work in its intended environment for its intended users. The questions include the following:

Is the site prepared adequately? Is air conditioning sufficient? Are lighting and ergonomic design sufficient?

Are users properly trained and ready to do work?

Are work cycles and evaluation of results identified sufficiently to allow implementation and verification of results?

When errors are found, are they getting resolved?

Are users taking charge as expected?

Are all current project members pulling their weight? Does everyone have enough work to do? Can people be freed to other projects?

Is communication between teams and between teams and users appearing satisfactory? Are users told whenever major problems occur? Are they participating in the decision making about error resolution?

Many of the questions above are technical in nature and would be referred to the SE to monitor. The project manager is like a mother hen and is supposed to worry about everything. Obviously, if the plan addresses the activities as it should, many of the answers to the above sets of questions are found in weekly progress reports of team members. Compiling the individual progress reports and project progress reports in a project log allows the manager and any of the staff to review decisions, problems and their resolutions, and other issues as they occur during the development.

AUTOMATED SUPPORT TOOLS FOR PROJECT MANAGEMENT

Project management support tools have increased in sophistication and performance since the mid-1980s when the first PC-based tools arrived. The tools in this section support project planning, task assignment and monitoring, estimation tools, and scheduling tools (see Table 3-5). Key tool capabilities

TABLE 3-5 Automated Support Tools for Project Management

Product	Company	Technique
CA-*products*	Computer Associates International, Inc. Islandia, NY	Project planning
DataEasy Project Management	Data Easy Software Foster City, CA	Task mapping
Demi-Plan	Demi Software Ridgefield, CT	Critical path project planning and tracking
Foundation	Arthur Anderson & Co. Chicago, IL	Project management Project planning
IEW, ADW (PS/2 Version)	Knowledgeware Atlanta, GA	Project planning
Life Cycle Manager	Nastec Southfield, MI	Project planning, task assignment, tracking
Life Cycle Project Manager	American Management Systems Fairfax, VA	Project planning, task assignment, tracking
Maestro	SoftLab San Francisco, CA	Problem tracking
microGANTT	Earth Data Corp. Richmond, VA	Project planning
Milestone	Digital Marketing Corp. Walnut Creek, CA	Critical path project planning and tracking
Multi-Cam	AGS Mgmt Systems King of Prussia, PA	Project planning and tracking

(Continued on next page)

TABLE 3-5 Automated Support Tools for Project Management *(Continued)*

Product	Company	Technique
PMS II	North America MICA Inc. San Diego, CA	Project planning, task assignment, tracking Critical path PERT
Primavera Project Manager	Primavera Systems Inc. Bala Cynwyd, PA	Project planning, task assignment, tracking
Project	Microsoft Bellevue, WA	Project planning, task assignment, tracking
Project Workbench, Fast Project	Applied Business Technology NY, NY	Project planning, task assignment, tracking
System Architect	Popkin Software and Systems, Inc. NY, NY	Project planning
Teamwork	Cadre Technologies Inc. Providence, RI	Planned completion date tracking
vsDesigner	Visual Software, Inc. Santa Clara, CA	Project completion tracking Critical issues monitoring

not considered here include word processing, spreadsheets, calendars, or interfaces to electronic mail (these are considered useful for all organization members). Other tools that are used by a project manager but are discussed in other sections of the text are for configuration management, quality control, and metrics.

SUMMARY

The project manager role is frequently separate and distinct from that of the software engineer. The software engineer is generally responsible for technical aspects of project work. Some tasks are joint, complementary activities shared by project managers and software engineers. For these joint activities, the software engineer contributes technical skills, and the project manager contributes organizational skills.

The project manager is solely responsible for most people-related aspects of projects. The three main tasks of the project manager are organizational liaison, employee management, and project monitoring and control. Organizational liaison includes creating working relationships with other organizations and departments, resolving project-related problems regardless of their nature, and reconciling the project design with expectations of others. Employee management includes working with Personnel to hire, fire, and staff the project. Employee management also includes individual employee monitoring to help them evaluate, set, and attain career goals. Project monitoring and control is the other major project management activity. Monitoring means to trace the progress of project work and compare it to budgeted time and resources to maintain progress. Control includes deciding and implementing project changes when progress is not satisfactory. Project changes might include change of job assignments,

introduction of training, or change to schedules, and plans.

REFERENCES

Abdel-Hamid, Tarek, and Stuart E. Madnick, *Software Project Dynamics: An Integrated Approach*. Englewood Cliffs, NJ: Prentice Hall, 1991.

Gilbreath, R. D., *Winning at Project Management: What Works, What Fails and Why*. NY: John Wiley and Sons, 1986.

Gildersleeve, Thomas R., *Data Processing Project Management*. New York: Van Nostrand Reinhold Company, 1974.

Glass, Robert L., *Software Conflict: Essays on the Art and Science of Software Engineering*. Englewood Cliffs, NJ: Prentice Hall, Yourdon Press, 1991.

Cleland, D. I., and William R. King, *Systems Analysis and Project Management*. NY: McGraw-Hill, 1983.

King, William R., and D. I. Cleland (eds.), *Project Management Handbook*, 2nd ed. NY: Van Nostrand Reinhold, 1988.

Pfeffer, Jeffrey, *Organizations and Organization Theory*. Boston: Pitman, 1982.

Rogerson, Simon, *Project Skills Handbook*. Lund, Sweden: Chartwell-Bratt, 1989.

KEY TERMS

champion
complimentary activities
critical path
evaluations
heuristic
implementation environment
implementation strategy
interface
liaison
personnel management
project control
project plan
request for proposal (RFP)
sponsor
status monitoring
task dependency diagram
user
vendor

EXERCISES

1. List and discuss three advantages and three disadvantages to project team members using time sheets to report work activities. What might some alternatives for reporting task progress and time spent be?
2. Write an honest appraisal of yourself for the work you have done in school toward your current degree. Give specific examples of good and, maybe, poor work. Rate your knowledge and skills gained in terms of a schedule that ends when you graduate.
3. Discuss the following comment: "It is important for a project manager to have been a programmer and an analyst. Otherwise, the manager has no feel for the problems and their severity."

STUDY QUESTIONS

1. Define the following terms:
   champion critical path heuristic
   liaison project plan
2. When and why are the software engineer and project manager roles split?
3. Describe the project manager's role in planning.
4. Describe a general planning methodology.
5. What kinds of reviews are done on project documentation? Why are they necessary?
6. What are five types of operations resources that might be needed on a project?
7. What is the minimum lead time recommended for resource requests?
8. What is an RFP and when is it used?
9. What is the purpose of a task dependency chart?
10. What is a critical path and why is it important?
11. Should a plan be finalized and *cast in concrete*?
12. List four types of assumptions made during planning and describe why each is important.
13. Why should project team members submit time sheets?
14. Describe how to assign staff to tasks. Why is the process rarely this simple?
15. Describe the heuristics for assigning staff to projects.
16. Should planned overtime be in a schedule?

17. List five things every person should know about his or her job when working on an application development project.
18. What are the three alternatives for implementation strategy?
19. What are the heuristics for deciding implementation strategy?
20. List two choices for implementation environment.
21. Describe the heuristics for deciding implementation environment.
22. What are the choices for methodology and life cycle?
23. Describe the heuristics for deciding methodology.
24. Describe the heuristics for deciding life cycle.
25. What is a liaison? What project manager duties require liaison work?
26. List the contents of a project status report.
27. What is politics and how does it affect application development work?
28. Why are performance appraisals done?

★ EXTRA-CREDIT QUESTIONS

1. List and discuss types of assessment from a performance appraisal. How does a manager ensure the ratings are fair and objective? What should a manager do if he or she does not like the person being reviewed?
2. Develop a project plan for ABC Video based on the information in Chapter 2 only. Use the case and this chapter to decide the tasks. Use your experience, whatever it is, to decide the times for the tasks. Do not look at other information in this or other texts when planning the work. What assumptions do you have? How comfortable are you with your estimates? Keep this assignment and redo it at the end of Chapter 6.

CHAPTER 4

DATA GATHERING FOR APPLICATION DEVELOPMENT

INTRODUCTION

Each phase of application development requires interaction between the developers and users to obtain information of interest at the time. Each phase seeks to answer broad questions about the application. For instance, in feasibility analysis, the questions are broad and general: What is the scope of the problem? What is the best way to automate? Can the company afford (not) to develop this application? Is the company able to support application development?

In analysis we seek *what* information about the application. For instance, What data are required? What processes should be performed and what are the details of their performance? What screen design should be used?

In design, we develop *how* information relating to the application. For example, How does the application translate into the specific hardware environment selected? How does the logical data design translate into a physical database design? How do the program modules fit together?

The kind of interaction that elicits answers to questions such as these differs by information type and phase. In this section we describe the alternatives for obtaining information to be used for application development. The alternative data gathering techniques are described, then related to application types. Then, ethical considerations in data collection and user relations are discussed.

DATA TYPES

Data differs on several important dimensions: time orientation, structure, completeness, ambiguity, semantics, and volume. Each of these dimensions is important in defining requirements of applications because they give guidance to the SE about how much and what type of information should be collected. Also, different data types are related to different application types and require different requirements elicitation techniques. Inattention to data dimensions will cause errors in analysis and design that are costly to fix. Error correction cost is an increasing function of the phase of development (see Table 4-1).

In addition to obtaining information, we also use the techniques for validating the information and interpretation in the proposed application. Use of validation techniques during each phase increases the likelihood that logic flaws and misinterpretations will be found early in the development.

TABLE 4-1 Cost of Error Correction by Phase of Development

Phase in Which Errors are Found	Cost Ratio to Fix the Error
Feasibility/Analysis	1
Design	3–6
Code/Unit Test	10
Development Test	14–40
Acceptance Test	30–70
Operation	40–1000

From Boehm, Barry, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

Time Orientation

Time orientation of data refers to past, present, or future requirements of a proposed application. Past data, for example, might describe how the job has changed over time, how politics have affected the task, its location in the organization, and the task. Past information is exact, complete (if maintained), and accurate. There is little guessing or uncertainty about historical records.

Current information is information about what is happening now, and its relevance in determining the future. For instance, current application information relates to operations of the company, the number of orders taken in a day, or the amount of goods produced. Current policies, procedures, business industry requirements, legal requirements, or other constraints on the task are also of interest in application development. Current information should be documented in some way that it can be read by the development team to increase their knowledge of the application and problem domains.

Future requirements relate to changes in the industry expected to take place. They are inexact and difficult to verify. Economic forecasts, sales trend projections, and business 'guru' prognostications are examples of future information. Future-oriented information might be used, for example, by managers in an executive information system (EIS).

Structure

Structure of information refers to the extent to which the information can be classified in some way. Structure can refer to function, environment, or form of data or processes. Information varies from unstructured to structured with interpretation and definition of structure left to the individual SE. The information structuring process is one in which the SE is giving a form and definition to data.

Structure is important because the wrong application will be developed without it. For instance, knowing that the user envisions the structure of the system to be one with 'no bureaucracy,' minimal user requirements, and no frills, gives you, the SE, a good sense that only required functions and data should be developed. In the absence of structuring information, technicians have a tendency to develop applications with all 'the bells and whistles' so the users can never complain that they don't have some function.

An example of structuring of data is shown in Figures 4-1 and 4-2. When you begin collecting information about employees for a personnel application, you might get information about the employees themselves, their dependents, skills the employees might have, job history information, company position history, salary history, and performance reviews.

The information comes to you in pieces that may not have an obvious structure, but you know that all of the data relates to an *employee* so there must be relationships somewhere. In Figure 4-2, we have structured the information to show how all of the information relates to an employee and each other in a hierarchic manner. Each employee has specific one-time information that applies only to them, for instance, name, address, social security number, employee ID, and so on. In addition, each employee might have zero to any number of the other types of information depending on how many other companies they have worked at, whether they have children, and how long they have worked at the company. The most complex part of the data structure is the relationship between position, salary, and reviews. If salary and performance reviews are disjoint, they would be as shown, related to a given

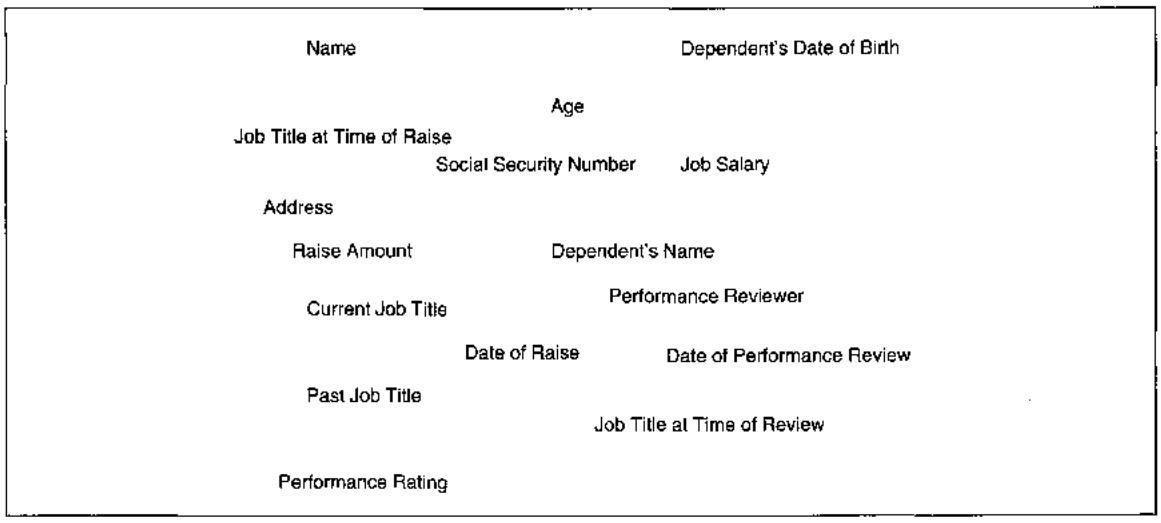


FIGURE 4-1 Unstructured Personnel Data

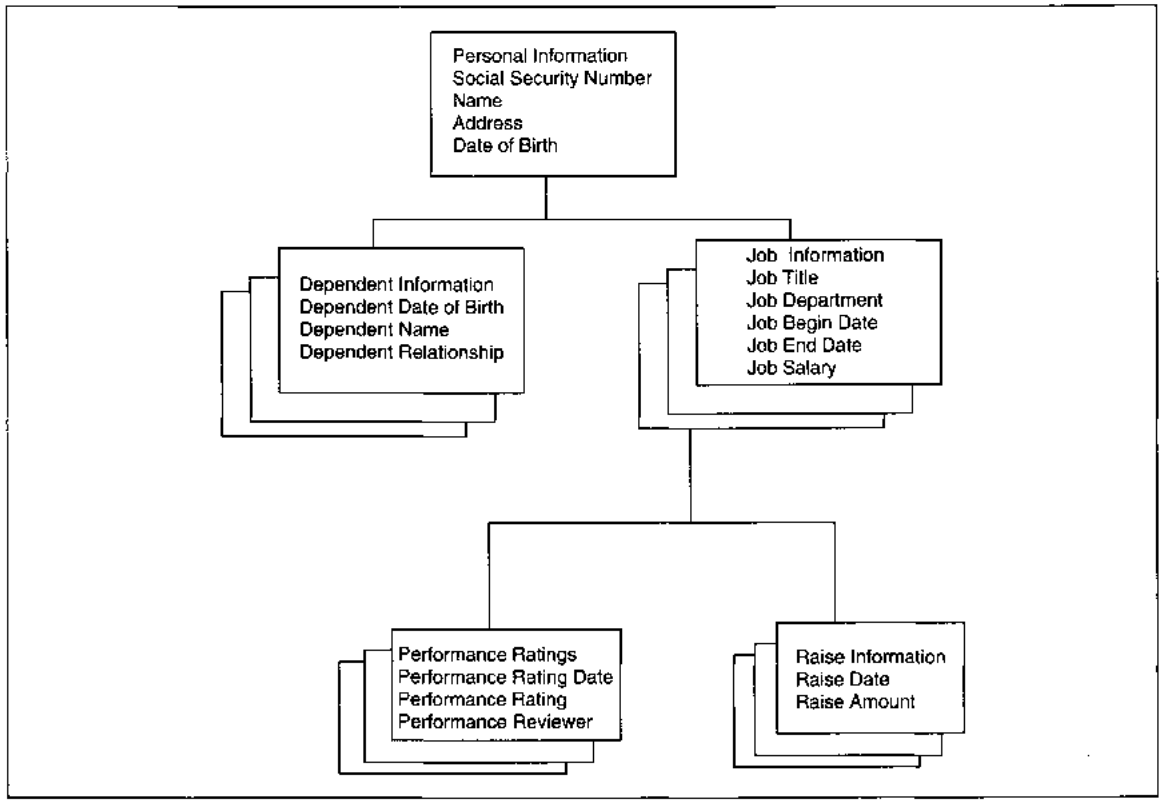


FIGURE 4-2 Structured Personnel Data

position the person held in the company (see Figure 4-2). The other option is that salary changes are dependent on performance reviews and the hierarchy would be extended another level.

Completeness

Information varies in **completeness**, the extent to which all desired information is present. Each application type has a requisite level of data completeness with which it deals. Transaction processing systems deal with complete and accurate information. GDSS and DSS deal with less complete information. EIS, expert systems, or other AI applications have the highest levels of incompleteness with which they must cope.

In applications dealing with incomplete information, the challenge to you is to decide when the information is *complete enough* to be useful. Sometimes this decision is made by the user, other times it is made within the application and there need to be rules defining *complete enough*.

Ambiguity

Ambiguity is a property of data such that it is vague in meaning or is subject to multiple meanings. Since ambiguity deals with meaning, it is closely related to semantics. An example of ambiguity is to ask the following query:

PRINT SALES FOR JULY IN NEW YORK

In this query, New York can mean New York State or New York City; both answers would be correct. Obvious problems will occur to a person who asks that request for one context (the state) and gets an answer for the other context (the city). Contextual cues help SEs to define the one correct interpretation of ambiguous items; further problems arise because of multiple semantic interpretations within a single context. For that reason, semantics is discussed next.

Semantics

Semantics is the study of development and change in the meaning of words. In business applications, **semantics** is the meaning attached to words. Meaning is a social construction; that is, the people in the organization have a collectively shared definition of how some term, policy, or action is really interpreted.

Semantics is important in applications development and in the applications themselves. If people use the same terms, but have different meanings for the terms, misunderstandings and miscommunications are assured. If embedded in an application, semantically ambiguous data can never be processed by a program without the user being aware of which 'meaning' is in the data. Applications that have semantically mixed data then rely on the training and longevity of employees for proper interpretation of the data. If these key employees leave, the ability to correctly interpret the meaning of the data is lost. Losing the meaning of information can be expensive to the company and can result in lawsuits due to improper handling of information.

An example of semantic problems can be seen in a large insurance company. The company uses the term 'institution' to refer to its major clients for retirement funds. The problem is that 'institution' means different things to different people in the company. In one meeting, specifically convened to define 'institution,' 17 definitions surfaced. The problem with semantic differences is not that 16 of the 17 definitions are wrong. The problem is that *all 17 definitions are right*, depending on the context of their use. It is the SEs job to unravel the spaghetti of such definitions to get at the real meaning of terms that are not well defined at the corporate level. Unraveling the meaning of the term 'institution' took about 20 person-months over a two-year period to get the user community to reach consensus on the *corporate* definition of the term 'institution.'

Volume

Volume is the number of business events the system must cope with in some period. The volume of new or changed customers is estimated on a monthly or annual basis whereas the volume of transactions for business operation is usually measured in volume per day or hour, and peak volume. **Peak volume** is the number of transactions or business events to be processed during the busiest period. The peak period

might be annual and last several months, as with tax preparation. The peak might be measured in seconds and minutes, for example, to meet a Federal Reserve Bank closing deadline.

Volume of data is a source of complexity because the amount of time required to process a single transaction can become critical to having adequate response time when processing large volumes. Interactive, on-line applications can be simple or extremely complex simply because of volume. For instance, the ABC rental application will actually process less than 1,000 transactions per day. Contrast this volume with a credit card validation application that might service 50,000 credit check requests per *hour*. Credit card validation is simple processing; servicing 50,000 transactions per hour is complex.

Applications that mix on-line and batch processing using software that requires the two types of processes to be distinct, requires careful attention to the amount of time necessary to accommodate the volumes for both types of processing. For instance, the personnel application at a large oil company was designed for 20 hours of on-line processing with global access, and four hours of batch reporting. When the system went 'live,' the on-line processing worked like a charm because it had been tested, retested, and overtested. The batch portion, for which individual program tests had been conducted, required about *18 hours* because of the volume of processing. After several weeks, the users were fed up because printed reports had been defined as the means of distributing query results, and they had none. The solution required an additional expenditure of over $200,000 to redevelop all reports as pseudo-on-line tasks that could run while the interactive processes were running. Simple attention to the volume of work for batch processing would have identified this problem long before it cost $200,000 to fix.

Data Collection Techniques

There are seven techniques we use for data gathering during application development. They are interviews, group meetings, observation, temporary job assignment, questionnaires, review of internal and outside documents, and review of software. Each has a use for which it is best served, and each has limitations to the amount and type of information that can be got from the technique. The technique strengths and weaknesses are summarized in Table 4-2, which is referenced throughout this section.

In general, you always want to validate the information received from any source through triangulation. **Triangulation** is obtaining the same information from multiple sources. You might ask the same question in several interviews, compare questionnaire responses to each item, or check in-house and external documents for similar information. When a discrepancy is found, you reverify it with the original and triangulated sources as much as possible. If the information is critical to the application being correctly developed, put the definitions, explanations, or other information in writing and have it approved by the users separately from the other documentation. Next, we discuss each data collection technique.

Single Interview

An **interview** is a gathering of a small number of people for a fixed period and with a specific purpose. Interviews with one or two users at a time are the most popular method of requirements elicitation. In an interview, questions are varied to obtain specific or general answers. You can get at people's feelings, motivations, and attitudes toward other departments, the management, the application, or any other entity of interest (see Table 4-2). Types of interviews are determined by the type of information desired.

Interviews should always be conducted such that both participants feel satisfied with the results. This means that there are steps that lead to good interviews, and that inattention to one or more steps is likely to result in a poor interview. The steps are summarized in Table 4-3. Meeting at the convenience of the interviewee sets a tone of cooperation. Being prepared means both knowing who you are interviewing so you don't make any embarrassing statements and having the first few questions prepared, even if you don't know all the questions.

TABLE 4-2 Summary of Data Collection Techniques

Interviews	
Strengths	Weaknesses
Get both qualitative and quantitative information	Takes some skill
Get both detail and summary information	May obtain biased results
Good method for surfacing requirements	Can result in misleading, inaccurate, or irrelevant information
	Requires triangulation to verify results
	Not useful with large numbers of people to be interviewed (e.g., over 50)
Group Meetings	
Strengths	Weaknesses
Decisions can be made	Decisions with large number of participants can take a long time
Can get both detail and summary information	Wastes time
Good for surfacing requirements	Interruptions divert attention of participants
Gets many users involved	Arguments about turf, politics, etc. can occur
	Wrong participants lead to low results
Observation	
Strengths	Weaknesses
Surface unarticulated procedures, decision criteria, reasoning processes	Might not be representative time period
Not biased by opinion	Behavior might be changed as a result of being observed
Observer gets good problem domain understanding	Time consuming
Review Software	
Strengths	Weaknesses
Good for learning current work procedures as constrained or guided by software design	May not be current
	May be inaccurate
Good for identifying questions to ask users about functions—how they work and whether they should be kept	Time consuming

TABLE 4-2 Summary of Data Collection Techniques (*Continued*)

Questionnaire	
Strengths	Weaknesses
Anonymity for respondents	Recall may be imperfect
Attitudes and feelings might be more honestly expressed	Unanswered questions mean you cannot get the information
Large numbers of people can be surveyed easily	Questions might be misinterpreted
Best for limited response, closed-ended questions	Reliability or validity may be low
Good for multicultural companies to surface biases, or requirements and design features that should be customized to fit local conventions	Might not add useful information to what is already known
Temporary Assignment	
Strengths	Weaknesses
Good to learn current context, terminology, procedures, problems	May not include representative work activities or time period
Bases for questions you might not otherwise ask	Time consuming
	May bias future design work
Review Internal Documents	
Strengths	Weaknesses
Good for learning history and politics	May bias future design work
Explains current context	Saves interview/user time
Good for understanding current application	Not useful for obtaining attitudes or motivations
Review External Documents	
Strengths	Weaknesses
Good for identifying industry trends, surveys, expert opinions, other companies' experiences, and technical information relating to the problem domain	May not be relevant Information may not be accurate May bias future design work

TABLE 4-3 Steps to Conducting a Successful Interview

1. Make an appointment that is at the convenience of the interviewee.
2. Prepare the interview; know the interviewee.
3. Be on time.
4. Have a planned beginning to the interview.
   a. Introduce yourself and your role on the project.
   b. Use open-ended general questions to begin the discussion.
   c. Be interested in all responses, pay attention.
5. Have a planned middle to the interview.
   a. Combine open-ended and closed-ended questions to obtain the information you want.
   b. Follow-up comments by probing for more detail.
   c. Provide feedback to the interviewee in the form of comments, such as, "Let me tell you what I think you mean, . . ."
   d. Limit your notetaking to avoid distracting the interviewee.
6. Have a planned closing to the interview.
   a. Summarize what you have heard. Ask for corrections as needed.
   b. Request feedback, note validation, or other actions of interviewee.
      - Give him or her a date by which they will receive information for review.
      - Ask him or her for a date by which the review should be complete.
   c. If a follow-up interview is scheduled, confirm the date and time.

A good interview has a beginning, middle, and end. In the beginning, you introduce yourself and put the interviewee at ease. Begin with general questions that are inoffensive and not likely to evoke an emotional response. Pay attention to answers both to get cues for other questions, and to get cues on the honesty and attitude of the interviewee. In the middle, be businesslike and stick to the subject. Get all the information you came for, using the techniques you chose in advance. If some interesting side information emerges, ask if you can talk about it later and then do that. In closing, summarize what you have heard and tell the interviewee what happens next. You may write notes and ask him or her to review them for accuracy. If you do notes, try to get them back for review within 48 hours. Also, have the interviewee commit to the review by a specific date to aid in your time planning. If you say you will follow up with some activity, make sure you do.

Interviews use two types of questions: open-ended and closed-ended. An **open-ended question** is one that asks for a multisentence response. Open-ended questions are good for eliciting descriptions of current and proposed application functions, and for identifying feelings, opinions, and expectations about a proposed application. They can also be used to obtain any lengthy or explanatory answers. An example of open-ended question openings are: "Can you tell me about . . ." or "What do you think about . . ." or "Can you describe how you use . . .".

A **closed-ended question** is one which asks for a yes/no or specific answer. Closed-ended questions are good for eliciting factual information or forcing people to take a position on a sensitive issue. An example of a closed-ended question is: "Do you use the monthly report?" A 'yes' response might be followed by an open-ended question, "Can you explain how?"

The questions can be ordered in such a way that the interview might be structured or unstructured (see Table 4-4). A **structured interview** is one in which the interviewer has an agenda of items to cover, specific questions to ask, and specific information desired. A mix of open and closed questions is used to elicit details of interest. For instance, the interview might start with "Describe the current rental process." The respondent would describe the process, most often using general terms. The interviewer might then ask specific questions, such as, "What is the daily volume of rentals?" Each structured interview is basically the same because the same questions are asked in the same sequence. Tallying the responses is fairly easy because of the structure.

An **unstructured interview** is one in which the interview unfolds and is directed by responses of the interviewee. The questions tend to be mostly open-ended. There is no set agenda, so the interviewer, who knows the information desired, uses the responses from the open-ended questions to develop ever more specific questions about the topics. The

TABLE 4-4 Comparison of Structured and Unstructured Interviews

Strengths	
Structured	Unstructured
Uses uniform wording of questions for all respondents	Provides greater flexibility in question wording to suit respondent
Easy to administer and evaluate	Can be difficult to conduct because interviewer must listen carefully to develop questions about issues that arise spontaneously from answers to questions
More objective evaluation of respondents and answers to questions	May surface otherwise overlooked information
Requires little training	Requires practice
Results in shorter interviews	
Weaknesses	
Structured	Unstructured
Cost of preparation can be high	May waste respondent and interviewer time
Respondents do not always accept high level of structure and its mechanical posing of questions	Interviewer bias in questions or reporting of results is is more likely
High level of structure is not suited to all situations	Extraneous information must be culled through
Reduces spontaneity and ability of interviewer to follow up on comments of interviewee	Analysis and interpretation of results may be lengthy
	Takes more time to collect essential facts

same questions used above as examples for the structured interview might also be used in an unstructured interview; the difference is that above, they are determined as a 'script' in advance. In an unstructured situation, the questions flow from the conversation.

Structured interviews are most useful when you know the information desired in advance of the interview (see Table 4-4). Conversely, unstructured interviews are most useful when you cannot anticipate the topics or specific outcome. A typical series of interviews with a user client begins with unstructured interviews to give you an understanding of the problem domain. The interviews get progressively structured and focused as the information you need to complete the analysis also gets more specific.

User interview results should always be communicated back to the interviewee in a short period of time. The interviewee should be given a deadline for their review. If the person and/or information are critical to the application design being correct, you should ask for comments even after the deadline is missed. If the person is not key in the development, the deadline date signifies a period during which you will accept changes, after the date you continue work, assuming the information is correct.

It is good practice to develop diagram(s) as part of the interview documentation. At the beginning of

the next interview session, you discuss the diagram(s) with the user and give him or her any written notes to verify at a later time. You get immediate feedback on the accuracy of the graphic and your understanding of the application. The benefits of this approach are both technical and psychological. From a technical perspective, you are constantly verifying what you have been told. By the time the analysis is complete, both you and the client have confidence that the depicted application processing is correct and complete. From a psychological perspective, you increase user confidence in your analytical ability by demonstrating your problem understanding. Each time you improve the diagram and deepen the analysis, you also increase user confidence that you will build an application that answers his or her need.

Interviews are useful for obtaining both qualitative and quantitative information (see Table 4-2). The types of qualitative information are opinions, beliefs, attitudes, policies, and narrative descriptions. The types of quantitative information include frequencies, numbers, and quantities of items to be tracked or used in the application.

Interviews, and other forms of data collection, can give you misleading, inaccurate, politically motivated, or irrelevant information (see Table 4-2). You need to learn to read the person's body language and behavior to decide on further needs for the same information. Table 4-5 lists respondent behaviors you might see in an interview and the actions you might take in dealing with the behaviors.

For instance, if you suspect the interviewee of lying or 'selectively remembering' information, try to cross-check the answers with other, more reliable sources. If the interview information is found to be false, ask the interviewee to please explain the differences between his or her answers and the other information. The session does not need to be a confrontation, rather, it is a simple request for explanation. Be careful not to accuse or condemn, simply try to get the correct information.

Persistence and triangulation are key to getting complete, accurate information. You are not required to become 'friends' with the application users, but interviews are smoother, yield more information for the time spent, and usually have less 'game-playing' if you are 'friendly' than if you are viewed as distant, overly-objective, or noninterested.

Meetings

Meetings are gatherings of three or more people for a fixed period to discuss a small number of topics and sometimes to reach consensus decisions. Meetings can both complement and replace interviews. They complement interviews by allowing a group verification of individual interview results. They can replace interviews by providing a forum for users to collectively work out the requirements and alternatives for an application. Thus, meetings can be useful for choosing between alternatives, verifying findings, and for soliciting application ideas and requirements.

Meetings can also be a colossal waste of time (see Table 4-2). In general, the larger the meeting, the fewer the decisions and the longer they take. Therefore, before having a meeting, a meeting plan should be developed. The agenda should be defined and circulated in advance to all participants. The number of topics should be kept to between one and five. The meeting should be for a fixed period with specific checkpoints for decisions required. In general, meetings should be no longer than two hours to maintain the attention of the participants. The agenda should be followed and the meeting moved along by the project manager or SE, whoever is running the meeting. Minutes should be generated and circulated to summarize the discussion and decisions. Any follow-up items should identify the responsible person(s) and a date by which the item should be resolved.

Meetings are useful for surfacing requirements, reaching consensus, and obtaining both detailed and summary information (see Table 4-2). If decisions are desired, it is important to ask the decision makers to attend and to tell them in advance of the goals for the meeting. If the wrong people participate, time is wasted and the decisions are not made at the meeting.

Joint application development (JAD) is a special form of meeting in which users and technicians meet continuously over several days to identify application requirements (see Figure 4-3). Before a

TABLE 4-5 Interviewee Behaviors and Interviewer Response

Interviewee Behavior	Interviewer Response
Guesses at answers rather than admit ignorance	After the interview, cross-check answers
Tries to tell interviewer what she or he wants to hear rather than correct facts	Avoid questions with implied answers. Cross-check answers
Gives irrelevant information	Be persistent in bringing the discussion to the desired topic
Stops talking when the interviewer takes notes	Do not take notes at this interview. Write notes as soon as the interview is done. Ask only the most important questions. Have more than one interview to get all information.
Rushes through the interview	Suggest coming back later
Wants no change because she or he likes the current work environment	Encourage elaboration of present work environment and good aspects. Use the information to define what gets kept from the current method.
Shows resentment; withholds information or answers guardedly	Begin the interview with personal chitchat on a topic of interest to the interviewee. After the person starts talking, work into the interview.
Is not cooperative, refusing to give information	Get the information elsewhere. Ask this person, "Would you mind verifying what someone else tells me about this topic?" If the answer is no, do not use this person as an information source.
Gripes about the job, pay, associates, supervisors, or treatment	Listen for clues. Be noncommittal in your comments. An example might be, "You seem to have lots of problems here; maybe the application proposed might solve some of the problems." Try to move the interview to the desired topic.
Acts like a techno-junkie, advocating state-of-the art everything	Listen for the information you are looking for. Do not become involved in a campaign for technology that does not fit the needs of the application.

JAD session, users are trained in the techniques used to document requirements, in particular, diagrams for data and processes are taught. Then, in preparation for the JAD session, the users document their own jobs using the techniques and collecting copies of all forms, inputs, reports, memos, faxes, and so forth used in performing their job.

A JAD session lasts from 3 to 8 days, and from 7 to 10 hours per day. The purpose of the sessions is to get all the interested parties in one place, to define application requirements, and to accelerate the process of development. Several studies show that JAD can compress an analysis phase from three months into about three weeks, with comparable results. The advantage of such sessions is that users' commitment is concentrated into the short period of time. The disadvantage is that users might allow interruptions to divert their attendance at JAD meetings, thus not meeting the objective. JAD is discussed in more detail in the Introduction to Part II.

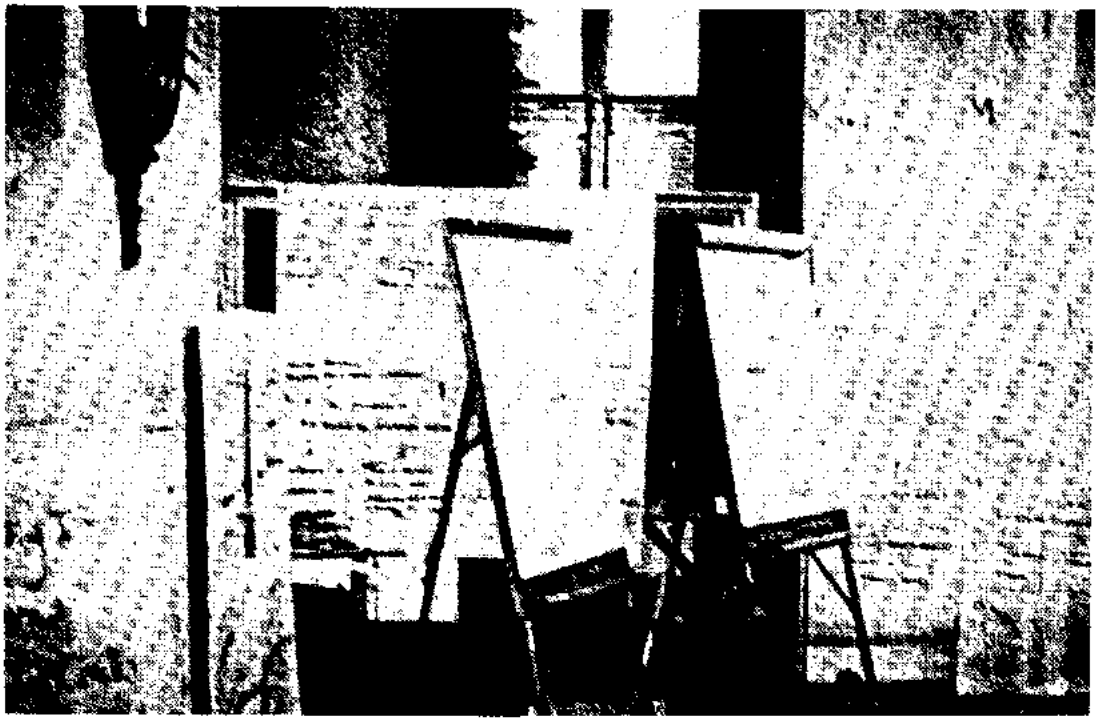


FIGURE 4-3 JAD Meeting

Observation

Observation is the manual or automated monitoring of one or more persons' work. In manual observation, a person sits with the individual(s) being observed and takes notes of the activities and steps performed during the work (see Table 4-2). In automated observation, a computer keeps track of software used, e-mail correspondence and partners, and actions performed using a computer. Computer log files are then analyzed to describe the work process based on the software and procedures used.

Observation is useful for obtaining information from users who cannot articulate what they do or how they do it (see Table 4-2). In particular, for expert systems, taking *protocols* of work is a useful form of observation. A **protocol** is a detailed minute-by-minute list of the actions performed by a person. Videotaping is sometimes used for continuous tracking. The notes or tapes are analyzed for events, key verbal statements, or actions that indicate reasoning, work procedure, or other information about the work.

There are three disadvantages to observation (see Table 4-2). First, the time of observation might not be representative of the activities that take place normally, so the SE might get a distorted view of the work. Second, the idea that a person is being observed might lead them to change their behavior. This problem can be lessened somewhat by extensive observation during which time the person being observed loses their sensitivity to being watched. The last disadvantage of observation is that it can be time-consuming and may not yield any greater understanding than could be got in less time-consuming methods of data collection.

Advantages of observation are several. Little opinion is injected into the SE's view of the work.

The SE can gain a good understanding of the current work environment and work procedures through observation. The SE can focus on the issues of importance to him or her, without alienating or disturbing the individual being observed. Some barriers to working with the SEs that are needed for interviews and validation of findings might be overcome through the contact of observation.

Some ground rules for observation are necessary to prepare for the session. You should identify and define what is going to be observed. Be specific about the length of time the observation requires. Obtain both management approval and approval of the individual(s) to be observed before beginning. Explain to the individuals being observed what is being done with the information and why. It is unethical to observe someone without their knowledge or to mislead an individual about what will be done with the information gained during the observation session.

Temporary Job Assignment

There is no substitute for experience. With a temporary job assignment, you get a more complete appreciation for the tasks involved and the complexity of each than you ever could by simply talking about them. Also, you learn firsthand the terminology and the context of its use (see Table 4-2). The purpose, then, of temporary job assignment is to make the assignee more knowledgeable about the problem domain. Temporary assignments usually last two weeks to one month—long enough for you to become comfortable that most normal and exceptional situations have occurred, but not long enough to become truly expert at the job.

Temporary assignment gives you a basis for formulating questions about which functions of the current method of work should be kept and which should be discarded or modified.

The disadvantage of work assignments are that it is time-consuming and may not be a representative period (see Table 4-2). The choice of period can minimize this problem. The other disadvantage is that the SE taking the temporary assignment might become biased about the work process, content, or people in a way that affects future design work.

Questionnaire

A **questionnaire** is a paper-based or computer-based form of interview. Questionnaires are used to obtain information from a large number of people. The major advantage of a questionnaire is anonymity, thus leading to more honest answers than might be got through interviews. Also, standardized questions provide reliable data upon which decisions can be based.

Questionnaire items, like interviews, can be either open-ended or closed-ended. Recall that open-ended questions have no specific response intended. Open-ended questions are less reliable for obtaining complete information about factual information and are subject to recall difficulties, selective perception, and distortion by the person answering the question. Since the interviewer neither knows the specific respondent nor has contact with the respondent, open-ended questions that lead to other questions might go unanswered. An example of an open-ended question is: "List all new functions which you think the new application should do."

A closed-ended question is one which asks for a yes/no or graded specific answer. For example, "Do you agree with the need for a history file?" would obtain either a *yes* or *no* response.

Questionnaire construction is a learned skill that requires consideration of the reliability and validity of the instrument. **Reliability** is the extent to which a questionnaire is free of measurement errors. This means that if a reliable questionnaire were given to the same group several times, the same answers would be obtained. If a questionnaire is unreliable, repeated measurement would result in different answers every time. Questionnaires that try to measure mood, satisfaction, and other emotional characteristics of the respondent tend to be unreliable because they are influenced by how the person feels that day. You improve reliability by testing the questionnaire. When the responses are tallied, statistical techniques are used to verify the reliability of related sets of questions.

Validity is the extent to which the questionnaire measures what you think you are measuring. For instance, assume you want to know the extent to which a CASE tool is being used in both frequency

of use and number of functions used. Asking the question, "How well do you use the CASE tool?" might obtain a subjective assessment based on the individual's self-perception. If they perceive themselves as skilled, they might answer that they are extensive users. If they perceive themselves as novices, they might answer that they do not use the tool extensively. A better set of questions would be "How often do you use the CASE tool?" and "How many functions of the tool do you use? Please list the functions you use." These questions specifically ask for numbers which are objective and not tied to an individual's self-perception. The list of functions verifies the numbers and provides the most specific answer possible.

Some guidelines for developing questionnaires are summarized in Table 4-6 and discussed here. First, determine the information to be collected, what facts are required, and what feelings, lists of items, or nonfactual information is desired. Group the items by type of information obtained, type of questions to be asked, or by topic area. Choose a grouping that makes sense for the specific project.

For each piece of information, choose the type of question that best obtains the desired response. Select open-ended questions for general, lists, and nonfactual information. Select closed-ended questions to elicit specific, factual information, or single answers.

Compose a question for each item. For a closed-ended question, develop a response scale. The five-response Likert-like scale is the most frequently used. The low and high ends of the scale indicate the poles of responses, for instance, *Totally Disagree* and *Totally Agree*. The middle response is usually neutral, for instance, *Neither Agree Nor Disagree*. Examine the question and ask yourself if it has any words that might not be interpreted as you mean them. What happens if the respondent does not know the answer to your question? Do you need a response that says, *I Don't Know*? Is a preferred response hidden in the question? Are the response choices complete and ordered properly? Does the question have the same meaning for every department and possible respondent? If the answers to any of these questions indicate a problem, reword the question to remove the problem.

If you have several questions that ask similar information, examine the possibility of eliminating one or more items. If you are doing statistical analysis of the answers, you might want similar questions to see if the responses are also similar (i.e., are correlated). If you are simply tallying the responses and

TABLE 4-6 Guidelines for Questionnaire Development

1. Determine what facts are desired and which people are best qualified to provide them.
2. For each fact, select either an open-ended or close-ended question. Write several questions and choose the one or two that most clearly ask for the information.
3. Group questions by topic area, type of question, or some context-specific criteria.
4. Examine the questionnaire for problems:
   - More than two questions asking the same information
   - Ambiguous questions
   - Questions for which respondents might not have the answer
   - Questions that bias the response
   - Questions that are open to interpretation by job function, level of organization, etc.
   - Responses that are not comprehensive of all possible answers
   - Confusing ordering of questions or responses
5. Fix any problems identified above.
6. Test the questionnaire on a small group of people (e.g., 5–10). Ask for both comments on the questions and answers to the questions.
7. Analyze the comments and fix wording ambiguities, biases, word problems, etc. as identified by the comments.
8. Analyze the responses to ensure that they are the type desired.
9. If the information is different than you expected, the questions might not be direct enough and need rewording. If you don't get useful information that you don't already know, reexamine the need for the questionnaire.
10. Make final edits, print in easy-to-read type. Prepare a cover letter.
11. Distribute the questionnaire, addressing the cover letter to the person by name. Include specific instructions about returning the questionnaire. Provide a self-addressed, stamped envelope if mailing is needed.

acting on the information, try to use one question for each piece of information needed. The minimalist approach keeps the questionnaire shorter and easier to tally.

Pretest the questionnaire on a small group of representative respondents. Ask them to give you feedback on all of the items that they don't understand, that they think are ambiguous, badly worded, or have responses that do not fit the item. Also ask them to complete the questionnaire. The answers of this group should highlight any unexpected responses that, whether the group identified a problem or not, mean that the question was not interpreted as intended. If the pretest responses do not provide you with new information needed to develop the project, the questionnaire might not be needed or might not ask the right questions. Reexamine the need for a questionnaire and revise it as needed. Finally, change the questionnaire based on the feedback from the test group. The pretest and revision activities increase the validity of the questionnaire.

Provide a cover letter for the questionnaire that briefly describes the purpose and type of information sought. Give the respondent a deadline for completing the questionnaire that is not too distant. For instance, three days is better than two weeks. The more distant the due date, the less likely the questionnaire will be completed. Include information about respondent confidentiality and voluntary questionnaire completion, if they are appropriate. Ideally, the questionnaire is anonymous and voluntary. To the extent possible, address the letter to the individual respondent.

Give the respondent directions about returning the completed questionnaire. If mailing is required, provide a stamped, self-addressed envelope. If interoffice mail is used, provide your mail stop address. If you will pick up responses, tell the person where and when to have the questionnaire ready for pickup.

Document Review

New applications rarely spring from nothing. There is almost always a current way of doing work that is guided by policies, procedures, or application systems. Study of the documentation used to teach new employees, to guide daily work, or to use an application can provide valuable insight into what work is done.

The term **documents** refers to written policy manuals, regulations, and standard operating procedures that organizations provide as a guide for managers and employees. Document types include those that describe organization structure, goals, and work. Examples of each document type follow:

- Policies
- Procedures
- User manuals
- Strategy and mission statements
- Organization charts
- Job descriptions
- Performance standards
- Delegation of authority
- Chart of accounts
- Budgets
- Schedules
- Forecasts
- Any long- or short-range plans
- Memos
- Meeting minutes
- Employee training documents
- Employee manuals
- Transaction files, e.g., time sheets, expense records
- Legal documents, e.g., copyrights, patents, trademarks, etc.
- Historical reports
- Financial statements
- Reference files, e.g., customers, employees, products, vendors

Documents are not always internal to a company. External documents that might be useful include technical publications, research reports, public surveys, and regulatory information. Examples of external documents follow:

- Research reports on industry trends, technology trends, technological advances, etc.
- Professional publications with salary surveys, marketing surveys, or product development information
- IRS or American Institute of CPA reports on taxes, workmen's compensation, affirmative action, financial reporting, etc.

Economic trends by industry, region, country, etc.
Government stability analyses for developing countries in which the application might be placed
Any publications that might influence the goals, objectives, policies, or work procedures relating to the application

Documentation is particularly useful for SEs to learn about an area with which they have no previous experience. It can be useful for identifying issues or questions about work processes or work products for which users need a history. Documents provide objective information that usually does not discuss user perceptions, feelings, or motivations for work actions.

Documents are less useful for identifying attitudes or motivations. These topics might be important issues, but documents may not contain the desired information.

Software Review

Frequently, applications are replacing older software that supports the work of user departments. Study of the existing software provides you with information about the current work procedures and the extent to which they are constrained by the software design. This, in turn, gives you information about questions to raise with the users, for instance, how much do they *want* work constrained by the application? If they could remove the constraints, how would they do the work?

The weaknesses of getting information from software review are that documentation might not be accurate or current, code might not be readable, and the time might be wasted if the application is being discarded.

To summarize, the methods of collecting information relating to applications include interviews, group meetings, observation, questionnaires, temporary job assignment, document review, or software review. For obtaining information relating to requirements for applications, interviews and JAD meetings are the most common.

Data Collection and Application Type

In this section, we identify the data gathering techniques most useful for each application type. Like most aspects of application development, the techniques can be used for all application types, but because of their strengths and weaknesses, they do not always result in the type of information that is needed most. In this section, we first match data collection techniques to the data types discussed in the first section. Then, the data types are matched to application types (from Chapter 1). Next, we match the data collection techniques to application types based on the data types they have in common.

Data Collection Technique and Data Type

Table 4-7 summarizes the discussion of the above sections. By matching technique for data collection to data type, we are more likely to identify information of interest than using other techniques. As the table shows, interviews and meetings are useful for eliciting all types of information. This is the reason they are most frequently used in application work.

Observation provides only crude numerical estimates of volumes, and is restricted to current time, varying ambiguity, and possibly variable semantics (see Table 4-7). Because the information from an observation is unstructured, some skill is required of the SE to impose a structure on it that fits the situation. Also, the information may be incomplete.

Questionnaires can ask structured questions about any time frame but only obtain complete answers for questions asked (see Table 4-7). If the questions are open-ended, the completeness might be quite low. Ambiguity in questionnaires should be low, but the question semantics might be misinterpreted by the respondents. Questions about volume at a department or organization level are usually inappropriate. Information about the volume of transactions or time for transaction processing for individual workers would get meaningful information.

TABLE 4-7 Data Collection Techniques and Data Type

Technique	Time	Structure	Completeness	Ambiguity	Semantics	Volume
Interview	All	All	All	All	Varies	All
Meeting	All	All	All	All	Varies	All
Observation	Current	Unstruct.	Incomplete	May vary	Varies	Crude measure
Questionnaire	All	Structured	Complete for questions asked	Low	Fixed but might be subject to interpretation	Individual volumes only
Temporary job assignment	Current	Unstruct.	Incomplete	Low-med.	Varies	For period of observation but may not be representative
Internal documents	Past-current	Unstruct.	Incomplete	Low-med.	Varies	Maybe
External documents	Mostly current-future	Unstruct.	Incomplete	Low-med.	Relatively fixed	N/A
Software review	Past-current	Structured	Complete for software	Low-med.	Fixed	Maybe

Temporary job assignments are similar to observation in having a high degree of uncertainty associated with the information obtained (see Table 4-7). The information tends to be current, unstructured, and incomplete depending on the period of work. Ambiguity varies from low to medium depending on how well-defined and structured the work is. Semantic content might vary depending on the shared definitions in the work group.

Documents provide unstructured, incomplete informations from which no relevant volume information is likely. The time orientation differs whether the documents are internal or external to the company (see Table 4-7). Internal documents are mostly oriented to the past or current situation. External documents are mostly oriented to current or future topics. The semantics of external documents on mature technologies or topics tend to be relatively fixed while that of internal documents might vary by department or division.

Software provides past, and possibly current, information that is structured because it is automated. The ambiguity should be low to medium, and semantics should be fixed since the application imbeds definitions of data and processes in code. Information on volumes may be present but should be cross-checked using other methods.

Data Type and Application Type

Application types are transaction processing (TPS), query, decision support (DSS), group decision support (GDSS), executive information (EIS), and expert systems (ES). Each of these has one or more predominate datatype characteristics that identifies its application. Table 4-8 shows all applications categorized for all data types. Here we discuss only

TABLE 4-8 Data Type by Application Type

Technique	Time	Structure	Completeness	Ambiguity	Semantics	Volume
TPS	Current	Structured	Complete	Low	Fixed	Any
Query	Past, current	Structured	Complete	Low	Fixed	Any
DSS	All	Structured	Varies	Low-med.	Varies	Med.-high
GDSS	Current-future	Unstruct.	Incomplete	Med.-high	Varies	Low
EIS	Future	Unstruct.	Incomplete	Med.-high	Varies	Low-med.
Expert system	Current based on past	Semi-structured	Incomplete	Med.-high	May vary	Low

the data types that differentiate between application types.

TPS contain predominantly known, current, structured, complete information (see Table 4-8). Recall that TPS are the operational applications of a company. To control and maintain records of current operations, you must have known, structured, current, and complete information.

Query applications have similar characteristics to TPS with the difference that they might concentrate on historical information in addition to current information (see Table 4-8). Queries are questions posed of data to find problems and solutions, and to analyze, summarize, and report on data. To perform summaries and reports with confidence, the data must be structured, complete, and interpreted consistently being both unambiguous and of fixed semantics.

DSS are statistical analysis tools that allow development of information that aids the decision process. The type of data that identifies DSS so that all time frames might be represented, may be incomplete, ambiguous, have variable semantics and medium to high volume (see Table 4-8). DSS might be used, for instance, in analyzing which of two variations on a given product might enjoy the larger market share. To do this analysis, past sales, current sales, and sales trends in the industry might all be analyzed and tied together to develop an answer.

GDSS are meeting facilitation tools for groups of people. GDSS tools operate in a structured manner working on data that is unstructured, current, and future-oriented. GDSS mostly deal with data that is incomplete and contains semantic and other ambiguities (see Table 4-8). The tools themselves are complete, unambiguous, and so forth, but the meeting information they process is not.

EIS are future-oriented applications that allow executives to scan the environment and identify trends, economic changes, or other industry activity that affect their governance of a company. EIS deal mostly with 'messy' data that is unstructured, incomplete, ambiguous, and contains variable semantics (see Table 4-8). Interpretation is always a problem with such data, which is why executives who excel at reading the environment are highly compensated.

Last, expert systems manage and reason through semistructured, incomplete, ambiguous, and variable semantic data (see Table 4-8). Experts and ESs take random, unstructured information and impose a structure on it. They reason through how to interpret the data to remove ambiguity and to fix the semantics. Therefore, even though the data coming into the application might have these fuzzy char-

TABLE 4-9 Data Collection Technique and Application Type

	TPS	Query	DSS	GDSS	EIS	ES
Interview	**X***	**X**	**X**	**X**	**X**	X
Meeting	X	X	X	**X**	X	X
Observation	X	X	X	Limited	Limited	**X**
Questionnaire	X	X	X			
Temporary job assignment	X	X	X			
Internal documents	X	X	X	Limited		
External documents	X	X	X	X	X	X
Software review	X	X	X	Limited	Limited	Limited

*Boldface identifies most frequently used method.

acteristics, the data processing is actually highly structured.

Data Collection Technique and Application Type

Finally, in discussing different data types, we desire to know which data collection techniques are best for each application type. By combining the information in Tables 4-7 and 4-8, we develop Table 4-9 to summarize data collection techniques for each application type. The table entry in boldface shows the principle method of data collection for each technique.

TPS and query applications can profit from the use of all techniques. Meetings and interviews predominate because they elicit the broadest range of responses in the shortest time (see Table 4-9). Observation and temporary job assignment are particularly useful in obtaining background information about the current problem domain, but need to be used with caution so as not to prejudice the design of the application. Questionnaires are useful when the number of people to be interviewed is over 50. Also, questionnaires are useful in identifying characteristics of users that determine, for instance, training required of users during organizational feasibility analysis. Also, if the screen requires, for instance, colors or different types of screen arrangements, questionnaires might be useful for presenting a small set of alternatives from which the actual users choose.

DSS also are shown as having a use for all data collection techniques, but not all techniques are practical in all cases (see Table 4-9). DSS are generally developed for use by people in jobs with a significant amount of discretion in what they do and how they do it. Therefore, observing or working with one or two people as representative may result in a biased view of the application requirements for a general purpose DSS. Even for a custom DSS, observation and job assignments might both be impractical if the SE does not know enough about the job being supported to interpret what she or he observes. The same holds true of documents. Documents, such as statistical reports, might be useful for providing samples of the types of analyses desired in a DSS. Other documents, such as policies, procedures, and so on, are not likely to be relevant to the application. For general purpose DSS with a large number of users, questionnaires are a useful way to identify the range of problems and analysis techniques required in the DSS. This information might be followed by interviews or meetings to determine DSS details.

GDSS are usually custom-built suites of software packages that provide different types of support for automated meetings. As such, the SE working on a GDSS environment needs to know the types of issues, number of participants, as well as types of reasoning and group consensus techniques desired. GDSS components are neither common knowledge nor frequently used; you might build one GDSS in a career. Therefore, significant time would be spent finding out about the market,vendors, and GDSS components. External documents on vendor products are useful in developing questions that elicit the required information. After knowledge of the market is obtained, interviews and meetings are useful to determine the specific requirements and to review, with users, what the GDSS can and cannot do. Other methods might have some limited value. For instance, observation of an actual meeting that might be automated would be useful for the SE to gain insight about how a tool might work. Internal documents that provide information about meetings that the GDSS is expected to provide would also be useful. Both of these techniques, observation and document review, have a specific limited role in providing the information needed to build a GDSS. Any software review that is done would be review of other company's GDSS facilities or of vendor products, rather than review of in-house software.

EIS are similar to GDSS in the rarity and general lack of knowledge about what an EIS is. EIS are not standard applications with a screen for data entry of some type and reports that are displayed. EIS are information presentation facilities that can be structured with menus and selection tools, but may display document pages, newspaper articles, book abstracts, summary reports, and so on. EIS are usually built for a small number of users, which eliminates the use of questionnaires. EIS are custom and one-of-a-kind environments for which past documents or software will be of limited value. Observation is most likely limited because executives would be uncomfortable in being observed. Temporary job assignment is not possible because you cannot just 'be an executive' for a week or two. This leaves external documents, interviews, and meetings as the most likely techniques for data collection (see Table 4-9). As with GDSS, external documents will be mostly to identify the market, vendors, and products. Interviews are most likely to be used to determine executives' information needs and preferred delivery platforms.

Finally, SEs use interviews, observation, and external documents the most in developing expert systems (see Table 4-9). Experts frequently can talk about external aspects of their jobs, the physical cues they use as inputs, and the result of their reasoning and how it is applied to the business. They are just as frequently unable to discuss their reasoning processes and how they put the cues together to make sense of unstructured situations. Experts, by definition of the term expert, have so *internalized* their work that they just do it. They don't think *consciously* about *how* they are doing what they do. Therefore, observation, in particular, the use of protocol analysis, is useful in getting information the expert might not be able to articulate. Protocol analysis is time-consuming and indefinite because you, the SE, are inferring a reasoning process from actions taken. At best, the protocol analysis gives you questions to ask about the work that assist the experts in discussing aspects of work they ordinarily cannot. Thus, observation is interleaved with interviews to discuss what is observed. As the process continues, structure is imposed on both the data and the problems to begin to develop the ES. The process of obtaining an expert's reasoning processes is called **knowledge elicitation**. The process of structuring the unstructured data and reasoning information is called **knowledge engineering**. Knowledge engineering is an activity that is difficult to learn and requires training through an apprenticeship approach in which the trainee works with an expert knowledge engineer.

PROFESSIONALISM AND ETHICS

A **profession** is defined as a job requiring advanced training. Computer information systems development and any job dealing with information technologies qualify as professions. **Professionalism** is acting in accordance with the highest expectations of a professional group. Those expectations are codi-

fied in professional codes of ethics for various organizations. The organizations relating most closely to IS professions are the Association of Computing Machinery (ACM) and Data Processing Management Association (DPMA). Both organizations have ethical conduct codes and the codes are similar. The most widely publicized code for the Association for Computing Machinery [1990], follows:

1. The developer shall act with integrity at all times.
   a. The developer shall qualify an opinion outside his or her area of competence.
   b. The developer shall not falsify his or her qualifications.
   c. The developer shall not knowingly issue false statements about the present or expected status of a system.
   d. The developer shall not misuse confidential or proprietary information.
   e. The developer will remain sensitive to and will reveal potential conflicts of interest.
2. The developer should constantly strive to increase his or her competence in the profession.
   a. A developer will diligently attempt to develop systems that perform their intended functions and satisfy the organization's needs.
   b. A developer will help his or her colleagues develop professionally.
3. A developer shall accept only assignments for which there is reasonable expectation of meeting the goals of the system.
4. A developer should use his or her special knowledge to advance the health, privacy, and general welfare of the public and society.
   a. A developer should always consider the individual's right to privacy when working with data.
   b. A developer should refrain from participating in a project in which he or she feels there will be undesirable consequences for individuals, organization, or society as a whole.

If you read the ACM Code of Ethics carefully, note that it contains ethical topics and professionalism topics. To separate out what is professional conduct from what is ethical conduct, we first define ethics terms and relate ethics to IS professions. Anything that is unethical is also unprofessional, but the reverse is not true. Professionalism is a broader subject than ethical behavior. In fact, the early name for codes of ethics was 'codes of professional behavior.' Ethics is in the section on data collection because many of the issues are concerned with user relations and are most evident in data collection activities.

So, what is ethics? **Ethics** is the branch of philosophy that studies moral judgment and reasoning. A **dilemma** is any situation requiring a choice between two unpleasant alternatives. Therefore, an **ethical dilemma** is any situation in which a decision results in unpleasant consequences requiring moral reasoning. The addition of information technologies to organizations presents novel, little understood opportunities for unethical behavior that are rarely discussed in texts.

Ethics is an issue of growing interest as it relates to information technologies. You, as users and developers of ITs, are sometimes in particular circumstances that subject you to dilemmas that need to be reasoned through to reach an ethical decision. One problem with ethics is that it is misunderstood as religious upbringing and the application of religious thought to real life situations. In fact, that is incorrect. Ethical decisions and reasoning are based on philosophies of rights, equity, and utility, that is, the greatest good for the greatest number of people. Ethics requires evaluation of alternatives, requiring only belief in the equality and dignity of man. Next, we discuss ethics as it relates to different aspects of data collection and user interactions in application development. Then, a procedure for reasoning that is likely to lead to ethical decisions is presented for your use.

Ethical Project Behavior

Confidentiality

Always be trustworthy of information told in confidence. In fact, assume that any interview information is in confidence, unless the person being interviewed is specifically told that it is 'on the record.' Besides being unethical, telling 'tales out of school' will eventually return to hurt your career.

If you think some information gained in privacy should be shared, ask if the interviewee minds if you discuss it. With permission, the bounds of confidentiality are removed and you are free to discuss the information.

The exception to this rule occurs when a person confides in you about an illegal act. You are legally bound to report any illegal activity to the managers, company authorities, and police, if no action is taken. By law, if you do not report illegal acts, you are an accessory to the act and are also libel to legal action.

Privacy

Experts have a right to know when their experience and knowledge are being used in an application. The basic rule is treat others as you would like to be treated. Would you like it if the company observed your use of computers and built systems based on it? Especially in building expert systems there are ethical issues about ownership of expertise. There should be no observation, in person or by computer, without permission. No one should be coerced into cooperation. Participation should be voluntary.

Ownership

Computers are now so much a part of corporate life that we tend to get confused about who owns the resources. On an intellectual level, most people recognize that the company that owns the computers also owns the computer time. But, in a given situation, most people feel that if the resource is not used it is wasted, and that computer time is like the *ether*, a free resource that is there for the taking. Most executives do not feel the same way, whether or not there is a policy about computer resource use.

Find out, in advance, the company policy or owner feelings about personal use of computing resources, then follow their guidance. Actions like running a program for a friend, doing personal finances, keeping track of the baseball team, and so on may or may not be ethical, depending on how the company feels about the use of its resources.

Who owns work and work-related products should be spelled out in detail so that if you feel something is rightfully yours, so does the client/company and you can feel ethical about taking it. For instance, technical, user, or operational documentation, screen designs, data dictionary, program code, vendor literature, or other products that you develop or gather in the course of development are all subject to ownership confusion. If you work for a consulting company and develop a proprietary application, like ABC's rental system, you have no right to sell the processing to other companies. This right is *negotiable* and belongs only to the client unless that right is specifically itemized in the contract. Be clear about ownership and you are less likely to be fired or sued over ownership rights.

The expertise that you gain from working on a project is **intellectual property**. Expertise is yours unless you sign a contract to the contrary. However, it is unethical to use your company-specific knowledge for personal, noncompetitor, or competitor financial gain unless you have an agreement with your employer about such use. Usually employers ask that you not divulge proprietary information, but the definition of *proprietary* may be open to interpretation. Also, employers can bar you from using information for one to two years if they can prove that it might hurt their business. The best course of action is to get such issues in the open and decided in advance so no conflict occurs.

Politics

Try to never be mixed up in a political battle. This is easier than it sounds, especially if you are the SE or project manager. **Politics** is the science of management often driven by personal motivation. In organizations, most people have the company's interest in mind when they make decisions; everyone is also assumed to at least consider their personal situation in decisions, as well. Some people put personal improvement ahead of all other considerations, even to the detriment of the company. Extreme selfish motivation without regard to the outcome for others or the company is unethical.

In a political battle, the politician(s) try to manipulate the project results to improve their position in the company. Political maneuvering might take

different forms: stalling, lying, artificial requirements, false cooperation, or different public and private statements. You, as the SE or PM, must become sensitive to such actions and learn how to diffuse them. The tactics are manifested in the discussion of interviewee actions and interviewer reactions in Table 4-5.

Courtesy

It is not necessary to tell every project problem to the user. You are ethically bound to discuss problems that might impact schedule, budget, or accuracy. *When* to tell a user about problems requires common sense. You should tell them early enough to warn them that the problem is coming, and late enough not to have been a whistleblower for nothing. Never wait until the last minute when nothing can be done to fix the problem, or all project participants lose credibility. Always solicit user assistance in problem resolution once they are told. The purpose of weekly status meetings is to provide status and identify problems and their anticipated resolution. These problems always foreshadow schedule and budget problems when they remain outstanding for a long period. A problem outstanding several months with no solution in sight will probably impact the schedule and budget. In keeping the user up-to-date on technical problems you indirectly apprise them of potential cost and budget overruns.

Personal Manner and Responsibility

When people work on a project with others, they sometimes lose sight of their contribution as standing on its own for quality review. Somehow the notions of 'on time, within budget, and accurate' have meaning to the project but not to the individual who is coding and testing a module. One role of the PM and SEs is to instill the sense of responsibility in every person. Each person should know their tasks, budget, expected resource use, and due date. Each person should be held accountable for meeting their deadlines and for having no errors in the code. Accountability is easy to displace in project work; who is responsible becomes diffuse. Some people say the project manager is always accountable. Some say the analysts and SEs. Some say no one. The short answer is that *everyone* is responsible for and should be made accountable for his own work and its integration into the project whole.

Do not talk to your manager, client, or your employees about work problems that do not relate to project completion. This is just good business. Managers and clients want answers and solutions, not problems. Therefore, they should be informed of status and problems that might someday affect them, but should otherwise be left alone. A manager doesn't want to know how Suzie in the typing pool or Carl in the copy room butchered your work. You deal with it and forget it. If you have a problem with the quality of someone's work who does not report to you, mention it to that person, and if unresolved, talk to their manager. The less accusatory and more factual you can be, the less like a whiner and complainer you appear. Be sure you can back up any accusations you make.

Do not tell the client or your manager about your personal problems unless you have a personal relationship. Personal problems can always be blamed for everything that goes wrong, but that is neither adult nor ethical. Henry Ford's famous quote, "Never complain, never explain," comes to mind here. Your job at work is to work, so just do it.

Do not get emotionally involved with the user. If there is a budding relationship, it can wait until the project is complete. Emotional involvements are easy to fall into when you are together 10 to 15 hours a day for months at a time. They also are prone to collapse as soon as a new project begins and you and they both work with others 10 to 15 hours a day. Emotional attachments cloud judgment and do not belong in the office.

Never intentionally mislead. Never lie. Never give false impressions, false perceptions, or any information that might cause users to infer a better, bigger, more functional application than you plan to deliver. Users will form their opinions based on what you and their managers tell them. Don't oversell the application and what it can do for their job. Also, if a downsizing is taking place at the same time, don't falsely give people hope that their jobs will be saved

when they might not. You don't raise alarms, but you don't give false hope either.

Ethical Reasoning

When you feel you are confronted with a problem that requires ethical reasoning, you need some way to identify all potential stakeholders, to evaluate the alternative courses of action, and to reason through the alternatives. One such method is presented here as a way to initiate reflection on your own thinking about the way you reason through tough problems. This is certainly not the only method of problem reasoning.

Identify Stakeholders

First, identify who might benefit or suffer from your decision. This action identifies **stakeholders**, people who have a stake in the outcome of your action. This is a difficult task, especially with computer use when you might not know the stakeholders personally. Stakeholders might be stockholders of a company, the company itself, your boss, you, the user community, the user/client for the application, society, or people subject to direct or indirect connection to the application. For instance, space shuttle astronauts, patients in a hospital, people who live near the plant in which the application will run, e-mail recipients, report users, governments, data entry clerks and their managers, all might be stakeholders.

Identify Actions Stakeholders Would Choose

Then, identify the action each stakeholder would prefer you to take and why. This task defines all possible actions. Begin with yourself. What do you want to do? Why do you think this is the best decision? Answer these questions from the perspective of each stakeholder group. Putting yourself in each stakeholder group's position requires objectivity and distance from the problem.

Eliminate Alternatives

Next, determine if there are any policies, procedures, laws, or other guidelines that make one or more alternatives untenable. Cross them off the list. Once a type of conduct crosses over into governance by laws, it is no longer an ethical issue, but becomes a legal one. Always obey the laws of the country you are in and the country you represent. For instance, bribery is a way of life in many countries, but not in the United States. Therefore, you are legally bound not to use bribery in business when you work for an American firm.

Policies and procedures of companies are similar in codifying conduct, but do not hold the same stringency of penalty for their transgression. Violation of policies is usually a fireable offense, meaning you lose your job when you violate a policy. Procedures are less stringent, but are expected to be followed. You might receive a letter of reprimand for not following a procedure exactly.

Guidelines, such as the professional code of ethics listed above, also provide heuristics about conduct to help you in governing your work behavior. There is no direct penalty in not following a code of ethics. You might be sued or fired, but the punishment is not from the professional organization.

Reason Through Negative Outcome Alternatives

For the possible courses of action remaining on your list, reason through each by asking key negative questions. If the answer to any of these negative outcome questions is yes, remove the alternative action from the list.

Are the rights of any person or group violated by this action? Consider the right to privacy, ownership of information about individuals' buying habits, payment habits, income, tax status, and so on. Consider the rights to company privacy of customer, financial, personnel, medical, and other proprietary information. Ask if the lack of security and access controls, for instance, subject the database to casual browsing by system users. If such browsing could result in a violation of privacy to customers, it should be prevented.

Does taking this action result in inequitable treatment of a person or group? Equitable treatment requires judgment of equality. In multinational companies, inequity might be seen as a business deci-

sion. For instance, many US corporations initially got into international business by dumping their second rate quality goods in other markets. Was this ethical? The answer is in the manner in which it was done. If the goods were sold as second quality, there is no issue. If the goods were sold as first quality, the companies basically lied and were unethical.

Companies might be subject to inequity because of their internal staff quality, too. Does the company lose money because of the inefficiency of design? A manager, for instance, might insist on using a particular software because he knows it, even though it is not efficient for the task. The manager is making a trade-off of current knowledge versus cost and time for learning a new product, that can cross the line into unethical behavior when it costs the company tangible amounts of money. Using mainframes which rent for millions instead of networks that cost thousands could be construed as unethical when networks are not even considered because of a lack of expertise. In other words, making a business decision to stay with a significantly more expensive alternative after considering all alternatives, is ethical. Avoiding a comparison of alternatives or making a decision because of technical ignorance is not ethical.

Does taking this action have the potential of placing a person or company in jeopardy financially, physically, legally, or morally? Hospital applications that hook patients to computerized monitors, transportation industry applications that affect safety of planes and cars, power plant applications that deal with monitoring power-generation equipment, and so forth, are all potentially life-threatening. We need such applications, but their design and maintenance must be of the highest possible quality to pose the least risk to human life. If corners are cut on analysis, design, or testing, lives can be lost.

Reason Through Neutral and Positive Outcome Alternatives

For remaining actions, ask key positive outcome questions to select the best alternative. Does taking this action result in the best possible outcome for all stakeholders? What is the result of taking no action?

If only negative outcomes are possible, does taking this action result in the least harm to all stakeholders? If this is the case, who suffers and what type of injury? If the stakeholder is warned in advance, can the problem be averted?

Select a Course of Action

When all the pros and cons of each alternative have been identified, select the alternative that produces the greatest good for the greatest number of people, that does not violate anyone's rights, and that results in the most equitable decision, with all stakeholders' equity considered.

SUMMARY

Data gathering is done during every phase of application development, but serves different purposes in each phase. The types of data collected depends on the type application and phase of development.

Data types refer to the characteristics of data for time-orientation, structure, completeness, ambiguity, semantics, and volume. Attention to data types in selection of data collection technique is less likely to *cause* errors and more likely to *find* errors than inattention to data type. The cost of errors rises dramatically the later in the development process it is found. Time orientation of data refers to past, present, or future data requirements for an application. Data structure refers to the extent to which data can be classified. Data completeness is the extent to which desired information is present. Ambiguous data have unclear or multiple meanings; companies strive for unambiguous definitions for data. Data semantics are the meanings, we as organization employees, give to data. Volume is the numbers of each item of interest in an application. Volumes can have widely varying time orientations. SEs must attend to peak as well as average volume.

Several data collection techniques were discussed, including interviews, group meetings, questionnaires, observation, temporary job assignment, review of internal and external documents, and review of software. Interviews are meetings between two or three people for obtaining any type

of information. Interviews can be structured or unstructured. Questions asked can be open-ended or closed-ended.

Group meetings include four or more people and can substitute for interviews or can be used to validate interview findings. Joint application development meetings are a special type of meeting specifically convened to develop application requirements. Special training and planning are required for JAD sessions. Both interviews and meetings require attention to an agenda and time period.

Observation is the monitoring of one or more persons' work. Observation is useful for learning a problem domain and is most often used in expert system development. A data analysis technique called protocol analysis is used to infer the reasoning processes of experts from detailed manuscripts of their actions during a period.

Temporary job assignment is an alternative means of gaining problem domain experience for nonmanagerial, nonexecutive jobs. Questionnaires are structured forms of interviews conducted on many people, usually more than 50. Statistical techniques are frequently used in analyzing questionnaire results. Reliability and validity of the questions are issues to be considered in questionnaire development.

Document review is useful in gaining background information about an application area. Documents can be internal or external to the company.

Software review is the analysis of programs and documentation to learn the details of a current application.

In developing the information about data collection technique related to application type, we also related data collection technique to data type and data type to application type. From these analyses, we find that interviews and meetings are most frequently used because they are the only techniques useful regardless of application type. The other techniques have specific purposes for each application type. For instance, software review for TPS, temporary job assignment, or observation are useful in gaining problem domain experience. Observation is most useful in expert system development. External documents are important in unique GDSS and EIS development. Questionnaires are most useful in DSS for general use in a company, for surveying user preferences for design options, or for obtaining detailed information about the application from a large number of people.

KEY TERMS

closed-ended question
data ambiguity
data completeness
data semantics
data structure
data time-orientation
data volume
dilemma
document
ethical dilemma
ethics
intellectual property
interview
joint application development (JAD)
knowledge elicitation
knowledge engineering
meetings
observation
open-ended question
peak volume
politics
profession
professionalism
protocol
questionnaire
reliability
semantics
stakeholder
structured interview
triangulation
unstructured interview
validity

REFERENCES

Flaaten, Per O., Donald J. McCubbrey, P. Declan O'Riordan, and Keith Burgess, *Foundations of Business Systems*, 2nd ed. Fort Worth, TX: Dryden Press, 1992.

Gause, Donald C., and Gerald M. Weinberg, *Exploring Requirements Quality Before Design*. NY: Dorset House Publishing, Inc., 1989.

Lucas, Henry C., Jr., *The Analysis, Design, and Implementation of Information Systems*, 4th ed. NY: McGraw-Hill, Inc., 1992.

Mockler, Robert J., and Dorothy G. Dologite, *Knowledge-based Systems: An Introduction to Expert Systems*. NY: Macmillan Publishing Co., 1992.

Zahedi, Fatemah, *Intelligent Systems for Business: Expert Systems with Neural Networks*. Belmont, CA: Wadsworth Publishing, 1993.

EXERCISES

1. Ethics is far from a settled issue, especially as it relates to use of information technologies. One issue, for instance, is that development of artificially intelligent applications might be unethical because we do not know how they will turn out. That means, we cannot predict if a person or company will get hurt. Debate this issue and develop conclusions for your class. Summarize the debate and send it to a trade magazine such as *Communications of ACM*, *Computerworld*, or *Datamation*.
2. For ABC Video, play the roles of Vic, Mary, and Sam. Either write or playact an interview to elicit requirements for the proposed rental application. Mix the use of open and closed questions to follow a chain of logic.
3. Develop a questionnaire that might be used with the user community of the Office Information System case in the Appendix.

STUDY QUESTIONS

1. Define the following terms:
   - ambiguity
   - ethical dilemma
   - joint application development
   - professional
   - professionalism
   - reliability
   - semantics
   - structure of data
   - triangulation
2. Why are data types important? What happens when the wrong data collection techniques are used? How does data collection technique relate to costs in applications?
3. How do data types relate to applications?
4. Discuss the cost of fixing errors in applications.
5. How do ambiguity and semantics differ? Why are they both important?
6. When are temporary job assignments not a useful data collection technique?
7. What type of information can be got from temporary job assignments?
8. What is the use of reviewing documents? How do you choose whether to review internal or external documents?
9. Why would you ever review software? What are the pitfalls of software and software documentation review?
10. Compare and contrast individual interviews and meetings, listing two purposes that are the same for both techniques and two that are different.
11. Compare and contrast structured and unstructured interviews.
12. Compare and contrast open-ended questions and closed-ended questions.
13. Describe how an unstructured interview progresses. What types of questions are used as the opening? How does the interviewer know what types of questions to ask? What types of questions are used after the opening?
14. Which kinds of data can you best get from observation?
15. Which kinds of data can you best get from external document review?
16. Which kinds of data are you unlikely to get from a questionnaire?
17. Which data collection technique is most useful for obtaining expert reasoning processes? Why? Describe the use of the technique.
18. Which data collection technique is most useful for obtaining executive needs for an EIS?
19. Why are expert systems and EIS unique?
20. Which question types are used for factual, detailed explanations of work processes?
21. How do you select between structured and unstructured interviews?
22. What is the typical follow-up to an interview? Who does what and when?
23. Why are meetings a useful data collection technique? How do you plan a meeting to avoid wasting time?
24. Describe how to develop a questionnaire.
25. Describe protocol analysis. When is it used? What application type(s) is it most used for?
26. What type of data are most likely in a DSS?
27. Describe the time-orientation of EIS. What type of data is associated with EIS?
28. Describe knowledge engineering. When is it used and why?

29. What is the difference between professionalism and professional ethics?
30. Discuss three of the six areas of ethical conduct by IS professionals.
31. Describe an ethical dilemma you might face in application development work. How should it be dealt with?
32. Describe the reasoning process for developing an ethical solution to some issue.

★ EXTRA-CREDIT QUESTIONS

1. For ABC Video's rental application, we still do not know accurate counts for volumes of rentals, late returns, on-time returns, late fees, or customers. How would you go about finding this information? Be specific in identifying a data collection technique, the number of people involved, and the amount of time involved. At what stage of the development process should this information be got?
2. The ACM's Code of Ethics, number 2, discusses the need for developers to constantly increase competence in their profession and to help others to do likewise. Is this an ethical issue? Who are the stakeholders to the issue? Reason through the issues and develop your own thoughts on the subject. Compare them to classmates, arguing for your position.
3. List and define the data type for all data currently identified for ABC's rental application. Refer to Chapter 2 for the data definitions.

PART II

PROJECT INITIATION

The two chapters in this section address the activities that take place before analysis of a specific project begins. Project initiation can take place in several different ways. First, it can be part of a larger enterprise reengineering effort. Second, a project might be initiated as part of an information systems planning effort. Third, a project might be initiated based on a user request for a specific project. All three methods of project initiation are equally feasible and equally useful in beginning an application development project.

Chapter 5 addresses the first two project initiation efforts. The main discussion is how to do a reengineering design of an organization and plan applications and technologies to support the redesign. Enterprise level planning, such as an information systems plan, is described as a subset of activities that focus on applications only and are an abbreviated reengineering study. Most researchers and industry experts, such as James Martin, recommend that at least an information systems plan (ISP) is a worthwhile planning activity in existing organizations. Both reengineering and ISPs result in plans for multiple applications which are prioritized for development.

Enterprise level planning exercises are costly, and some companies cannot afford to spend computer resources on such studies. In these companies, application development projects are initiated via a direct request from a user. Also, companies that do enterprise level plans might desire to reconfirm recommendations that might be two or three years old. For direct initiation and for reconfirmation of recommendations, a user memo to the Information Systems Manager or to an IS Steering Committee can initiate project assessment. Such an assessment is called a *feasibility study*.

Chapter 6 details the activities involved in a feasibility study. A feasibility study is performed to assess the financial, technological, and organizational readiness of the company for the application. Feasibility is an important analysis that is usually conducted on individual application projects rather than on a whole group of applications, such as might be identified in an ISP or organizational reengineering project. The feasibility analysis determines the extent to which new technologies, skills, or training are required by the user and developer staffs and assesses the ability of the company to pay for the development project.

Part of the technical feasibility is to define a direction for the application development through an evaluation of technical development alternatives. For instance, an application might be on-line or real-time; it might be on a standalone PC, on a PC connected to a local area network, or on terminals attached to a mainframe; it might use a 4GL database software such as Oracle™ or a full-service database such as IMS DB/DC.[1] Likely alternatives are evaluated to determine the extent to which functional requirements would be supported, and to determine any alternative-specific benefits that might be present. A recommendation for technical concepts is made and may (or may not) be accepted at the completion of the feasibility study. Even though the concept need not be cast in concrete at this time, it helps to have a sense of the operational environment for conducting the analysis phase of the project.

A risk assessment should be performed as part of feasibility analysis. The risk assessment identifies technical, personnel, and financial problems that could hinder the successful completion of the project. For each risk defined, two types of plans are developed. First, a contingency plan to deal with the problem if it should occur is defined. Second, immediate steps to minimize the probability of the risk's occurrence are planned and taken.

1 Oracle™ is a trademark of the Oracle Corporation. IMS DB/DC is a product of IBM Corporation.

CHAPTER 5

ORGANIZATIONAL REENGINEERING AND ENTERPRISE PLANNING

INTRODUCTION

As the economy becomes more global and the business climate more competitive, companies need to reevaluate what they do and how they do it. Reengineering is the evaluation and redesign of business processes. The goal is to streamline the organization to include only the business functions that *should* be done rather than necessarily improve on what is done today. Reengineering can introduce radical change into organizations with information technologies as key to supporting new organizational forms and providing information delivery to its users.

When radical approaches are not necessary (or wanted), the techniques of reengineering can be scaled down to provide enterprise level plans for information systems. Enterprise level planning techniques originally were developed in response to managers' complaints that IS departments did not respond to their information needs and frequently built applications that the company did not need. Enterprise planning techniques match IS plans to organization plans and are also used within the context of reengineering. Techniques include stakeholder analysis, critical success factors, and information systems planning (ISP). In this chapter, we first develop the conceptual basis and methodology for reengineering. Enterprise techniques are defined for use in reengineering analysis. Then, enterprise level IS planning, without organization design, is described. The last section identifies computer-aided software engineering (CASE) tools that support reengineering and enterprise level analysis techniques.

CONCEPTUAL FOUNDATIONS OF ENTERPRISE REENGINEERING

Organizational reengineering is the evaluation and redesign of business processes, data, and technology (see Figure 5-1). The goals of reengineering are to achieve *dramatic* improvements in quality, service, speed, use of capital, and reduced costs. The rationale for business reevaluation comes from need. The need may be to turn around a failing company, to increase competitiveness, to improve customer service, to increase product quality, or any combination of these. The philosophy of reengineering is

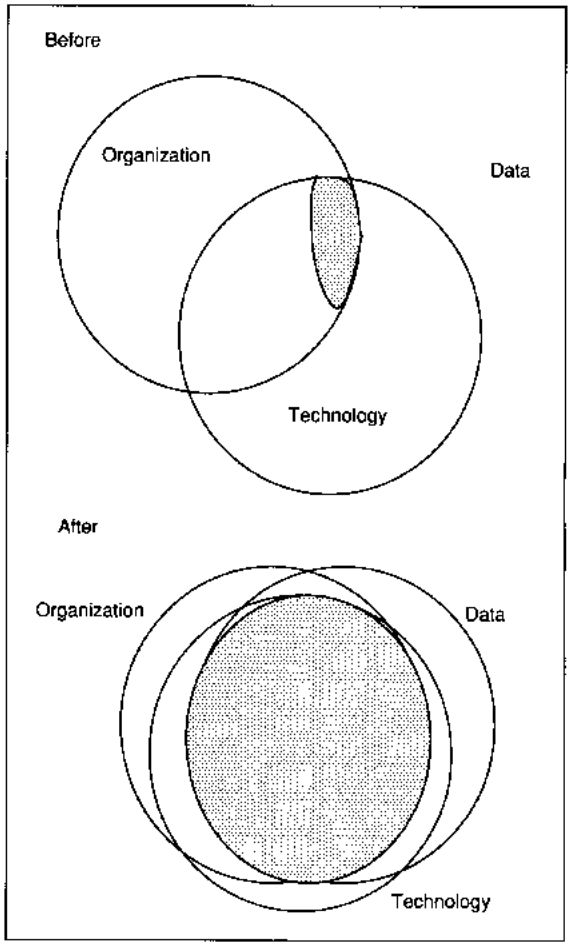


FIGURE 5-1 Reengineering Targets

that, when implemented alone, total quality programs, organization redesign, or information technology are inadequate for an organization to realize its potential. The main resources of organizations today are people and information. Both people and data, the raw material of information, have to be optimized to even *try* to meet the company's potential. Organization redesign optimizes the people resource; the interjection of quality improves both organization and data. Complete reevaluation of technology that provides the information infrastructure optimizes the data and delivery of data to the people who need it. This chapter discusses how to evaluate the organization and its information requirements, how to reengineer both the organization *and* the technology, and how to plan for the implementation of radical change.

Reengineering theory comes from management and IS. Management theories about organization design, job design, and reskilling are all used in redesign of work and the organization structure.

First, good management practice dictates that only essential activities be done. To assure this, reengineering assumes an organization level plan for all functions, activities, and processes that accomplish the activities. It also assumes that the plan is actively managed to ensure that all processes directly relate to the organization's mission, goals, and objectives. Nonessential processes, departments, and layers of management are eliminated to streamline, speed, and lower the cost of process performance.

Second, in job redesign, a caseworker approach is preferred to an assembly line approach. **Caseworkers**[1] have increased control, decision making, authority, and discretion. Redesigned, enlarged jobs improve the quality of work life, thus, improving the quality of work.

To satisfy employees and customers, for instance, customer service departments might adapt a caseworker approach to work. In the caseworker approach, employees know the entire process from beginning to end and work independently to service their personal customers. In addition, the caseworker works closely with the marketing and sales force for those same customers. The consequences of caseworkers are great. The customer service agents have reskilled, enlarged jobs that are more interesting. Intrabusiness communications between, for instance, sales and customer service, are improved. External customer relations should also improve because customers have one consistent representative with whom they work.

Enlarged jobs are not a way to squeeze more work out of already overworked people. In the customer service example, initially a clerk does a small number of activities that present a partial view of a large number of customers. In a reengineered job,

1 Hackman [1990].

the clerk does a large number of related activities that present a complete view of a small number of customers. The move is away from an assembly line approach and toward self-sufficient workers or work groups.

The first reengineering improvement for caseworkers comes from job redesign. If the 80–20 rule is applied to most businesses, 80% of the transactions in the business are the norm, and 20% are exceptions. Organizations are typically designed to handle exceptions well. The 80% of their work that is normal tends to take much longer than needed. One goal of reengineering is to increase handling speed and quality of handling for the 80% of normal transactions by an order of magnitude, for instance, by at least 10 times. A second goal is to decrease the number of exceptions to as close to zero as possible. For instance, at Ford, one way to prevent errors in the receipt of goods from vendors was to accept only complete, exact shipments. Any item that did not match an order item caused the entire order to be returned. Vendors got the message quickly that Ford would not accept their shoddy work practices any more and were forced to revise their procedures as well.

Empowerment of the caseworkers comes from job redesign, removal of errors from the process, *and* from the use of any and all information and technologies that assist them in performing their job. Information technologies *enable reengineering*. **Information technologies** (IT) are any technologies that support the storage, retrieval, organization, management, or processing of data. A technology plan and goals should be developed and managed at the organization level.

In addition, **data**, the raw material for information, requires recognition and organizational commitment as a corporate resource. As a corporate resource, data requires the same careful planning and ongoing management as cash-on-hand, office equipment, or personnel. Data must be managed at the corporate level as a key asset of the organization.

To manage and plan for the organization structure, its data, and its technology, enterprise level (i.e., the entire organization is the enterprise) plans must be devised. These plans, or **'architectures,'** provide a snapshot of the current organization. An **enterprise architecture** is an abstract summary of some organizational component's design. The organizational strategy is the basis for deciding where the organization wants to be in three to five years. When matched to the organizational strategy, the architectures provide the foundation for deciding priorities for implementing the strategy.

The organization **process architecture** identifies the major functions of the organization, the activities that define the functions, and the processes that accomplish the activities. Examples of each of these levels are shown in Figure 5-2. It does not detail the procedures for how to do each task.

During reengineering analysis, the entire process architecture is reevaluated for its support in achieving organizational goals. For processes that survive the analysis, the organization is redesigned. Theories of interdependence, linking mechanisms, and organization design are applied to structuring work groups in the reengineered organization.[2] These theories are not new. Rather, theorists and practitioners have talked about them for years with little movement of theory into practice. Over the same years, information technologies matured sufficiently to support the integration and data sharing required of the *information organization*. In the early 1990s, a ground swell of changing companies became an avalanche, with many companies trying to implement the theories using information technologies to support the revised organization.

The second architecture, **data architecture**, identifies the enduring, stable data entities (people, places, organization, events, and applications) that are critical to the organization maintaining itself as a going concern. IS theories of information modeling and information systems planning are used in data analysis. In particular, entity-relationship modeling is used for documenting data and its relationships. Entity-process analysis is used to design subject area databases. Entity-application analysis and process-application analysis are used to define automation requirements. These analyses originated in IBM's

2 Interdependence theory is Thompson's [1967]. Galbraith [1976] and Galbraith and Nathansen [1979] propose linking mechanisms with some organization design. Other organization design work is listed in the references.

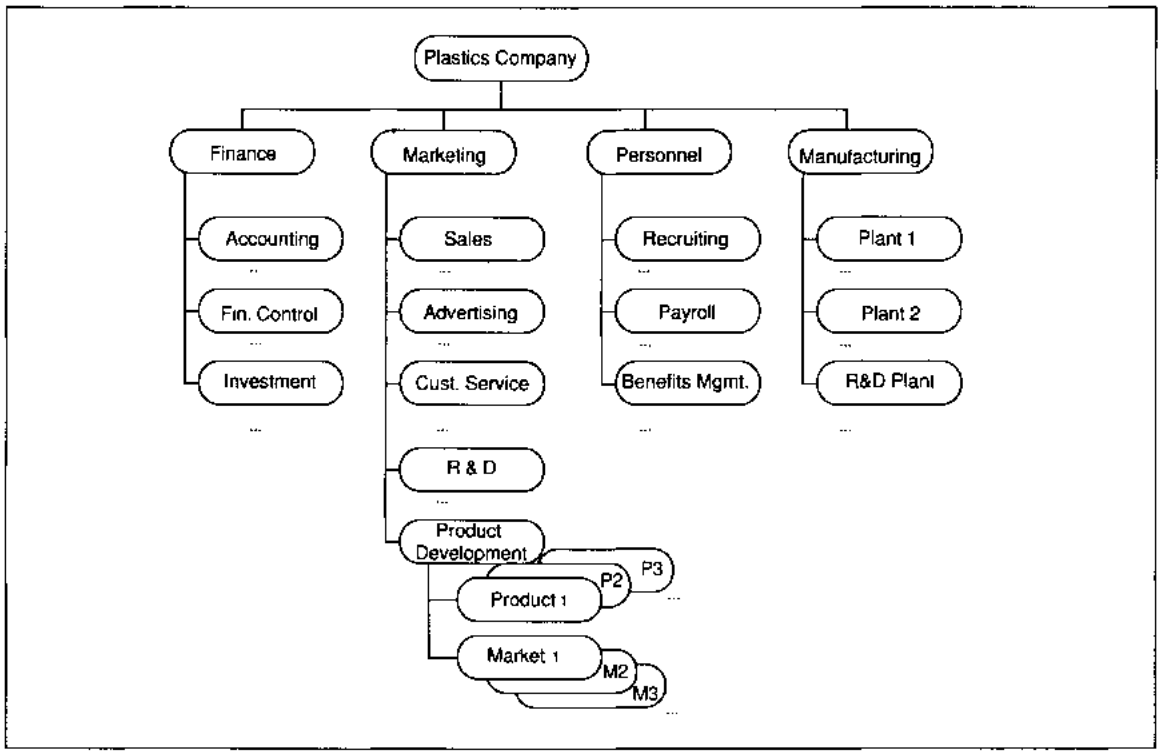


FIGURE 5-2 Sample Process Architecture

information systems planning (ISP) methodology and are expanded in reengineering.

The **network architecture** identifies all locations of work and their communications requirements. It is the basis for deciding telecommunications support.

Finally, the **technology architecture** contains information about platforms [e.g., mainframe, local area network (LAN), or personal computer (PC)], special-purpose technologies (e.g., multimedia, imaging, e-mail) and the locations of each. By mapping the network and technology architectures, organization level technology changes can be identified. New technologies, such as imaging, can be evaluated and positioned to provide the most leverage to the organization.

Successful reengineering is not assured. Necessary conditions, or *absolute prerequisites*, for reengineering include:

1. Management commitment, usually from the CEO or top manager of the organization.
2. Formally articulated organizational mission, goals, and objectives.
3. Full commitment of the reengineering team.
4. Training and support for the reengineering team.
5. The desire to change the organization and its culture.

In addition to the necessary conditions, reengineering assumes the following:

1. *Nothing escapes review.* The reengineering team has as its mission to evaluate the organization, including its structure, jobs, data, processes, and technology. Recommendations in any of the five areas of assessment may be made.

2. Enlarging jobs and empowering job holders as caseworkers rather than as assembly line workers is desirable.
3. Business and IS organizations must become *partners* in the redesign and technology empowerment.
4. In improving quality of processes, elimination of errors via elimination of functions and superfluous processes is desirable.
5. There are no technology constraints. Recommendations will be made without regard to current budgetary, organizational, or other constraints. Implementation planners, based on recommendations and manager's priorities, will attend to constraints.
6. Data shareability is desired. While normalizing data within an application environment minimizes redundancy in an application, *minimizing organizational data redundancy* via data administration and across applications is the *real* goal. Building subject area data bases and providing data access based on need rather than on organization structure is the means to achieving organizationally minimized data redundancy.

This assumption of no constraints may not be realistic in that politics and survival of participants can subvert the desired objectivity in a reengineering project. One of the management challenges in reengineering is to prevent politics from preventing the needed change.

Industry leaders and successful turnaround companies who now thrive provide the motivation for sweeping change. These companies are organized differently from their competition. Industry leaders today tend to have fewer departments, fewer layers of management, and fewer people doing analogous jobs than their competition. Their success is partly organizational and partly cultural. These successful companies succeed because they define their market in terms of what their customers want and demand, *then they exceed those expectations*. Because these companies do not have excess structure, they are flexible to continuously reevaluate what they are doing and how well they are doing it.

Ford Motor Company, for instance, turned around their losing company when introducing their 'Quality is number one' program. They compared their organization to others, including Japanese firms, and found they had many more people performing similar functions. In some cases, like the accounting area, the difference was more than 10 to 1 in numbers of people. Ford threw away the book about how accounting should be done, eliminated parochial interests about where decisions should be made, made data sharing from databases universal, and reduced their staff by over 60%. The result of the extensive changes is happier people with more skills used in a given job. Individual jobs are done faster and more cheaply with almost no errors.

The philosophy of reengineering is to define stakeholders' goals and then exceed them. The philosophy is based on the idea that change can be good. Companies must scan the business horizon and actively change the organization as needed to lower costs, and to improve, speed, and increase the quality of service(s) in meeting its mission. *They must be equally proactive about discontinuing* services, departments, applications, or technologies that no longer relate to organizational goals and objectives. In short, the organization must be proactive rather than reactive about all aspects of its operation.

PLANNING REENGINEERING PROJECTS

Schedules for reengineering projects can be based on several different scenarios. The goal of all scenarios is the same: redesign of organization, jobs, processes, data, and supporting technologies. A secondary goal is that all redesign planning be completed in a short period of time. The short period should be within four months from the time the team is formed until all recommendations are presented to the senior manager sponsor(s).

Reengineering projects can be completed faster or slower depending on several factors. First is the amount of actual time spent by each team member. Ideally, all team members should be relieved of their

current duties and assigned full time to the reengineering effort. In reality, the best managers, who you want on the team, also are the most needed to run the current business. So, part-time or short duration full-time commitments might *have* to suffice.

In all cases, one to four senior IS staff (i.e., consultants, senior analysts, software engineers, or project managers) are assigned full time to the project. Much of the work performed during the reengineering project is identical to that performed as part of an information systems planning exercise. IS staff who already know ISP only need to learn several types of matrix analysis and organizational design to be fully capable of performing the reengineering work.

The second major factor in determining the amount of time is the size of the organization being analyzed. A 100-person, five-department organization can be analyzed easily within four months. A 10,000-person, 200-department with four hierarchic levels can also be analyzed within four months, but requires more people and more discipline to the team. A good rule of thumb is to have one person for every 10–15 departments or every 100 jobs.

Four months is the time most authors recommend for completion of the entire reengineering project, from inception to development of the implementation plan. The actual pilot testing and implementation of the changes might take several years to complete. There are several good reasons for a short time schedule. First, managers cannot suspend their work indefinitely and run a company, too. If several people are allocated full time it drains the management resource. Second, with a mentality oriented to quarterly results in the United States, most managers will not wait longer than that to prepare for change. Third, the project is bound to be known throughout the organization soon after it begins. When reorganizations are imminent, work is replaced by gossip and worry. The shorter the time of the reengineering study, the less lost work to the organization.

When the end date is mandated, the team does the amount of work they can accomplish within the time constraint. This approach to work is called 'level of effort.' With a **level of effort** approach, the team works at capacity up to the deadline and, what does not get done, does not get done. For large projects, then, the level of effort approach assumes an incomplete analysis.

The assumption here is that error-prone and bottleneck processes are the targeted activities. While a high-level description of the entire enterprise is possible, only the problem activities are actually in the level-of-effort study.

Scenarios for three levels of user manager participation are provided in Figures 5-3 through 5-5. Figure 5-3 shows a short burst of participation, similar to a joint requirements planning (JRP).[3] In this scenario, users and analysts are trained and go off-site for an intensive 4–8 days (depending on the size of the organization) of requirements, data, process, and entity-process analysis. An alternative that minimizes the amount of time managers are absent from work is to hold the JRP meetings over one or two weekends. More than 90% of the data gathering can be completed using the JRP approach. In this scenario, most of the analyses are done by the full-time project staff, but are presented for review and decisions to the user-team participants. *In no case do the IS staff make the decisions and recommendations alone.*

The second scenario assumes constant part-time participation over time (see Figure 5-4). In this scenario, user managers are available for meetings, interviews, and analysis sessions 1–3 hours each day. They must be committed to participating and must not waver from participation, or the project will falter. Notice the dotted lines for all activities. The dotted lines imply a part-time, longer activity. The full-time IS staff actually do most of the legwork, interviews, and preparation for analyses. But, once again, the decisions are made by the user managers, not the IS staff.

The final scenario assumes full-time commitment for the duration of the project (see Figure 5-5). With full-time users and full-time IS staff, the length of the project can be as short as three weeks and, for large organizations (e.g., 1,000 people, 50 departments), as long as 16 weeks. Table 5-1 shows the major tasks and activities with expected percentages of effort for each task.

3 JRP is an innovation of IBM Corporation. It is fully discussed in the introduction to Part II.

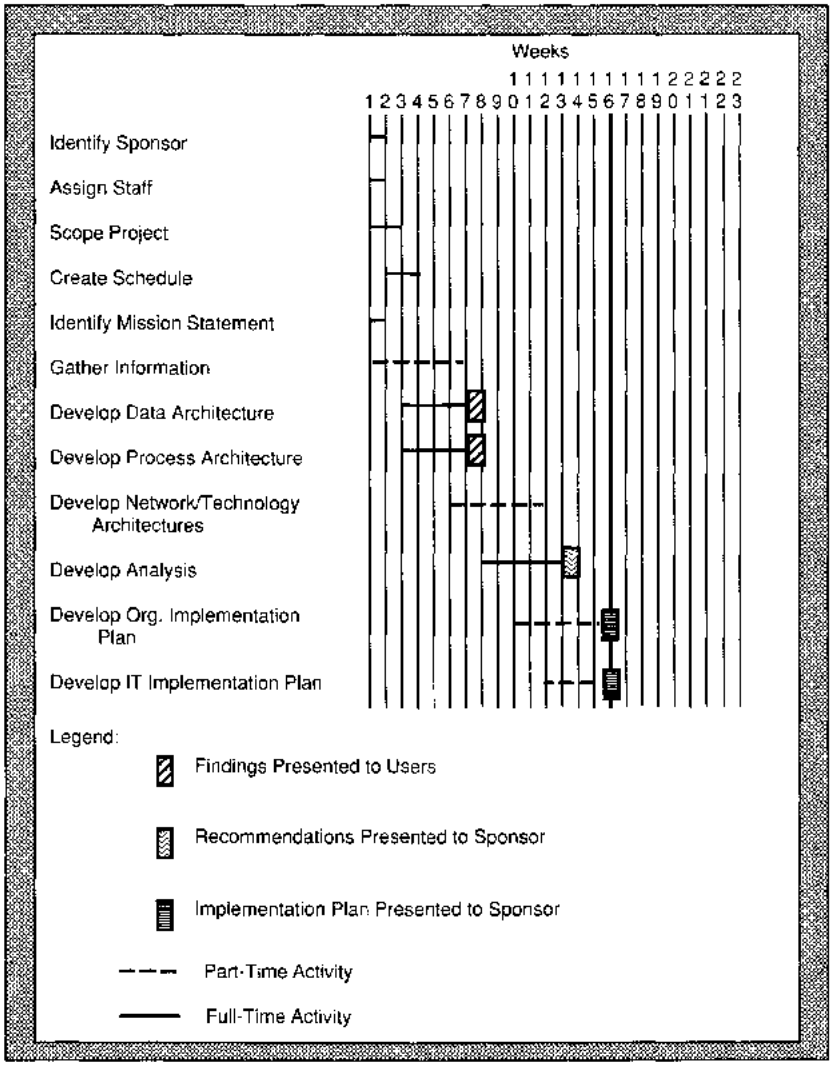


FIGURE 5-3 Reengineering with Part-Time Users

REENGINEERING METHODOLOGY

Reengineering is most easily done within the scope of information system planning (ISP) projects. With a greater balance of process and data analysis, and several additional activities, reengineering uses the same information as the ISP. The major steps and their results, type of questions asked, and analyses are listed in Table 5-1. The steps are summarized in Figure 5-6 which shows a significant amount of overlap between steps. The times allocated to the tasks are as individual stand-alone activities and do not reflect the amount of actual time spent on the step. For instance, the architectures are all allocated one week. But they are preceded by activities of four weeks during which they should also be developed.

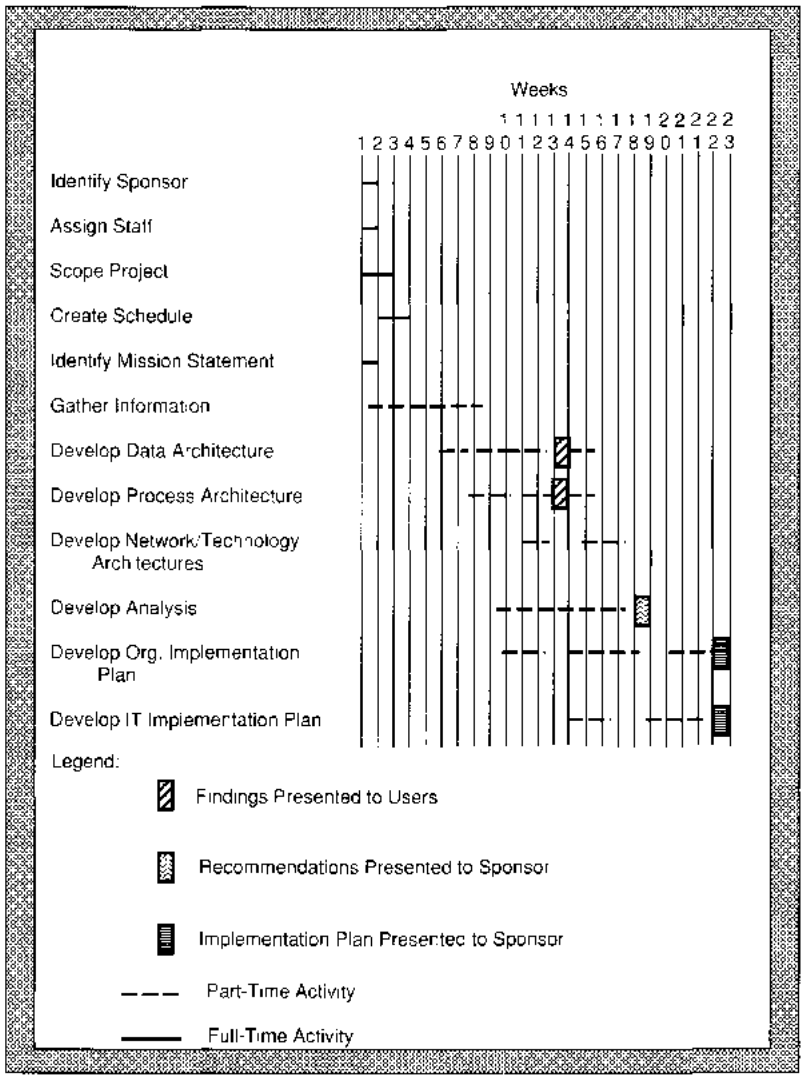


FIGURE 5-4 Reengineering with Continuous, Part-Time Users

All of those particular steps are iterative and require three to five weeks to complete. A detailed description of each reengineering step follows.

Identify Project Sponsor

The first step of reengineering is to enlist or be enlisted by the project sponsor. The **project sponsor** is a *senior* level manager who will pay for *and* champion the project. A **champion** is an individual with commitment, enthusiasm, credibility, and influence who can act as a 'cheerleader' for the project and its outcomes. The sponsor is the overall project manager for the reengineering project and must have the authority, fortitude, and desire to change the organization and its work, based on the recommendations from the reengineering analyses.

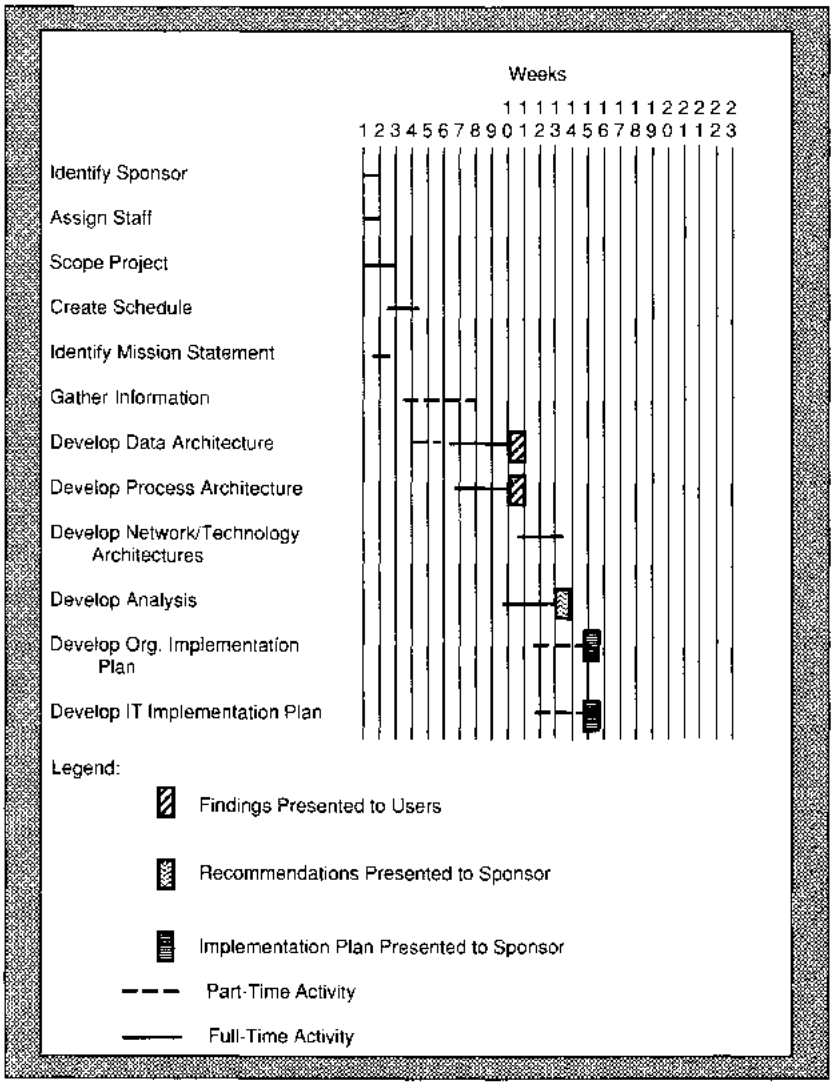


FIGURE 5-5 Reengineering with Full-Time Users

Assign Staff

Three or four user area, senior, or middle managers should be assigned to the reengineering project for a period not to exceed four months. At least one month of the initial commitment should be full time; the remainder of the work may require only part-time commitment. Two or three senior IS managers, or SEs, or data administrators, or consultants should be assigned to the project full time for its entire duration.

All team members should attend a reengineering workshop or class *together* to fully acquaint them with the techniques and goals of the activity. The individuals assigned must have commitment to this work. They *must* be senior enough and good enough at their own jobs to have instant credibility within their organization. Without both of these

TABLE 5-1 Percentage of Reengineering Effort by Task

Activity	% Effort
Obtain sponsor	N/A
Initiate project	N/A
Assign staff	N/A
Scope project (Concurrent with next two activities)	2–5% 2 Days
Develop schedule	2–5% 3–5 Days
Identify mission statement	2–5% 1 Day
Gather data	20–25% 3–4 Weeks
Develop process architecture	6–10% 3 Days–1 Week
Develop data architecture	6–15% 3 Days–1 Week
Develop and analyze entity/process matrix	20–25% 3–4 Weeks
Develop implementation plan	20–25% 3–4 Weeks
Develop technology architecture	6–10% 3 Days–1 Week
Total duration	100% 12–17 Weeks

requirements, the target of four months for the total effort is doubtful.

Scope the Project

The key criteria for properly scoping a reengineering project are data self-sufficiency and user commitment. **Data self-sufficiency** is defined as 70% (or more) of data used in performing the business functions that must originate within the subject organization. The goal of **scoping** is to identify a group of departments that create their own information and are not dependent on other departments for data to do their work. Control over data creation equals data self-sufficiency.

The second criteria is user commitment. **User commitment** means that the managers participating in the reengineering project must be committed to changing the organization. This is not as difficult as it might sound. Few people enjoy their job when they know it is inefficient and hampered by ineffective organization or systems designs. When the best managers in an organization that needs change are assigned, they become enthusiastic about the prospect of designing the work groups to fit the work. Because their positions in the company are not at risk, there is little reluctance to participate.

Determining data self-sufficiency requires development of a quick entity-relationship diagram (ERD), process hierarchy, and entity/process matrix. The results should be about 80% complete and address the major entities and processes. The analysis of the matrix is to determine where data are created, nothing else. If data are not created within the organization, the amount of data and the creating (or originating) organization are identified and added to the study. In addition, the amount of data for all entities created within the organization must be identified to determine the percentage of data self-sufficiency. The percentage is derived from the formula shown as Figure 5-7.

The inputs to the formula (I) identify a count of transactions or other work items generated within the target reengineering organization. The outside work (O) represents a count of transactions or other work items coming into the department from elsewhere in the organization. Outside work is not subject to review or error reduction, and the goal is to keep it to a minimum in the study. In Figure 5-7, the target organization generates 75% of its own data and is, therefore, data self-sufficient enough to benefit from reengineering.

Less than 70% data sufficiency implies too narrow a scope because of too great a data dependency on outside organizations. Lack of data self-sufficiency artificially constrains (or may mask potential) elimination of errors, organizations, or levels of management that are not needed. If the scope is too narrow, the analysts present the information to the sponsor and request a broadened scope to include the information-creating organization(s).

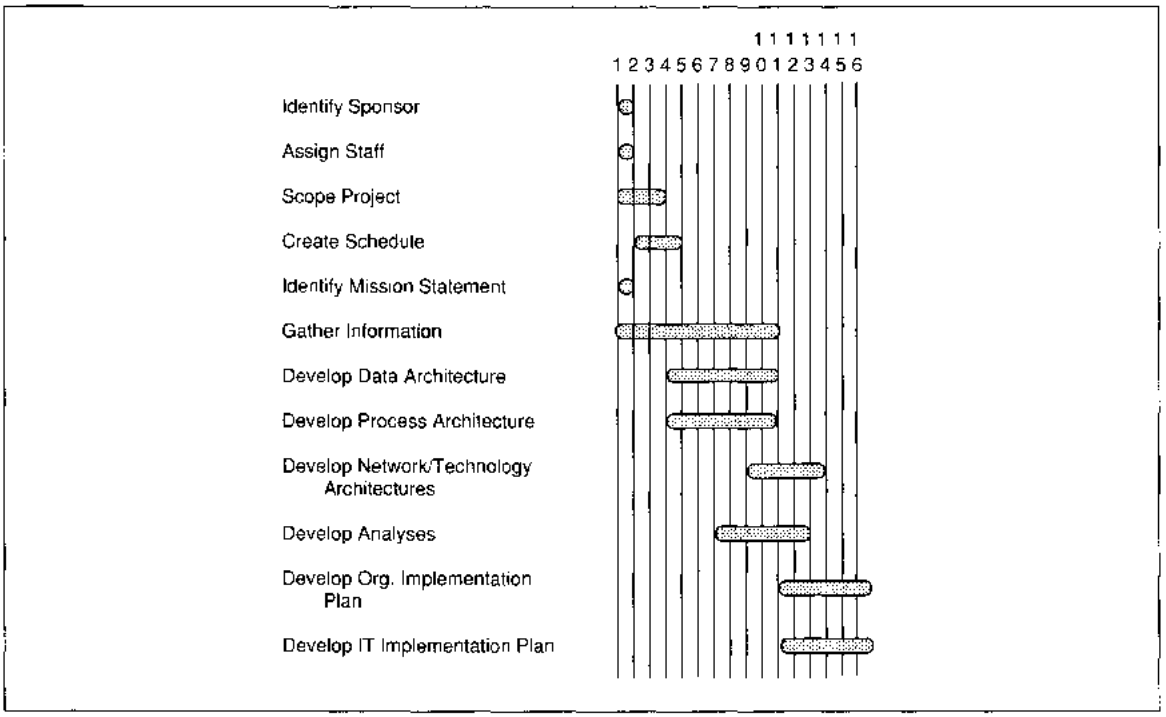


FIGURE 5-6 Overlap Between Reengineering Tasks

Formula:

$I / (I + O) * 100 = \%$ DS

Example:

I = 750,000 records

O = 250,000 records

750,000 / (750,000 + 250,000) * 100 = 75%

Legend:

I = Data generated inside the reengineered departments

O = Data generated outside the reengineered departments that is required for them to do their work.

DS = Data Self-sufficiency

FIGURE 5-7 Formula for Determining Data Self-Sufficiency

For instance, reengineering might target an accounting function. About 90% of the information in an accounting function originates from other organizations within the company. Without also including those functions in the reengineering study, changes that address, for instance, data accuracy or work location problems, are unlikely to be successful.

The scope might not be complete until the next several tasks are partially complete, due to a lack of information about data and responsibilities. Therefore, the initial scope should be reexamined before completion of the entity/process matrix analysis to reconfirm data self-sufficiency.

Create a Schedule

The team creates a schedule for the entire reengineering project not to exceed four calendar months. Each step has an estimated range of time that should

be allotted as a percent of the project total shown in parentheses (see Table 5-1). Each task is assigned to a team member who is held accountable for the work.

Identify Mission Statement

Identify the mission statement for the organization with quantified goals for measurement. A **mission statement** is a short paragraph summarizing the overall purpose of the organization. The details of the document should include goals and objectives, along with determinants of success (i.e., critical success factors) for each, with required data for measuring the extent to which the goals are met (i.e., means-end analysis).

If the organization has no mission statement, or has no quantified goals and objectives, do not attempt to develop these for the organization. Disband the reengineering group and have the managers work on perfecting the mission, goals, and objectives before reconvening the reengineering effort.

Goals should have a three- to five-year horizon and should be specifically measurable (i.e., quantified) (see example in Figure 5-8). There should be at least one goal for each sentence in the mission statement. Goals relate to **stakeholders** who are people affected by the outcome. Some stakeholders include customers, vendors, stockholders, owners, and boards of directors.

Increase the number of new customers by 5% each year for five years.

Increase sales to existing customers by 8% per year for five years.

Increase number of rentals per store visit by providing an expert system to assist in selecting movies for rental.

Reduce sales support expenses by 10% in one year.

Reduce overhead expense by 10% each year for two years.

FIGURE 5-8 Example of Organization Goals for ABC Video

Identify critical success factors for determining that goals are met. A **critical success factor** (CSF) defines some essential process, data, event, or action that must be present for the outcome to be realized. For instance, if the goals in Figure 5-8 are desired, a CSF might be *Ensure that sales staff are fully trained in locating movie information.*

The last step of critical success factor analysis is to decide what information is required to measure goal success. In the example, goals relate to sales. The CSF also relates to 'training.' Success measures for sales and for sales staff knowledge of how to find movie information are required. Periodic evaluation with training for ill-informed sales staff is one way. Management needs to know evaluations that have taken place and misinformed staff who have been retrained. If the same person(s) are being retrained, management intervention might be warranted.

Intangible goal measurement is just as important as tangible goal measurement. An intangible goal might be increased customer satisfaction. To measure this, an outside polling company can canvass customers and ask different recall or direct questions about their satisfaction with the company's services. Recall-type questions are of the form: Which vendor that you work with has the best customer service? Direct questions are of the form: Rate the customer service of company *x*.

The next step is to link each CSF, critical information measure, and goal to functions, processes, technology and data in the organization. If this step cannot be completed yet, defer completion of this task until information gathering is complete. If new entities or processes are defined through CSF analysis, add them to the list for reengineering analysis.

Gather Information

Gather information on processes, data, process problems, quality problems, data problems, accessibility to data, timing of work (e.g., lags that cause idle time), time constraints for performance, and problems related to timing. A sample list of questions are:

What are the major steps to accomplishing each process?

Which processes/procedures are required to accomplish the mission, goals, and objectives?

What data are used as input? Where does it come from? Who enters or creates data? uses or retrieves data? changes or updates data? deletes data?

How is the input transformed by the process to produce the results? That is, what do you do when you do your job?

What data are passed between processes? What is the current storage media for the data (e.g., computer, fax, paper, verbal, memo, etc.)?

Are the different types of data that you need for your job used sequentially or in parallel? Could you describe the procedure?

Where are time lags in your job during which you are waiting for someone else to give you work or information? How do you deal with these lags?

Where are quality problems? How do you deal with errors? What is the source of each type of problem? Where (in which process or outside organization) is each problem detected? Where are quality problems within the procedures you use to do your job? How do you try to guard against these problems?

What would you do differently if you could design your own job? How might computer technologies help you? Suppose you have all the new computer and other technologies available for your job's use. What technology would you use and how?

Information might come from forms, screens, reports, phone messages, fax messages, automated applications, policy and procedure books, and so on. The people actually doing the work provide this information.

Most information is obtained through an interview format. Interviews should be individual or in small groups (groups should have members who share common goals to minimize political conflicts). All middle and senior managers for the organization should be interviewed in addition to representative white-collar, blue-collar, or clerical staff. Treat the sessions as fact-finding, not fault-finding. Address all the topic areas for which information is required.

If you think you are getting incomplete or false information, cross-check, or triangulate, the information by asking the same questions of multiple sources. For instance, Manager *A* says his major problem is caused by erroneous data received from Manager *B*'s area, and Manager *B* did not identify the problem in your first discussion. Return to Manager *B* and reinterview him or her, specifically discussing data quality as a problem identified by the other area.

To validate the complete findings, make a group presentation to all interviewees for final confirmation that the information is accurate and complete.

Summary of the Architectures

In this section, we expand Zachman's[4] **information systems architecture (ISA) framework** to describe how to express the reengineering information in terms of architectures. The four architectures of interest in reengineering are data, process, network, and technology. First, we define the framework and information presented at each level. Then, reengineering information is translated into the four architectures.

Conceptual Levels of the Architecture

The information systems architecture (ISA) describes distinct architectures relating business context to application context. The five levels are described in general terms below and are summarized in Figure 5-9. Only the first two levels, scoping and enterprise analysis, are used in reengineering.

Information systems application development and organizational redesign are complex engineering activities that are similar to constructing a building or an airplane. The ISA describes the intellectual levels of detail needed for complex engineering

4 John Zachman [1987]. Zachman's architecture discusses data, process, and network. ISA does not yet include a technology architecture. This idea is from reengineering consulting which requires a view of the technology as a basis for technology redesign.

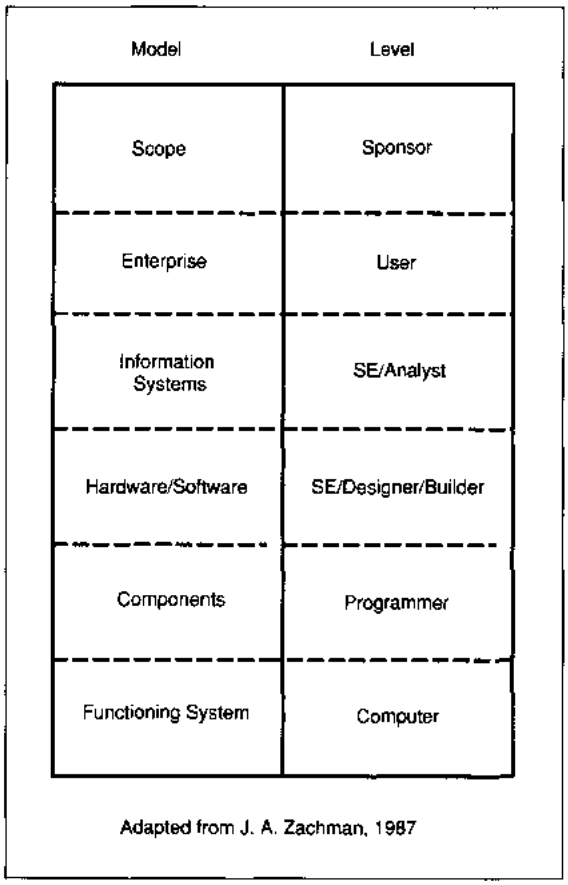


FIGURE 5-9 Conceptual Levels of the Architecture

activities. Then, it links them to data, processes, networks, and technologies—the components of computer applications and reengineering.

In all three businesses—aerospace, architecture, and systems—we begin with a sponsor's idea of what the *item* being built should look like. This is the *scope of the reengineering project* that defines what is in and what is not in the problem. If the *item* is a house, for instance, users talk about a two-story colonial with four bedrooms, three bathrooms, and a fireplace in the family room. For reengineering, the sponsor targets departments doing order processing and customer assistance. In this case, the *item* is the order processing department.

The user talks to an expert to describe his or her view of the item, and the expert translates the user's idea into an enterprise level, logical description of the item. **A logical description** is one that lists *what* is done without saying *how*. The item begins to take more shape and be less specific. The description of the item is somewhat more abstract. For the house, we now have a family room of 13.5 feet by 16 feet with a cathedral ceiling that is open to the kitchen with entries to the foyer and living room. For reengineering, we have an order entry process that includes order receipt, order change, order inquiry, inventory allocation, creation of shipping papers, movement of goods, invoice creation, and an interface to accounts receivable. Both of these descriptions are significantly more detailed than the first. Neither description is complete. We still don't know the type of windows in each room, for instance. Nor do we know, for reengineering, whether the work is automated, how an order is processed, or whether any of the steps can be done together. In both cases, the details are unimportant *at this level*.

At the next level, the expert translates the logical, enterprise view of the item into terms and information that are useful to the analysts of the item. So, the expert (or different experts) translates the enterprise view into a logical information systems design. The logical design still describes *what* the item will do, but in more detail than before, and in terms understood by application developers. In reengineering, the logical design is very specific about the item, its parts, and how they fit with the other items and their parts. In our order processing example, we would know what data, what fields, what processes and their details, timing of processing, what applications and technology are currently used to support the work. Designers can review the detailed logical design and see *how* it can be automated.

In the next step, designers review the logical design and translate it to specific materials, thus creating a technology-based model. In reengineering studies, this translation takes redesigned work, work groups, departments, data, and technology as inputs. The inputs are translated into database schemas, applications' design specifications, network designs, and specific hardware/software platforms for supporting the redesigned work. In the order process-

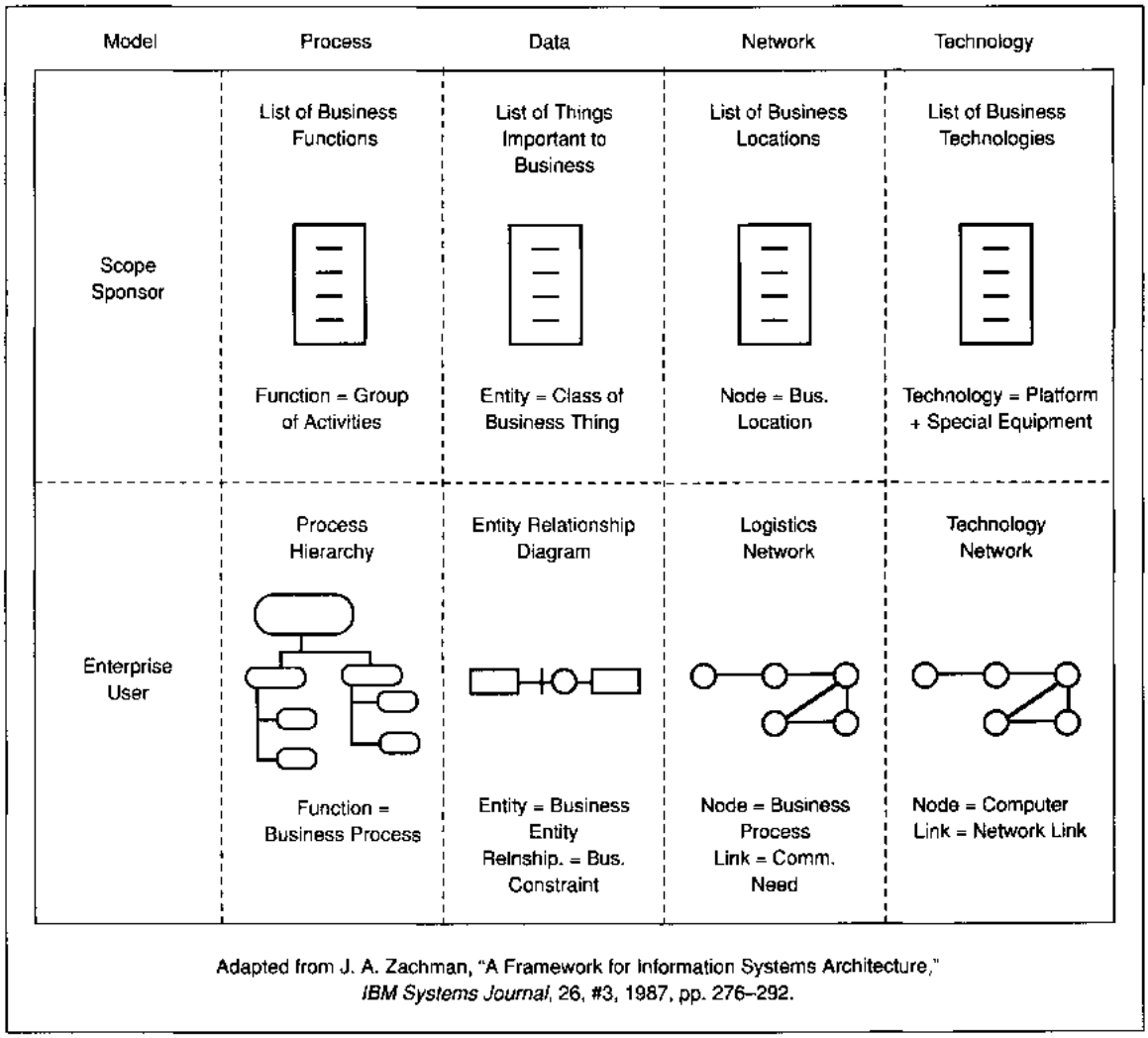


FIGURE 5-10 Reengineering Levels and Architecture Domains

ing example, the requirements specification would be translated into program specifications for specific hardware, software, and language.

Finally, at the lowest component level, schemas, specifications, and technology plans are implemented and translated into working computer components.

Only the scope and enterprise models are discussed in this chapter; the other levels are too computer-oriented and not appropriate for reengineering.

Domains of the Architecture

The conceptual domains apply to four organizational domains: data, process, network, and technology. A **domain** is an area of interest. The **data domain** defines the entities of interest to the target organizations and the interrelationships between them. The **process domain** describes the functions, activities, and processes of the target organizations, without any identification of how they are accomplished. The

EXAMPLE 5-1

ABC VIDEO MISSION STATEMENT

The mission of ABC Video is to develop and maintain quality relationships with customers, vendors, and employees.

For customers, we provide a large selection of current and classic videos for rental at a fair price. We assist them in selecting videos with courtesy, service, and a minimum of bureaucracy.

For vendors, we order videos with reasonable lead times and timely payment of bills.

For employees, we provide a congenial atmosphere with comfortable, clean, and safe working conditions for a fair wage.

Process	Data	Network	Technology
Video Selection	Customer	Location = 1	None
Service Request (i.e., process rental)	Video Rental Vendor	(inferred)	
Order Creation	Order		
Accts. Payable	Video (= goods in inventory)		
Payroll	Employee		
Personnel			

network domain describes the organization from a geographic perspective. The **technology domain** describes the organization of work from a technology platform perspective.

Translating Information into Architecture

There are two levels of architecture we describe in this section for reengineering: the scope and the enterprise model.

Scope

In reengineering, we assume that the mission statement fully expresses the scope of the organization. The mission statement is translated into network technology, process, and data scopes to initiate the reengineering effort. Example 5-1 shows a mission statement for ABC Video and how it might be translated to identify the scope of the four domains. At the scope level, we should know the major entities of interest to the organization and the business functions and their activities.

The network and technology domains may or may not be mentioned in the mission statement. The sponsor or user participants define these when they are not in the mission statement. The **network scope** defines the location of work for each activity. The **technology scope** defines technology platform by location. Because ABC has only one location and no technology, it is a simple example. Another example here is for a plastics subsidiary of a large international company. Figure 5-11 shows existing hardware platforms listed by location. In Figure

Hardware Platform—Scope

Location 1
- Mid-Size Computer
- LAN 1—25 PCs
- LAN 2—15 PCs
- LAN 3—42 PCs

Location 2
- LAN 4—23 PCs

Location 3
- Mid-Size Computer
- 5 Stand-alone PCs
- 3 CAD/CAM Platforms

Location 4
- Mid-Size Process Control Computer
- LAN—25 PCs
- 1 CAD/CAM Platform

Location 5
- Mainframe

FIGURE 5-11 Plastics Company Hardware Platform Scope

Activity by Location

Location 1
- Finance—3 products at this location
- Accounting—All products
- Customer Service—All products
- Product Management—3 products
- Personnel/Payroll

Location 2
- Finance—2 Products at this Location
- Product Management—2 Products
- General Manager

Location 3
- R & D
- Manufacturing Setup

Location 4
- Manufacturing Plant

Location 5
- Corporate Headquarters

FIGURE 5-12 Plastics Company Activity by Location

5-12, the activities from the process hierarchy are reused and identified by location.

At this point in reengineering, if the mission statement were suspect in its completeness, a stakeholder analysis might be developed to determine if all constituents of the organization are represented. If they are not, the mission statement would be redrafted to include missing constituencies. While this redrafting takes place, the reengineering study ceases operation. A stakeholder is any person who interfaces with, works for, or otherwise is impacted by an organization. Stakeholders include the owner, managers, employees, suppliers, customers, creditors, government, community, and competitors. Ideally, representative stakeholders from each group should review the strategy and offer suggestions for improvement.

When stakeholders are identified, the goals of each stakeholder are defined and related to the organization's functions and strategies. If a goal does not match a current function or strategy, management determines if the goal will, in fact, be met. The goals are translated into strategies which, in turn, are translated into work. The intention of stakeholder analysis is that rational, reasonable goals should have both strategic and organizational functions that relate to the attainment of goals. Even if goals are omitted from the final strategy, at least all stakeholders and their desires are identified and considered.

Enterprise Models

At the enterprise level, the user managers work with information systems (IS) project representatives to define business areas in logical terms. The principle business modeling activities include entity-relationship diagrams (ERD) for data, functional decomposition diagrams for work processes, a network diagram of process communication needs, and a technology network diagram

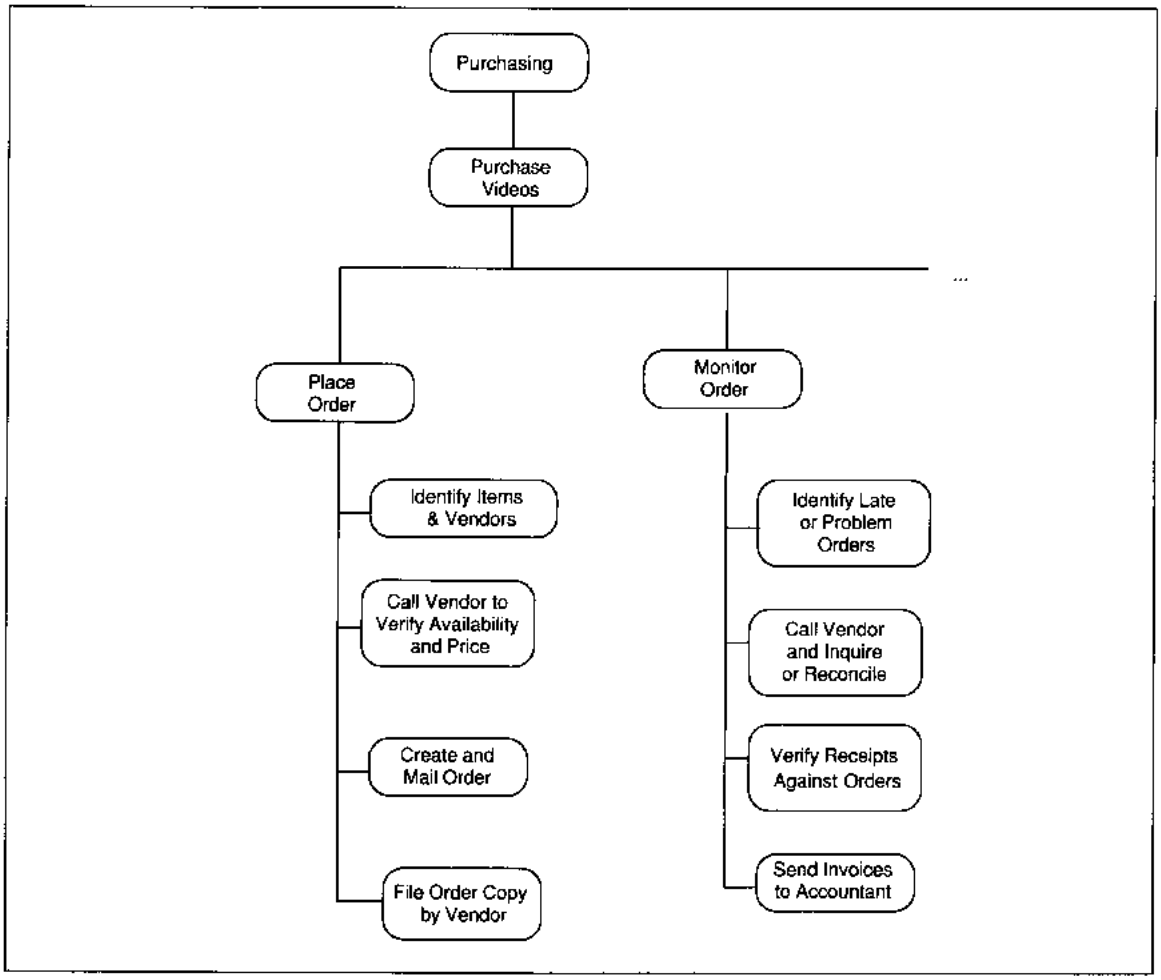


FIGURE 5-13 ABC Video Process Hierarchy

showing technology deployment. The ERD documents major data types and their interrelationships. The functional decomposition identifies business functions and their component activities and work processes. The network architecture shows the location of work and intraorganizational communication requirements. The technology architecture shows the hardware platforms by location and the telecommunication linkages between them. All four architectures are developed piecemeal as information becomes known. (ERD and functional decompositions are discussed in detail in Chapter 9 and are only summarized here.)

PROCESS ARCHITECTURE. Process architecture development is concurrent with data gathering. The time recommended in Table 5-1 is for completion and validation of the information. The decomposition first identifies business functions, then the component activities and their processes. Figure 5-13 shows an example. A **business function** is a group of on-going activities that accomplish some complete job that is within the mission of the enterprise. Functions are general and fit most organizations. For instance, accounting and personnel are found in most organizations regardless of industry or business type. At the next level of detail, an **activity**

defines one or more related procedures that accomplish some task. For accounting, for instance, activities might be monthly close, maintaining chart of accounts, or daily transaction processing. At the lowest level of detail for this diagram, a **business process** identifies the details of an activity, *fully defining* the steps taken to accomplish the activity. Business processes within an accounting monthly close might be gathering information, validating information, performing initial analysis, and so on.

The steps to developing a functional decomposition diagram include:

- Identifying the functions of the target organizations
- Interviewing the representatives from each area to identify the activities performed for each function
- Further identifying the processes for each activity

During the decomposition process, business problems are identified by the interviewees. The problems are prioritized by the users with the reengineering team in order of their significance to the organization's quality and function. Usually the number of major problems to be identified is fixed and between five and ten. Without a limit, the problem findings could overwhelm the analysts. Also, having the number of major problems fixed requires users to reach consensus about the seriousness and scope of problems.

DATA ARCHITECTURE. This activity is concurrent with data gathering. One week of extra time is recommended to allow completion and validation of information. The data architecture is defined in an entity-relationship diagram. An entity is some person, object, concept, application, or event from the real world about which the organization maintains data. A relationship is a mutual association between entities.

For instance, a customer creates an order. *Customer* and *order* are entities; *create* is their mutual relationship. Figure 5-14 shows a basic ERD that summarizes this relationship. ERDs can be much more elaborate and include the number, or cardinality, of the relationship, and information about

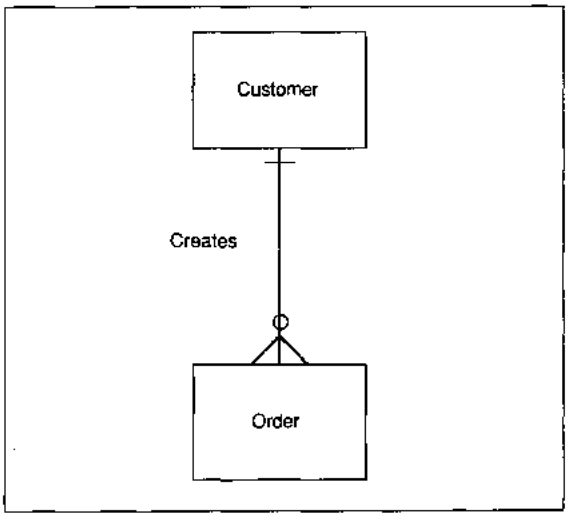


FIGURE 5-14 Sample Entity Relationship Diagram

whether or not the relationship is required. Cardinality identifies one-to-one, one-to-many, and many-to-many relationships. Each customer can have many orders; therefore, this is a one-to-many relationship. So in this ERD the cardinality is one-to-many. The *many* side of the relationship is shown with 'crow's feet' on the diagram. Orders don't come from thin air; there must be a customer to have an order. Conversely, customers are not *required* to always have orders. Therefore, *customer* is required, and *order* is optional in the relationship as signified on the diagram by the short bar and small oval, respectively.

The steps, then, to developing an ERD are:

- Identify data entities, including new entities required to attain and name organization goals
- Link entities to show their interrelationships
- Define relationship cardinality and the required/optional nature of relationships

NETWORK ARCHITECTURE. The enterprise level of network architecture defines organization activities from the functional decomposition performed at each location and communications requirements between them. The architecture

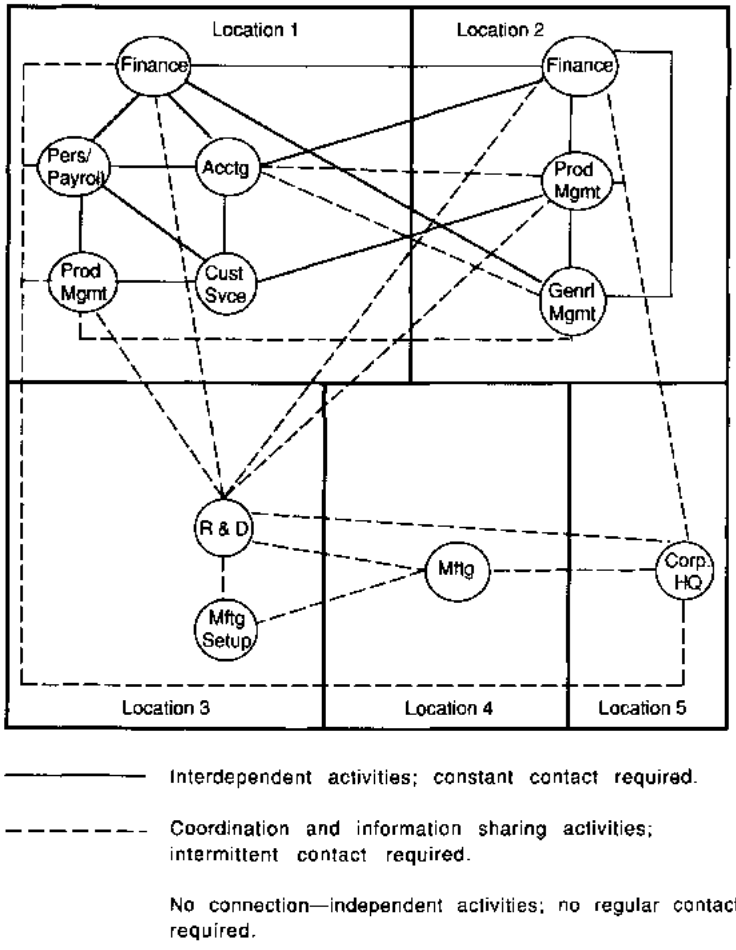


FIGURE 5-15 Plastics Company Network Architecture

described in this section is of the current organization. During reengineering, if the changes recommended affect the locations of work or the activities of work, then the network architecture is redrawn to mirror the recommended organization. When the changes are presented to the sponsor for approval, the old network and recommended network architectures should be contrasted to highlight the changes.

The process hierarchy defined functions, activities, and processes. The network architecture *could* define any of these levels. For ABC Video, we would choose the function level because there is only one work location. For the plastics company example (see Figure 5-15), the activity level is chosen because functions located in more than one place may not include the same activities at all locations. Using the activity level gives a further level of detail, and accuracy, to the work. If the company were very decentralized and diverse, the analysis *could* be at the process level.

For the architecture, each activity is placed in a

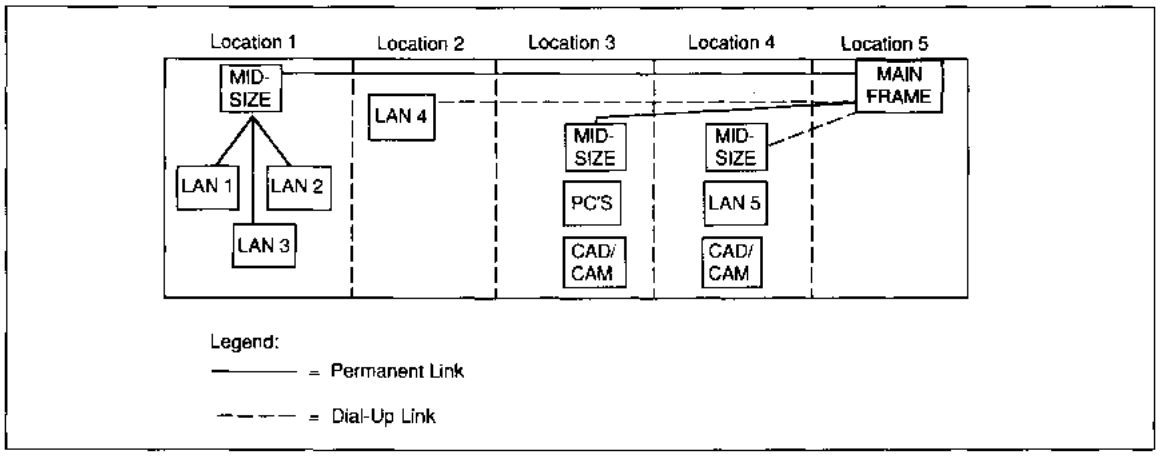


FIGURE 5-16 Plastics Company Technology Architecture

circle within a square identifying a location (see Figure 5-15). The circles are connected when the activities require communication to complete their work.

TECHNOLOGY ARCHITECTURE. The technology architecture creates a network diagram of existing technology at each location using a network technique similar to the network architecture (see Figure 5-16). Then, the technology platforms are connected with lines to show telecommunications linkages between them. Dotted lines are used to show dial-up linkage. Solid lines are used to show permanent connections. At this level, other special hardware, such as imaging, CD-ROM, or technologies such as ISDN, are connected to the platform to which it is attached.

Like the network architecture this is a snapshot of the current technology deployed throughout the organization. If the recommendations for the redesigned organization eliminate or change locations, a second technology architecture is created to depict the new view of the organization.

At this point, the team is complete in their data gathering. The team conducts a group meeting with all previously interviewed individuals to summarize their findings and present the diagrams. The purpose and sole focus of the meeting is to verify the accuracy of the information presented. No further analysis, and no suggestions on the analysis, should be discussed at this meeting.

Architecture Analysis and Redesign

The analysis uses a series of matrices matching the architectures to redesign the organization, its data, applications, and technology infrastructures. The process and data architectures are the basis for the organization and data design. The current applications are mapped to the redesigned organization and data to recommend changes to the application environment. The technology and network architectures are analyzed to recommend telecommunications and technology infrastructure changes that best meet the enterprise's goals. These analyses are discussed here.

Organization and Data

A process called affinity analysis is used to analyze the data and processes. Think of this as normalizing data across the organization. **Affinity** means 'attraction' or 'closeness.' **Affinity analysis** clusters processes by the closeness of their functions on data

Entities =	Purchase Order	PO Item	Vendor Item	Inventory Item	Vendor
Processes =					
Identify Items & Vendors			R	R	CRU
Call Vendor to Verify Avail/Price			RU		RU
Create & Mail Order	CRUD	CRUD	CRU	R	R
File Order Copy by Vendor	R	R			
Identify Late & Problem Orders	R	R	R	R	RU
Call Vendor & Inquire on Order	RU	RU	RU	R	R
Verify Receipts against Order	RU	RU	RU		
Send Invoices to Accountant	RD	RD			

FIGURE 5-17 ABC Video Data/Process Matrix

entities they share in common. Because the average data/process matrix has about 400 entries, affinity analysis is best accomplished through an automated tool, such as ADW™.[5]

A matrix of processes from the process hierarchy diagram and data entities from the entity-relationship diagram is created. The processes are written in rows down the left side and data entities across the top (see Figure 5-17). Use the lowest level processes, such that all elemental processes for the organization and application area are present. When writing the process name, append a prefix to identify the activity and function from the hierarchy diagram.

In each cell, identify the functions each process is allowed to perform on data. Possible functions are create (C), retrieve (R), update (U), and delete (D). One or more of the letters, as defined by the current organizational responsibilities, are entered for each entity. This matrix gets its nickname from those functions; it is a **CRUD Matrix**.

Affinity analysis relates processes by their responsibility in *creating* shared entity information. The create responsibility for 80+% entities shared between processes shows high affinity. An affinity matrix is iteratively refined by affinity groups or processes with entity creation responsibility. In a typical 20×20 matrix with 400 cells, five to seven affinity clusters will emerge. Affinity clusters may contain processes from several current organizations; organizational location of responsibility is not of interest in this exercise.

Several clusters may overlap. This is normal and not a cause for worry. If only one cluster emerges, clustering continues with analysis of update responsibility, and, if necessary, delete and retrieval responsibility. When a reasonable number of clusters emerges, the next step begins. A reasonable number may be one to five clusters for a small organization, such as ABC, or six to nine for a large organization.

5 ADW is a trademark of Knowledgeware, Inc., Atlanta, Ga.

Figure 5-18 shows affinity clusters for ABC. A first analysis of create responsibility would place *Create & Mail Order* in a group and *Identify Items & Vendors* in a group without classifying the other entities. The final clusters shown in the figure emerge after also analyzing update, delete, and retrieval responsibility. The lines highlight the clusters and simplify diagram interpretation; they do not necessarily include all actions in the clusters. Notice that the *Call Vendor to Verify . . .* process overlaps both clusters. It is placed in the second cluster because it also updates *Vendor* information.

The next step is to analyze organizational adequacy. Each process is individually analyzed first to ensure process-goal correspondence. If the process is specifically tied to the organization goals, objectives, and mission, mark it for retention. If the process is *not* tied to the organization goals, objectives, and mission, either link it to goals or objectives, or mark it for elimination.

Next, for processes that are candidates for elimination, determine if they also create, update, or delete data. What is the relationship of this process to 'close' processes? Is it in a sequence with other processes? If so, can those processes take on its data responsibilities (thus enlarging the scope of the process)? If the eliminated process also stands alone, where else is the data used? If the answer is nowhere, mark the data for elimination. [Plan to return to the individual(s) who identified either the process or the data to confirm that you have not missed some information linking the process or data to the mission.] If data is created by the process marked for elimination, but updated and deleted elsewhere, can the other processes assimilate data creation? What other information will those processes now need in order to be able to create the entity? Ask similar questions for updating and deleting the data.

Next, analyze the current organization design. First, is each data entity created only in one process?

Entities =	Purchase Order	PO Item	Vendor Item	Inventory Item	Vendor
Processes =					
Create & Mail Order	CRUD	CRUD	CRU	R	R
Call Vendor & Inquire on Order	RU	RU	RU	R	R
Verify Receipts against Order	RU	RU	RU		R
Send Invoices to Accountant	RD	RD			
File Order Copy by Vendor	R	R			
Identify Late & Problem Orders	R	R	R	R	RU
Identify Items & Vendors			R	R	CRU
Call Vendor to Verify Avail/Price			RU		RU

FIGURE 5-18 Affinity Clusters in ABC Data Process Matrix

If not, is there some business reason why two processes are creating the same data? Or is there historically introduced redundancy? If the former, continue the analysis. If the latter, combine the processes and eventually redo the affinity analysis. Second, are the processes that cluster together in the same department? If so, the organization need not change. If not, then realign the organization boundaries to have all processes that create the same data reporting to the same manager. Expand the scope of the pro-cesses to include as much of the create-update-delete processing as possible. Needs for retrieval or access affect future plans rather than this decision process.

When the process analysis is complete, the remaining processes are all critical to the organization mission. The next task is to tentatively redefine jobs within the context of the remaining processes. The goals of job redesign are to enlarge and enrich the jobs, and to eliminate interprocess dependencies through job design. Interprocess dependency is eliminated or reduced by the caseworker approach to job design and by expanding data access to all who use it.

To define a job, begin with the processes in a function. Add processes to the job until either the skill mix or activity served changes. Then, define another job until either the skill mix or activity changes. Continue to define jobs until all processes are assigned. There may be jobs that span activities but they should be exceptional.

After jobs are completely defined, map them to functions by their affinity, that is, in terms of their data creation and usage. Do not pay attention to the number or types of jobs reporting to functions at this point. Again, concentrate on eliminating errors, paper, and dependencies. When all jobs are mapped to activities, the first phase of organization redesign is complete. The next phase takes place during the implementation planning.

The second analysis and redesign that results from process/data analysis is for subject area databases and applications to support them. This is a more subjective analysis than job redesign because there is no theory of application development and how to size applications. The current thought is that applications that support well-defined subject areas will provide the best organizational support. The reason for this is that subject areas, data entities, and attributes are all fairly static. With well designed data, the processing can change without affecting the database.

First, use entity clusters to define subject area databases. Check that each entity is also linked to at least one goal or objective. If an entity is not linked, either establish the correspondence, or mark it for elimination. Conversely, analyze the processes which use the entity. If this is the only data used by the process, but the process is tied to some goal, determine the presence of data to measure progress toward the goal and, if needed, add a new entity to the list; otherwise, if the related process also stands alone, mark both the entity and the process for elimination.

The subject area databases defined by affinity analysis should be mapped to current, automated applications. If the subject areas are completely automated and the applications are integrated, no changes are needed. Rarely is this the case. Usually several applications process pieces of subject area data and both manual and automated usage of data is required. The only integration is through the experience of users who know where to go for information they need.

Redefine applications to support each subject area of data. Define application changes for process changes that reduce problems. Define ad hoc query facilities for all jobs requiring retrieval access to data. Assume on-line processing for most application work. Identify and recommend technologies that streamline and speed information storage and delivery. Based on the problems and solutions identified, determine the potential impact of applications for meeting goals. Prioritize applications for development to achieve the greatest impacts first.

Network/Technology Design

Before either the network or the technology designs are done, the receptiveness of the sponsor and managers to the changes in jobs and applications should be verified. If they support the work to date, the network and technology analyses can continue. If they do not support the job redesign or are reluctant about

application suggestions, those aspects of the reengineering must be defined acceptably before this analysis.

There is no theory of network or technology design at the enterprise level. Rather, we have rules of thumb that must be evaluated in each business context. First, if the job redesign and process analysis substantially change the activities being performed in the organization, the enterprise network model should be recast in terms of the revisions. Next, if locations are significantly different, the technology model should be redrawn to reflect revised locations.

When the two network diagrams are acceptable, they are compared and analyzed to recommend new and changed technologies for supporting the new organization.

Using the technologies identified as needed to fully support jobs, develop an overview of the technology for the organization. Classify types of applications on mainframes, local area networks (LANs), and stand-alone personal computers. Classification should identify applications by size, 'corporateness' of data, data sharing requirements, specialized technology required, and number of users.

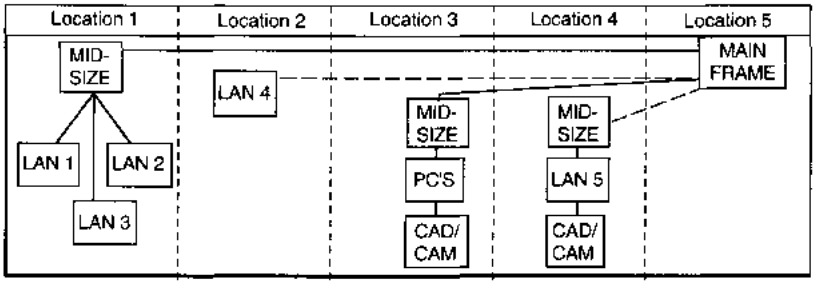
Across the organization, rationalize the use of technology resources, minimizing the overall cost to the organization. If new technologies are recommended, develop estimates of implementation costs and benefits, including average cost per expected user employee. If possible, identify incremental costs for expanding the user base once the technology is installed. Include training costs in the estimates. Identify and recommend possible uses for technologies to reduce incremental costs of use.

This activity is one in which the IS representatives have the most value added during reengineering. Being technology literate, IS representatives can work with their technology planners to determine possible technologies for consideration that have not been identified before. The IS people should take the lead in the rationalization of technologies. Deciding the type of applications that belong on various platforms for the organization requires the knowledge and guidance of the IS steering committee or the IS director (i.e., Chief Information Officer, MIS Manager, or some similar title). Explanations of the applications mapping to technology platforms should be in business terms but based on sound understanding of the technology involved.

An example of network/technology redesign for the plastics company example is provided. The plastics company architectures in Figures 5-15 and 5-16 are used to create the revised network in Figure 5-19. One obvious problem is that organizations that need to communicate for work are not electronically connected. This suggests a network change to interconnect all interdependent activities. This change means that the LANs that are only connected through a star configuration in *Location 1* might be connected via a backbone to the midsize computer. Backbones in each location with multiple LANs can be connected to provide intra-location communications, freeing the larger machines for inter-location connection and data processing. With this type of network design, everyone in the company can communicate with everyone else.

After this cursory analysis, we next look at the technologies used for subject databases and applications. First, the subject databases are added to the technology map. If pieces of databases are scattered, integrate them or determine distribution requirements. This type of recommendation should be coordinated with the applications recommendations which are probably similar. Recommendations about centralization, decentralization, federation, or distribution of both data and processes should be considered. Changes in all infrastructure software such as telecommunications monitors, database management software, terminal interfaces, and so forth should be considered for each activity at each location. Advantages and disadvantages of all technologies, current and proposed, should be developed and an estimated cost-benefit analysis developed.

In the plastics company example, software and applications are added to the network/technology analysis shown in Figure 5-20. Order information is only available at Location 1, even though all sales and product management organizations (Locations 1 and 2) require access. These data differences in what currently exists to what is required show the type of findings in network/technology analysis. To determine the best course of action, more information



Current Applications				
Customer Maintenance Order Processing Personnel Management Payroll (Locations 1, 2, 3) Accounting Financial Management (local only)	Financial Management (local only)	R&D R&D Manufacturing	Scheduling Manufacturing Control Payroll (local only)	Accounting Consolidation Accounting (local only) Financial Management (local only) Payroll (local only)

Required Applications				
Customer Maintenance Order Processing Personnel Management Payroll (Locations 1 & 2) Accounting Financial Management (local only)	Customer Maintenance Order Processing Accounting (local only) Financial Management (local only)	R&D R&D Manufacturing	Order Inquiry Subsystem of Order Processing Scheduling Manufacturing Control Payroll (local only)	No change

Legend:

———— = Permanent Link

- - - - - - = Dial-Up Link

FIGURE 5-19 Plastics Company Network and Technology Analysis

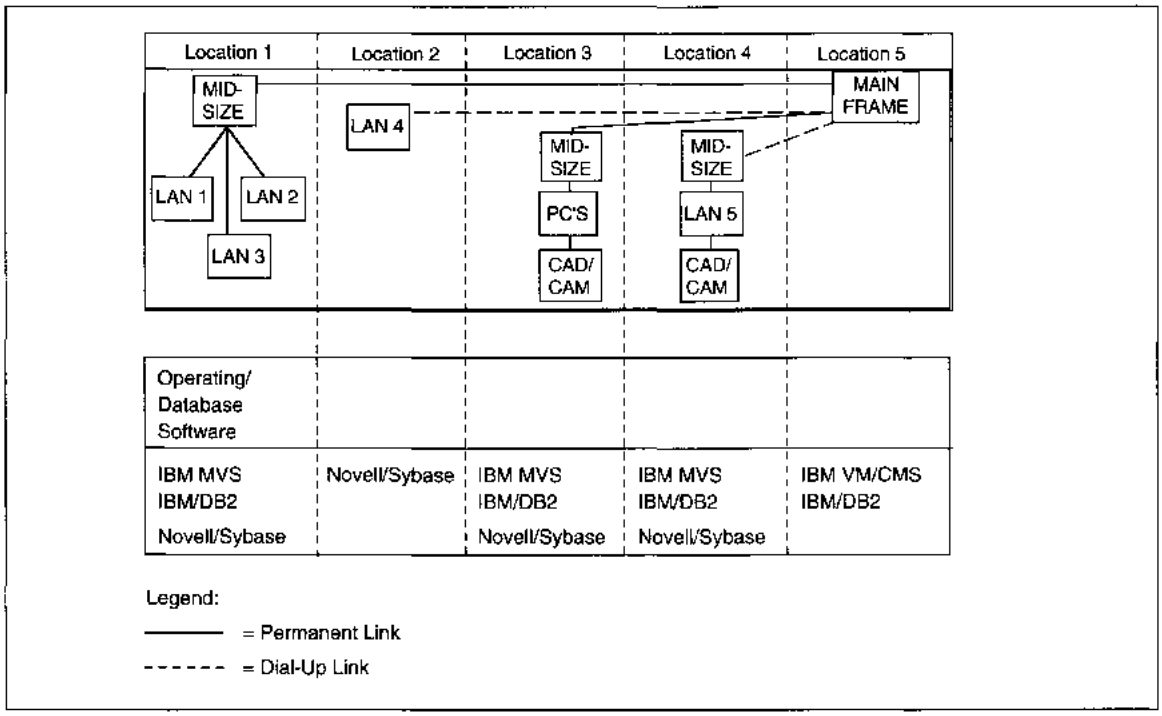


FIGURE 5-20 Plastics Company Technology and Software Details

might be requested of the locations. For instance, do they need up-to-the-minute information? Why or why not? The answer to this question determines the need to redevelop the applications as on-line rather than batch. If the locations need up-to-the-minute information, on-line applications are required. Let's say that the sales and product management information users need orders only as of the previous close of business and that customer service agents in Location 1 would like up-to-the-minute information because most changes are made the same day. This information about needs gives the reengineering team the details they need to make intelligent recommendations about application changes. In this case, either on-line order entry with retrieval, or the entire application as on-line might be acceptable alternatives.

Next, consider new technologies to manage paper and work flow. For instance, do using groups need facsimiles of the paper forms? In some industries, such as insurance, the answer would be yes. In plastics manufacturing, the answer is no. So, imaging or other micro-forms management hardware and software are not considered.

Specific operating environments should be considered next. If the networks are used to pass electronic mail and data files back and forth, the operating environments do not necessarily have to be the same. If, however, on-line query and file sharing across environments is desired, the network operating systems and database management software probably should be the same to simplify user access. This type of decision is aided by development of a cost-benefit analysis for data access using consistent software. What is cost of change? What is the risk and cost of not changing? How much added time is required, per request, to formulate and obtain information with no change, and with change? The answers to these questions are used to determine the redesigned operating environment.

In the plastics example, the current environment down-loads information nightly from Location 1 to

Locations 2 and 4. The managers at those locations would like access to interim data *if* the applications are moved to an on-line environment. In other words, they want the access if the data are more current. Customer service needs current information. We decide to move to the on-line environment and provide networkwide access to data and services on the net. If the network operating systems (NOS) and data bases are incompatible with this idea, they would need to be replaced and made compatible.

To summarize, the network and technology architectures are superimposed and compared to decide company changes. Then, technology requests and application and software recommendations are superimposed on the revised technology diagram. Evaluation of requests, suggested changes from IS, and recommendations from the organization design team takes place by analyzing each change. Change evaluation includes cost-benefit analysis, development of advantages and disadvantages of change, and issue analysis with information supplied by potential users.

Implementation Planning

Once the analysis and recommendations are complete and tentatively approved, a plan to prioritize and sequence the changes is developed. A reengineering study is of limited use if there is no road map for *how to attain* the recommendations based on where the organization is today. Implementation planning designs the map. The steps of this phase are:

1. Develop job descriptions.
2. Define the organization.
3. Plan information technology.
4. Plan training.
5. Plan implementation.

Define Job Descriptions

This is a first-cut at describing the new positions. The jobs still require human resources evaluation and refinement during the next stage: implementation. To develop jobs, we reanalyze the tentative job descriptions, attending to data needs for each job. We define jobs as including related job skills for similar, related data. For each job, list the processes, data, and skills required of an incumbent. When the subject area database changes, create a new job, but keep as a goal that each job should do some 'whole thing,' have decision power, access to all needed data, and be self-contained. Keep in mind that constraints on job identification are data self-sufficiency, process self-sufficiency, and minimal coupling to other jobs and processes.

For each job, identify the processes and entities. Identify the technologies that would achieve the job objectives with the utmost speed and accuracy. Use suggestions (and return for more specific information if necessary) from interviewees about technology that might be used. At this point, do not worry about capital expenditures for technology. Keep technology information for the technology/network analysis.

Question all current methods of work and all process dependencies. For instance, do you need paper copies of orders? By law, you need records of orders, not *paper* orders. Devise schemes that eliminate paper, eliminate creation of paper, and eliminate any handling of paper. Replace paper with technology whenever the information must be retained for legal or governmental compliance.

Concentrate on implementing change to eliminate *all* identified problems. Relate each process and entity to one or more problems identified; determine how to improve quality of process and eliminate the errors. Finally, concentrate on eliminating dependencies between functions and between processes. Interfunctional dependency is minimized by eliminating physical interactions or replacing them with technology based interactions. For instance, eliminate shipping papers by providing the shipping department with access to the order database.

For each job within each process, write job descriptions to align job goals with the corporate goals and objectives. The outcome of this exercise is to give every individual the means—management structure, data, and technology—of meeting those goals. Give every individual, at every level, specific measurable responsibilities. Recommend changes to the compensation plans to relate compensation to meeting/exceeding of objectives and goals.

For each newly clustered, enlarged job, analyze its relationships with other jobs to minimize inter-job linkages. Reanalyze each job to ensure data and process self-sufficiency. Finally, define defect-free work procedures. If errors must be dealt with, describe where they might occur and their proper handling.

Define the Organization

A first-cut organization structure will have three layers: CEO, functional managers, and everyone else.

The implication is that self-directed work teams with either a limited hierarchy or a matrix management organization will result. Other organizational forms can result but are not specifically defined in any of the reengineering methodologies. The steps to developing a new organization design are: map jobs to functions, analyze relationships between jobs placing jobs in clusters or work groups, based on data self-sufficiency, process self-sufficiency and minimal coupling of clusters, and determine the location of work (in large organizations some jobs are centralized, some decentralized, and some centralized with replication in the remote locations). If the first-cut does not result in a completely irrational organization design, it might be accepted as it is for trial. If there are too many different clusters (use 5–7 as the rule) or too many different jobs in a cluster (use 5–15 as the rule), additional reevaluation might be required.

Grouping of jobs is based on their interdependence. There are three types of interdependence in organizations: pooled, sequential, and reciprocal. **Pooled interdependence** is a relatively independent, low level of interaction between departments or jobs. **Sequential interdependence** defines a serial relationship between departments or jobs. **Reciprocal interdependence** defines highly interrelated activities that are worked on jointly by multiple units requiring feedback and constant adjustment. For instance, a bank loan department might be viewed as relatively independent (i.e., pooled) from other parts of a bank in that they need customer information received from the customer for their decision with no other units involved. Purchasing, receiving, and payables are sequentially interdependent in that they all use purchase order data. Yet all these job types have different job skills; that is, they each make different decisions and perform different actions based on their access to the purchase order information. A reciprocally interdependent department is a hospital intensive care ward in which many specialists with different skills and knowledge all work toward the same goal of patient recovery.

To group jobs, three methods of organization design deal with the three types of interdependence. If the jobs relate to each other sequentially, cluster jobs with similar skills together. Affinity groupings of processes and entities are used to decide skill requirements. Clusters may be sequentially dependent with jobs within each cluster providing different skills. Plan to provide shared database access to link clusters; this minimizes paper movement and ensures data access.

For example, look at the bank loan department again. Bank loan department processes are sequentially related after the loan is made. Once the loan commences, records are established and payments are received, posted, and analyzed. In an assembly line approach, these processes are different jobs. In a caseworker approach, all of these processes are within one job. Caseworkers could conceivably monitor loans for any customer, but usually have a case 'load' that is defined by alphabetic groupings of last initial of loan-maker names or some similar scheme.

If the processes have pooled interdependence, then job clusters contain one job type. For pooled interdependence, use subject area data as the deciding factor on when to create a new job. Each job, cluster, or group should have its own data self-sufficiency.

If the jobs are reciprocally interdependent and pass work back and forth, or need discussion on details regularly during the performance of work, design work groups in the same way you designed jobs. That is, design work groups to include all skills needed to perform one activity or function. Find all of the jobs that reciprocally share information; then, define the set of different jobs that would comprise a work group. Try to keep groups small with under 12 different jobs represented. For instance, engineers, raw materials purchasing, manufacturing, and

quality control may all need access to the same design drawings, specifications, and components lists. They may be able to identify alternatives, make decisions, and improve quality simply by sharing responsibility for finished products. These job types would be clustered in work groups (i.e., quality circles).

Plan Information Technology

The next step is to redefine the IS environment. The rationale for deciding priorities is to correct the major problems first, and/or meet the goals/objectives with the largest impact on net income. The steps to develop an IS redevelopment plan are:

1. Compile all subject area database and application changes, redevelopment, enhancement requirements.
2. Compile all technology and network infrastructure requirements.
3. Map technology and network needs to database and application needs.
4. Define software reengineering projects.
5. Define new application development projects.
6. Determine priorities for all projects.
7. Develop a plan for two years of development and reengineering work. Develop a tentative 3–5 year plan for the remaining projects.

To develop the technology plan, create three matrices: technology/process, process/entity, and an entity/technology matrix. The technologies are all those identified by interviewees and team members as potentially useful in the organization. Complete each matrix. In each cell of the process/technology matrix, enter whether the technology speeds delivery, improves accuracy, improves service, or lowers cost. Enter all improvements that apply. This matrix is used to determine priorities for change.

In the entity/technology matrix identify which data entities are already fully or partially automated and the type of automation. Types of automation include file, application database, or subject area database.

Using the original process/entity matrix, identify the extent and type of automation for each process/data cell. Types of automation for processes include full or partial, and batch, on-line, or real-time. These matrices may not be 100% complete, but are used to guide the implementation planning process by providing a summary of planned changes.

Plan Training

Develop a training plan to upgrade skill levels to meet new performance requirements, recommending how current jobs can be mapped onto the new jobs. This should be a skeleton plan defining sequencing of training and approaches—outside company, inside company, phased by department, phased over time, and so on. Actual training details cannot be complete until human resources' redefinition and formalization of job descriptions and levels, and estimates of number of people to be trained for each job are known. The plan should be sufficiently detailed to allow a pilot test of the training and new work approach before its complete deployment.

Plan Implementation

Develop an implementation plan that reflects some phased approach to changing the organization. The number of people in any one job type might be difficult to determine if the jobs are very different from the present. Moving from the assembly line to the caseworker or group work approaches changes the entire equation; more, rather than less, people might actually be needed. Human resources might be able to assist in this type of estimating. If estimates of numbers of people in caseworker jobs are too vague, a pilot study can be conducted to facilitate estimating of total personnel needs.

When the mapping is complete, summarize the recommended changes and determine how they can be implemented. The possible approaches are pilot organization, phased implementation (by function, location, business priority, or application), or total cut-over. Develop timing of changes. If the changes are expected to take more than six months, determine how the organization, processes, data, or technology can be streamlined, changed, added to, or

eliminated *now* to provide immediate improvement and correction of some problem(s).

Enterprise Analysis Without Organization Design

Even without the extensive organization and technology redesign of reengineering, an enterprise analysis helps managers establish applications priorities and develop a plan for introducing new applications and technologies into their organizations.

The same analyses for entities and processes are performed. Current automation of the affinity clusters are summarized on the diagram. Recommended changes are mapped to organization goals and strategies to decide priorities for change. The changes from enterprise analysis are incremental and relate to applications and subject area databases. Sweeping technology and network reassessment are missing from this activity. Likewise, organization problems and finding obsolete functions are not goals of this analysis.

When organization problems are identified, they can be referred to the sponsor for consideration. One example of organization problems is identified from the entity/process matrix after affinity analysis is performed. Each process should have a prefix identifying its original function and activity relationships. If the function/activity prefix for each creating process for each entity is not the same, an anomaly is found in that multiple managers have responsibility for creating the same data. The idea is that processes which *do* share responsibility for creating some entity should report to the same manager. The same manager can minimize conflicts and maximize coordination and control over data creation.

A second type of organization problem is found in the process hierarchy diagram. Because the diagram is built to describe its information without regard to current organization, some overlap or duplication of activities may be found. When this occurs, an effective technique for showing duplication, for example, is to draw shadow boxes, behind the process (or activity or function) duplicated. Then, on each box, identify the organization having the responsibility, one box for each organization. This effectively communicates organizational overlap without a need for additional comment, and is less inflammatory than verbal or text descriptions because it is presenting organizational facts.

Automated Support Tools for Organizational Reengineering and Enterprise Analysis

The tools needed to support organization reengineering are similar to those for project planning, but include process hierarchy diagrams, entity-relationship diagrams, network architectures, and technology architectures. Many tools support one or more of these requirements. Few tools on the market currently support all of these requirements. The automated support tools are summarized in Table 5-2.

Summary

Reengineering of an organization reevaluates data, processes, technologies, and communications needs to ensure that an enterprise meets its goals as stated in its mission statement. The activities of reengineering include the data collection, analysis, and development of recommendations to meet organizational goals through radical redesign of work.

Reengineering is intended to alter the shape and operations of an organization. Frequently, organizations and managers do not want sweeping change. When incremental change is desired, enterprise level

TABLE 5-2 Automated Support for Organizational Reengineering and Enterprise Analysis

Product	Company	Technique
Analyst/Designer Toolkit	Yourdon, Inc. New York, NY	Entity-relationship diagram (ERD)
Anatool, Blue/60 MacDesigner	Advanced Logical SW Beverly Hills, CA	ERD
Bachman	Bachman Info Systems Cambridge, MA	Bachman ERD
CA-*products*	Computer Associates International, Inc.	Data modeling Strategic planning
CorVision	Cortex Corp. Waltham, MA	ERD
Deft	Deft Ontario, Canada	ERD
ER-Designer	Chen & Assoc. Baton Rouge, LA	ERD
Excelerator	Index Tech. Cambridge, MA	ERD Structure chart
Foundation	Arthur Anderson & Co. Chicago, IL	ERD Project management Project planning
IEF	Texas Instruments Dallas, TX	ERD Enterprise analysis and planning Process hierarchy
IEW, ADW (PS/2 Version)	Knowledgeware Atlanta, GA	ERD Enterprise analysis and Planning Functional decomposition

analysis uses a subset of the analyses of reengineering to develop applications and subject area database development recommendations.

REFERENCES

Davenport, Thomas H., *Process Innovation: Reengineering Work through Information Technology*. Boston, MA: Harvard Business School Press, 1993.

Dunckel, Jacqueline, *Good Ethics, Good Business: Your Plan for Success*. North Vancouver, British Columbia: Self-Counsel Press, 1989.

French, W. L., and C. H. Bell, Jr., *Organization Development*. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1984.

Galbraith, Jay R., and Daniel A. Nathanson, *Strategy Implementation: The Role of Structure and Process*. St. Paul, MN: West Publishing Co., 1978.

Galbraith, J. R., *Organization Design*. Reading, MA: Addison-Wesley Publishing Co., 1977.

TABLE 5-2 Automated Support for Organizational Reengineering and Enterprise Analysis (*Continued*)

Product	Company	Technique
MacAnalyst, MacDesigner	Excel Software Marshalltown, IA	Decision table Entity class hierarchy ERD
Maestro	SoftLab San Francisco, CA	ERD
Multi-Cam	AGS Mgmt Systems King of Prussia, PA	ERD Enterprise analysis and planning Project management
PacBase	CGI Systems, Inc. Pearl River, NY	Enterprise analysis and planning Process decomposition
ProKit Workbench	McDonnell Douglas St. Louis, MO	ERD
Silverrun	Computer Systems Advisers, Inc. Woodcliff Lake, NJ	ERD
SW Thru Pictures	Interactive Dev. Env. San Francisco, CA	ERD
System Architect	Popkin Software and Systems, Inc. NY, NY	ERD
System Engineer	LBMS Houston, TX	ERD
Teamwork	Cadre Technologies Inc Providence, RI	ERD
Telon, and other products	Pansophic Systems, Inc. Lisle, IL	ERD
The Developer	ASYST Technology, Inc. Napierville, IL	ERD Structure chart Organization chart

Greiner, L. E., and R. O. Metzger, *Consulting to Management*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1983.

Hage, J., and M. Aiken, *Social Change in Complex Organizations*. New York: Random House, 1970.

Hackman, J. R., ed., *Groups That Work (and Those That Don't): Creating Conditions for Effective Teamwork*. San Francisco, CA: Jossey-Bass, 1990.

Hackman, J. R., and G. R. Oldham, *Work Redesign*. Reading, MA: Addison-Wesley, 1980.

Hammer, M., "Reengineering work: Don't automate, obliterate," Harvard Business Review, July–August, 1990, pp. 104–112.

Hammer, M., "From cow paths to data paths," *Computerworld*, December 25, 1989–January 1, 1990, pp. 16–17.

IBM Corporation, *Business Systems Planning Information Systems Planning Guide*, IBM Document # GE 20-0527-1, Armonck, NY, 1978, pp. 1–92.

King, William R., "Strategy set transformation," *MIS Quarterly*, March, 1978.

King, W. R., and D. I. Cleland, eds., *Strategic Planning and Management Handbook*. NY: Van Nostrand Reinhold, 1988.

Huse, E. F., *Organization Development*. New York: West Publishing Co., 1980.

Kouzes, James M., and Barry Z. Posner, *The Leadership Challenge: How to Get Extraordinary Things Done in Organizations*. San Francisco, CA: Jossey-Bass, 1990.

Lindenfeld, F., and J. Rothschild-Whitt, eds., *Workplace Democracy and Social Change*. NY: Porter, Sargent, 1982.

Rockart, John. "Critical success factors," *Harvard Business Review*, March–April, 1979, pp. 81–91.

Singh, Arvind, Comments from A Business Reengineering Workshop given in NY to TIAA, Performance Development Corporation, Princeton, NJ, January, 1992.

Sowa, J. F., and J. A. Zachman, "Extending and formalizing the framework for information systems architecture," *IBM Systems Journal*, Vol. 31, #3, 1992, pp. 590–616.

Thompson, J. D., *Organizations in Action*. New York: McGraw-Hill, 1967.

Zachman, J. A., "A framework for information systems architecture," *IBM Systems Journal*, Vol. 26, #3, 1987, pp. 276–292.

KEY TERMS

affinity
affinity analysis
architecture
business activity
business function
business process
caseworker
champion
critical success factor (CSF)
CRUD matrix
data
data architecture
data domain
data self-sufficiency
domain
enterprise architecture
information systems architecture (ISA) framework
information technologies
level of effort
logical description
mission statement
network architecture
network domain
network scope
organizational reengineering
pooled interdependence
process architecture
process domain
project sponsor
reciprocal interdependence
reengineering scope
sequential interdependence
stakeholder
technology architecture
technology domain
technology scope
user commitment

EXERCISES

1. Look at the questions suggested for data gathering on page 125. Think of other possible questions and why they might be good additions to those suggested. Discuss your suggestions with class members.
2. Describe how the information provided for the four architectures can be used in multiple ways as the basis for IS and organization redesign.
3. Discuss the differences in outcomes of an organizational reengineering project if one or more of the assumptions in the list on pages 116–117 are not met.
4. Try to develop process and data architectures for the Abacus Printing Co. case in the Appendix. Try to do an affinity analysis of the information. Develop a list of questions you need answered to do a complete analysis.

STUDY QUESTIONS

1. Define the following terms:

   data architecture
   enterprise analysis
   information technologies
   network architecture
   organizational reengineering
   process architecture
   technology architecture

2. What is the motivation for organizational reengineering?
3. What are the steps to organizational reengineering?
4. Why are caseworker or quality circle work groups preferred to the assembly line approach to work?
5. What is the 80–20 rule and how does it apply to reengineering?
6. What is an architecture and why is it important to reengineering?
7. What types of architectures are used in reengineering? What is the purpose of each architecture?
8. What is an entity and how is it used in the data architecture?
9. What is a platform and how is it used in the technology architecture?

10. List three prerequisites of reengineering. Why are they necessary conditions for a successful project?
11. What are four assumptions of reengineering?
12. Why are different scheduling scenarios necessary for the organization of reengineering projects?
13. What is a level-of-effort approach to work? Why is it used with reengineering?
14. Why is there overlap between reengineering tasks? Why is overlap necessary?
15. What is the role of the project sponsor?
16. List the types and roles of people who should be assigned to a reengineering project.
17. Why is data self-sufficiency the major criterion for scoping a reengineering project?
18. Describe a good mission statement. What makes the difference between a good mission statement and a bad one?
19. How are critical success factors used in reengineering?
20. List five information sources and the type of data that the team gets from each one.
21. Discuss the conceptual levels of Zachman's IS architecture. Which two relate to reengineering? Why are the others not used here?
22. What is the purpose of mapping the two levels of architecture into different domains? Why the domains chosen?
23. Who is a stakeholder? Why is a stakeholder important?
24. Describe a CRUD matrix and its use.
25. Why is affinity analysis important? What are the reengineering results that are based on affinity analysis?
26. List three rules of thumb for developing the network and technology recommendations.
27. Why is implementation planning important to a reengineering effort? When changes are dramatic, what is a good approach to implementing change in the organization?
28. How does enterprise analysis differ from organizational reengineering? Are these differences significant? Why not do enterprise analysis only?
29. Which automated support tools provide all desired functionality for reengineering support?
30. What are the functions desired of an automated support tool for reengineering?
31. What are the key criteria for proper scoping of a reengineering project? Explain.

★ EXTRA-CREDIT QUESTIONS

1. You have been named to lead an organization reengineering effort for a small, one-location company. The company has functions for accounting, purchasing, inventory management, shipping, and sales. The business of the company is retail sales of furniture. The current computer system supports the billing, shipping, and invoicing process. No one but employees in the accounting department use or access the computer at present. Develop a plan and sample questions you might ask the employees and the owner for an organization reengineering project.
2. What are factors that can cause a reengineering project to complete faster or slower? Explain.
3. Imagine that you work in a company that has all types of computer hardware and networks: mainframes, mid-size, PCs, wide-area mainframe networks, and local area networks. What are the issues in defining what data and applications should be on each type of hardware? Develop and discuss possible guidelines for data and application location selection.

CHAPTER 6

APPLICATION FEASIBILITY ANALYSIS AND PLANNING

INTRODUCTION

Feasibility is the first stage of application development. The purpose of the feasibility study is to ensure that the organization can accommodate the technology, organization changes, and cost of the new application. During feasibility analysis the major tasks are: define the scope and boundaries of the problem, generate technical alternatives, assess costs, benefits and risks, and recommend an application development strategy. The procedures described in this chapter are used for large, full life-cycle projects; selective and abbreviated forms of the analysis are used for iterative development and for small projects.

DEFINITION OF FEASIBILITY TERMS

The feasibility analysis tasks and the terminology of each are briefly described. The stages of work during feasibility are: gather information, develop alternatives, evaluate alternatives, and plan and document the recommended approach to development.

During the information gathering stage, the goal is to develop a request from a vague, general statement into a specific request with boundaries and scope completely defined. Key business and application leverage points are defined during the scoping activity. A **business leverage point** is some activity from which a competitive advantage can be gained. An **application leverage point** is some automated function that might provide a competitive advantage to the using business unit(s). Application leverage points frequently relate to improvements of *better*, *faster*, and *more* to work. Some business and application leverage points are:

Increase market share
Increase linkage to vendors or customers
Provide desired information that is not currently available.

Business and application leverage points are used as the starting point for developing the benefits that would result from a change in the current method of work. Benefits can be tangible or intangible. Both benefit types are important for management to decide whether or not to do the recommended changes. **Tangible benefits** are measurable improvements to a specific work product or process. For instance, reducing staff by 10 people

and the resulting cost savings are tangible benefits. **Intangible benefits** are not directly measurable. For instance, improved customer service through integrated database access has tangible and intangible benefits:

Tangible Benefits

Decrease operating cost by 10% in first year
Increase market share by 5% per year for three years

Intangible Benefits

Improve company image
Increase customer satisfaction
Improve employee job satisfaction
Provide faster and more accurate information to customer services representatives

Another tangible benefit might be faster response time for inquiry requests from five minutes to 15 seconds. An intangible benefit from the same action might be improved customer satisfaction. More satisfied customers are less likely to go elsewhere for their products, but proving that customer satisfaction is improved is difficult to quantify, and is intangible.

Also in information gathering, the business environment, competitive environment, and current method of performing the work that would be revised are described in sufficient detail to allow determination of appropriate changes. The functions and procedures that are needed in the new application are identified, as are problems with current procedures and new functions that are not part of current procedures.

After the current problem domain is understood, alternative approaches to the problems are developed. **Alternative approaches** to an application are different configurations of work, hardware, firmware, or software. Alternatives can begin with nonautomation alternatives, such as change in work flow, and progress to different platforms, software, and designs. Usually between two and five alternatives are considered. Alternative definitions include the technology, benefits, and risks of each approach. A **benefit**, as discussed above, is some improvement in the work product or process that results from a specific alternative. **Risks** are events that would prevent the completion of the alternative in the manner or time desired.

Risk assessment determines possible sources of events that might jeopardize completion of the application. In general, the goal is to develop the project on time, within budget, and without errors. Risk assessment and contingency planning help you meet this goal. **Contingency planning** is the identification of tasks designed to prevent risky events and tasks to deal with the events if they should occur. The goal is to minimize the possibility of the event occurring, but to also have a plan *just in case* the worst happens. Having a contingency plan prevents having to force decisions under pressure.

When the alternatives have been defined, they are evaluated. The number of requirements met by the approach is assessed, and the benefits and risks of each are weighed to identify the alternative with the least risk and most benefit. If an alternative exists that meets all required and optional requirements, meets all benefits, and has the least risk, it would be the recommended option. Most often, there is a mix of requirements met and risk incurred, that prevent selection of an alternative based on technical merits alone. Rather, several competing alternatives might be further evaluated to differentiate between them. To decide between the alternatives, development plans, costs, and financial analysis are developed.

A project plan is a schedule of tasks and estimated completion times for application development. A project plan includes tasks to be completed, tentative task assignments, staffing plans, and computer resources needed for the project. From the staff and resource estimates, costs of development are determined. If there are multiple alternatives, the costs of each are computed. The costs are used in the financial analysis which occurs next.

Several different types of financial analysis might be performed; the two most common ones are cost/benefit, and make/buy. **Cost/benefit analysis** is the computation of net present value for each alternative. **Net present value** (NPV) equalizes the cost estimates by accounting for the time value of money for multiperiod investments. A **make/buy analysis** chooses between alternatives for providing an item, such as a software application. The *make* analysis

estimates the cost of building a customized application, while the *buy* analysis estimates the cost of purchasing a package.

Other financial analyses, such as internal rate of return and payback period, might also be computed. **Internal rate of return analysis** determines the interest rate which equates cash investment outflow with positive cash flow. **Payback period analysis** determines the number of years required to recover the cash outlays based on the projected monetary benefits.

After all the analyses are performed, a final recommended alternative is defined. Technical and monetary considerations are balanced and a recommendation is based on some mix of them. For instance, a recommendation might be based on the fastest payback coupled with most requirements met. Alternatively, the decision might be based on the lowest NPV and the extent to which leverage can be maximized. When the alternatives are virtually equal in comparison, multiple approaches to the application are presented and the user, IS managers, and project team decide together what approach is best. This is often the case.

Finally, a feasibility document is created to summarize the feasibility analysis and the recommendation. The document is a summary of all of the preceding steps and analyses taken during the feasibility phase. Next, we discuss each feasibility activity in detail.

FEASIBILITY ACTIVITIES

Feasibility analysis is an activity that ranges from several days to several weeks in duration. In general, a feasibility should be completed in fewer than 12 weeks; after that point, one of two problems exists. Either the problem domain is too large and should be broken into smaller problem areas, or the feasibility team is going into too much detail and should summarize at a higher level. The information at the end of feasibility should be accurate enough to allow managers to decide on the worth of pursuing a project, but high level enough that an analysis phase to clarify details of requirements is needed. The information is incomplete with about 95% confidence in the accuracy of the information. Similarly, a budget and project plan produced at this high level of abstraction should have about an 80% level of confidence attached to it. This means that the budget and time schedule are ±20% inaccurate, and implies budget adjustment later in the project. In this section, we detail the actions of feasibility analysis and project planning outlined in the previous section. For each topic, guidelines for completing the work are presented and followed by an example of the activity for ABC Video.

Gather Information

Guidelines for Gathering Information

The four major tasks during information gathering are:

1. Define the business and work environments
2. Describe current system of work
3. Identify key benefits and leverage points
4. Identify broad system requirements

The activities are done in parallel rather than sequentially. As information is collected, leverage points and requirements emerge from discussions on which old procedures to keep and what new technology, procedures, data, or interfaces are needed.

If an enterprise level plan exists, the data gathering begins with the architectures to obtain an overall view of the current data, processes, and technology of the target business area(s) (see Figure 6-1). The process decomposition is used to identify and match the affected jobs and tasks with those suggested by the requesting application sponsor. The data architecture is used to identify what data are involved and the extent to which the data are already automated. The technology architecture is checked to identify hardware, software, and applications supporting the work functions today, and to identify potential platforms as operational sites for the new application. For each job affected, the technology architecture matches jobs (from the process architecture) with applications capabilities.

The architectures, if present, are the basis for obtaining information from the user departments

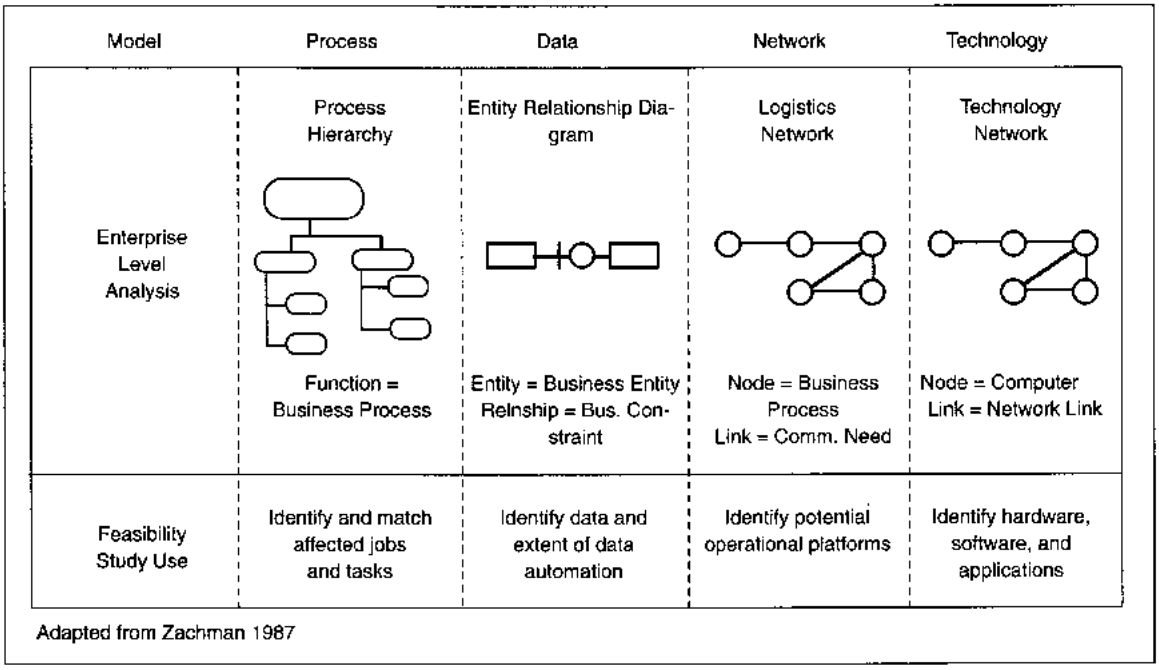


FIGURE 6-1 Enterprise Architectures in Feasibility Study

involved. Recall that the methods of data gathering (from Chapter 4) might include interviewing, document review, observation, talking to other companies, temporary work assignment, and questionnaire surveys. During feasibility, interviews, document review, and other companies are the primary information sources. Although the other methods could be used, they take more time and elicit more detail than required for feasibility analysis.

Assume you are doing the information gathering using interviews. You might work in two-person teams for interviews so that the project has a built-in backup for every person, should someone get sick, called on jury duty, or be reassigned. One person asks the questions while the other person acts as scribe taking notes. This method of interviewing results in fewer misconceptions and errors from forgetting than interviews by one person. At the end of every session, follow-up steps should be identified for both you and the interviewee. For instance, you might document the interview and ask the interviewee to review and correct your documentation. You commit to having the material back by a specific date and request the review within a set time. In this manner, you conclude the meeting with a commitment from the interviewee to do the review by a certain date.

During the writing of interview materials, graphical techniques for both data and processes can be used to synthesize the findings. The most common graphical techniques are entity-relationship diagrams (ERDs) for data, and process decomposition and process data flow diagrams for processes (PDFDs). Development of these diagrams is detailed in Chapter 9. An older variant of PDFDs called data flow diagrams (DFDs) are also used; they are detailed in Chapter 7. In general, ERDs capture information about the data entities that are within the scope of the study problem domain. An entity is any person, place, thing, or event about which the organization needs to keep data. The relationships between entities define some business-related association that is within the problem scope. The process decomposition diagram depicts the organization tasks that are being studied. The problem area is compared to the process hierarchy and ERD to

ensure correct scoping. PDFDs summarize the processes of the problem and relate them to each other, the outside world, and to data entities.

In addition to diagrams which summarize the procedures and data of the target problem domain, you also create text documents that describe the current process, the aspects of the current process to be retained, and the changes and motivation for changes. In general, text should be minimized because it is easily misinterpreted. Diagrams and graphics are preferred to text. Lists of items are preferred to paragraph form text. Requirements for the new application should be as specific as possible. For instance, a requirement might be stated 'reduce turnaround time from receipt of an order through invoicing from 14 days to 2 days.' During the systems analysis phase, the actual details of functions to implement this requirement are developed.

As we said above, key business and application leverage points are defined during the data collection activity. Leverage points are context specific. What might be a leverage point in one company and industry might be standard procedure in another company and industry.

An example of leverage points is provided by examining imaging technology. **Imaging** technology automates facsimiles of business forms. Image files are databases of forms with indexes for retrieval and linkage to data databases. Applications can be developed to integrate data and image information for users at terminals. The technology provides both business and application leverage by improving work flow and allowing the management of paper flow through an organization.

The leverage provided by imaging is highest in organizations that are information and paper intensive, for instance, insurance and financial services. These paper intensive industries are required, by law, to provide original document search capabilities. Before imaging technology, these companies either used microforms or paper, both of which have only rudimentary indexing capabilities. Microforms require their own viewing equipment that is neither intelligent, nor capable of integration to an application. Paper, if kept, is so voluminous that whole buildings are dedicated to document storage. Trying to retrieve specific documents and files requires armies of clerks and dedication to accurate refiling. Simply applying imaging technology by itself buys marginal improvement to paper management. The big payoff is in integrating imaging with software to manage work.

Work flow management software is integrated with imaging technology to schedule work for clerks, monitor document locations, and monitor work progress through any number of departments (see Figure 6-2). All of these actions can be done without fear of losing the document because it is an electronic image. Printing of the image is possible if a paper copy is needed by a clerk for some reason.

Imaging and work flow management together can flatten hierarchies, reduce the number of clerks involved in image production, and eliminate the need for clerks to manage files. Staff reduction is a business leverage point and a benefit of the activity. For individual jobs, frustration is reduced because information can not be 'removed for use' from an image file. Clerks are more productive and their jobs can be upgraded because the emphasis now can be placed on understanding and interpreting the information rather than on simply collecting all the information correctly. Thus, an application leverage point is present in enhancing jobs of the people in the work flow.

Leverage points identify benefits of the proposed application. Other benefits might be present and should be identified; they may not have a direct strategic impact. For instance, in keeping with the idea that most proposed applications are to improve work, benefits about more, faster access, integrated, or improved quality data might be defined. Similarly, automation of more tasks, faster report generation, integration of processing, or improved timing of response might all be benefits. Conversely, the new application might be expected to reduce staff, linkages between departments, work errors, and so on. These benefits are all tangible and measurable and should be identified.

Intangible benefits are equally important, but are harder to quantify. Intangible benefits are indirect, unmeasurable benefits with a high degree of uncertainty. For instance, one benefit of personal work stations with access to software has been a rethinking,

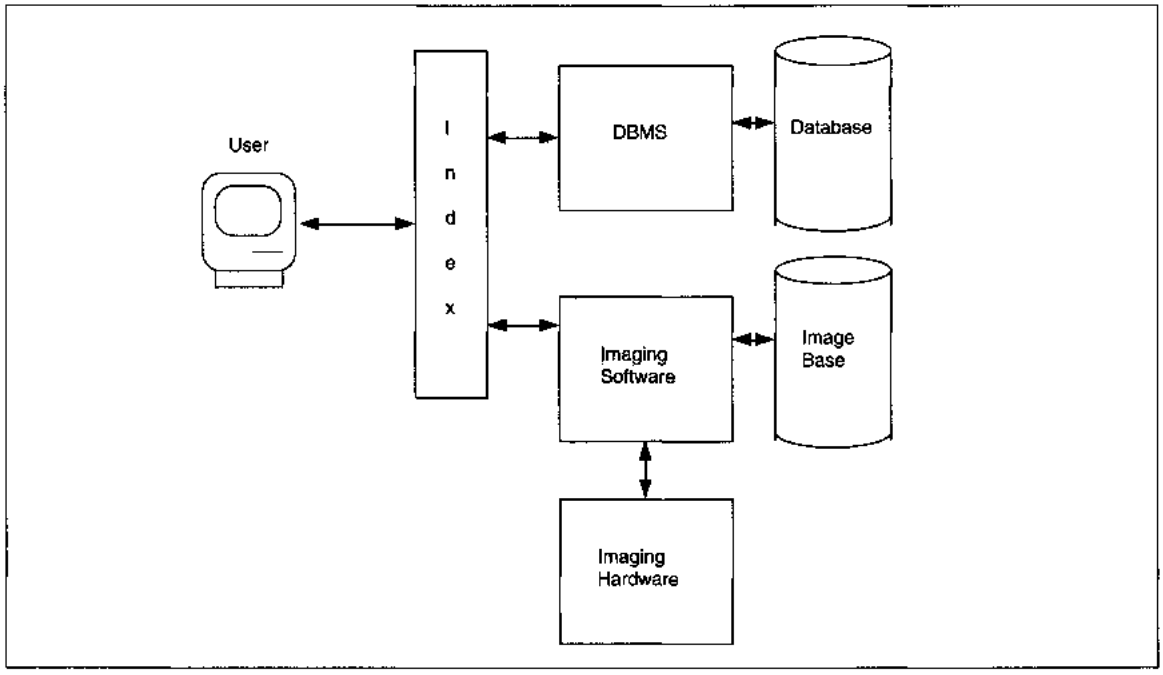


FIGURE 6-2 Logical View of Work Flow Management Software

by many people, of how they do their work. They now type their own documents directly and use secretarial support for changes and formatting. They do their own analyses and perform many different types of analysis that they could not do, and therefore never thought of doing, before they had desktop computer access. This type of change is an indirect benefit that increases the *effectiveness* of a person's work, while the tangible benefits deal mostly with *efficiency* improvements. Both types of benefits are important in application decisions.

The SE works with the users to define the tangible and intangible benefits relating to a project. Benefits identified are listed in the documentation of the proposed application, and a value is attached to each one. Tangible benefits are quantifiable by determining the change expected to result from the new application. Intangible benefits usually are listed with a possible range of benefit. In presenting this information to decision makers, you must be able to justify *why* intangible benefits exist. Managers will ask and expect the reasoning behind any expected financial gains, whether tangible or intangible.

Now let us turn to ABC Video to discuss how to perform the data collection activities.

ABC Video Information Gathering

Of the methods of data gathering available, several can be eliminated immediately. First, questionnaires for a total of six employees would be impractical. All employees are available for discussions during nonpeak times. Also, studying documentation is not possible because the manual methods are not documented. Observation and temporary work assignment can give some information about the current problems to be solved through automation, but are of limited value in actually designing the new application.

Talking to competitors is not feasible because they do not want to help the competition; however, to define benefits that might accrue for the ABC application, knowledge of competitor clerical assignments and computer systems is valuable. Observation of competitors is a good way to get some insight to the benefits Vic might get from

automation. The remaining data collection method, interviews, should be used extensively for Vic and the clerks to determine the work flow, problems, and possibilities for the ABC application. To supplement the interviews, we should observe competitors by using their services for a period of time to get information about their work assignments and applications.

For ABC, we define the current environment, proposed environment, leverage points, and benefits. Through Vic's interviews we find that ABC operates in a highly competitive environment. Large chain video rental businesses are crowding small one-shop businesses, like ABC, out of the market. ABC must remain competitive to stay in business and to grow as Vic expects. Vic sees the future to be in services offered to customers. In terms of video rental processing, service translates into minimal bureaucracy with as many variations on service to customers as possible.

Currently, ABC uses a manual method of video rentals. The customer chooses a video and presents the video cover (or title) to a clerk. The clerk locates the video, locates a rental card for the customer, and writes the current rental on the card. Charges for late fees are computed from the card if any are owed, and the customer pays for the current and any late rentals. The customer signs the rental card which is filed by the clerk. During the peak business period, from 6 P.M. to 10 P.M., the rental cards are placed in a pile for later refiling. Frequently, cards are misplaced and the customer is then not charged late fees. If a tape is never returned and the accompanying card is lost, Vic has no way to trace who has what tape(s). This method is error prone and subject to whims of clerks who have been seen changing return dates for friends who return tapes late. Also, the time involved in locating a given customer's rental card ranges from 30 seconds to several minutes during nonrush time, and can be as high as 10 minutes during the peak rush time because clerks are waiting to access the card file.

Vic's requirement for the new application is to provide a fast, simple method of providing rental processing and accounting without introducing any new bureaucracy into the process. The system must be on-line, accommodate at least five clerks working simultaneously, provide for growth in video inventory, and expansion of the business to other related sale/rental items. At a summary level, the data entities in ABC rental processing are customers, video inventory, and rentals. Figure 6-3 is an ERD showing the relationships between these entities. Also at a summary level, the major processes of rental processing are customer maintenance, video maintenance, and rent/return processing. These processes are summarized in Figures 6-4 and 6-5.

Figure 6-4 is a hierarchic process decomposition diagram for the business, showing many more functions than just the rental processing. The rental processing area has bold lines to highlight it from the rest of the diagram. This diagram is developed at the enterprise level to ensure that the correct departments and processes are accounted for in an application development effort.

Figure 6-5 is a high level process data flow diagram for the rental activities only. The diagram shows the inputs, processes, and outputs of the rental activity. Inputs are rent/return requests, payments, process requests, new customer information, and new video information. Processes are maintenance, reporting, and rental/return. At the feasibility level, this is an acceptable level of detail for data and procedure knowledge and documentation.

To determine leverage points for ABC's application, we examine what the application does for ABC in the context of its industry and competitive environment. To do this we ask and answer several questions. First, can this application give ABC a competitive position in the industry? The answer to this question must be no. ABC is a one-shop organization that might grow to several branches but is not expected to grow to national prominence. Therefore, the application might give ABC a presence in the local market, but the application's strategic impact on the industry is zero.

Second, does the application give ABC competitive advantage in the local industry? All other things being equal, the application could give some local advantage over other video stores in Dunwoody, Georgia, the town where the company is located. The impact on the local industry, in terms of suburban Atlanta, is close to zero. The other 'things' that must be equal or better for ABC to obtain a local

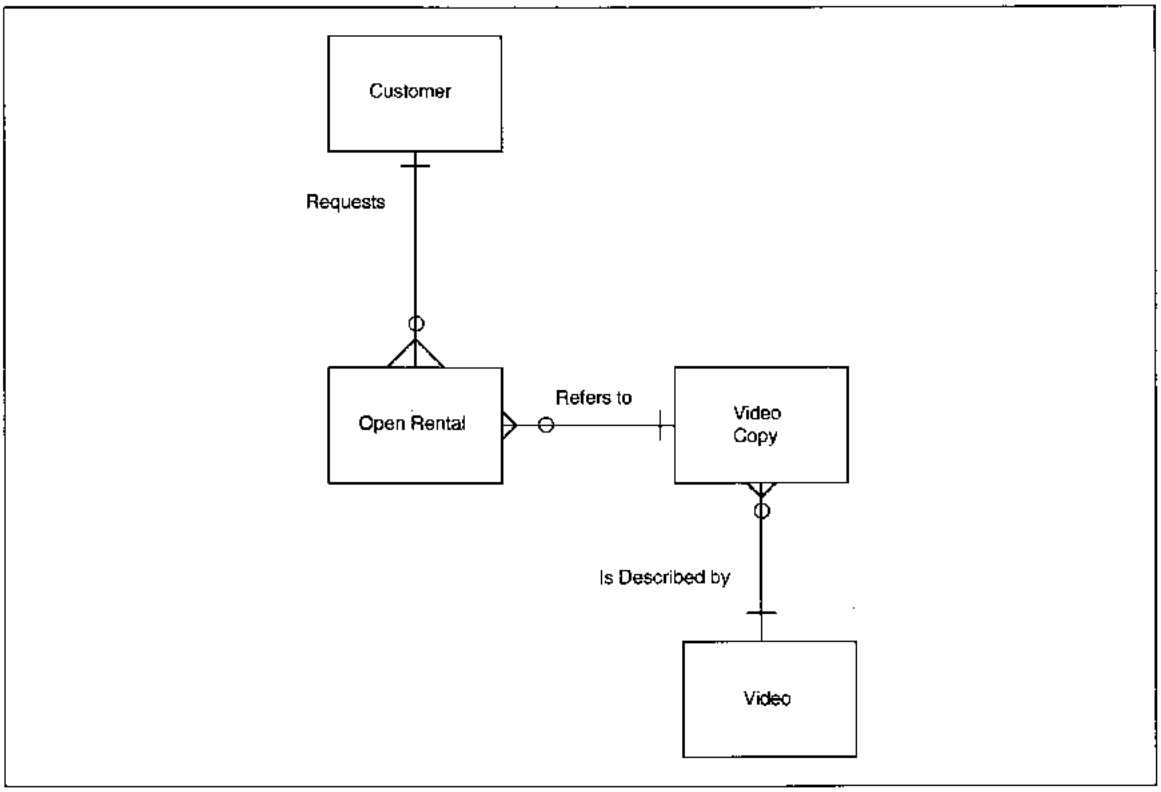


FIGURE 6-3 ABC Entity-Relationship Diagram

advantage include the number and variety of videos available, desirability of the location, and attitude of clerks to customers. For this discussion, we assume that location, attitude and variety of videos are at least equal.

Observation of the applications of the rival video stores is required to assess the potential impact of the subject application. There is a national chain store down the street, approximately .8 miles away. That store is evaluated since it is the closest competition. The chain store sells and rents Nintendo™, Sega Genesis™, and computer software as well as videos; plus, the chain store sells tickets to local rock concerts and events, and sells records, CDs, and audio tapes. Thus, the chain store is a recreational electronics store while ABC is simply a video rental store.

The fact that ABC is specialized and the chain store is general works in ABC's favor because of relative staffing levels. There are usually four clerks working in the chain store. Of the four clerks, two are at cash registers at which lines average three waiting patrons during peak periods. One of the other clerks roams the store assisting customers while the other clerk processes ticket orders. There are frequently lines at the ticket counter, especially when a famous rock group's tickets go on sale. Sometimes there are several hundred people on line. On average, there are 12 customers in the store at all times, with a peak average of 20. The peak times are the same as ABC's—6 P.M. to 11 P.M. Of the 20 customers during peak time, about 10 people actually rent or purchase something. The average age of a rental customer is about 19.

Contrast this situation with ABC. Five clerks work at ABC during the peak hours of 6 P.M. to 11 P.M. The remainder of the time, three clerks are on hand. The clerks, in general, do not roam the store

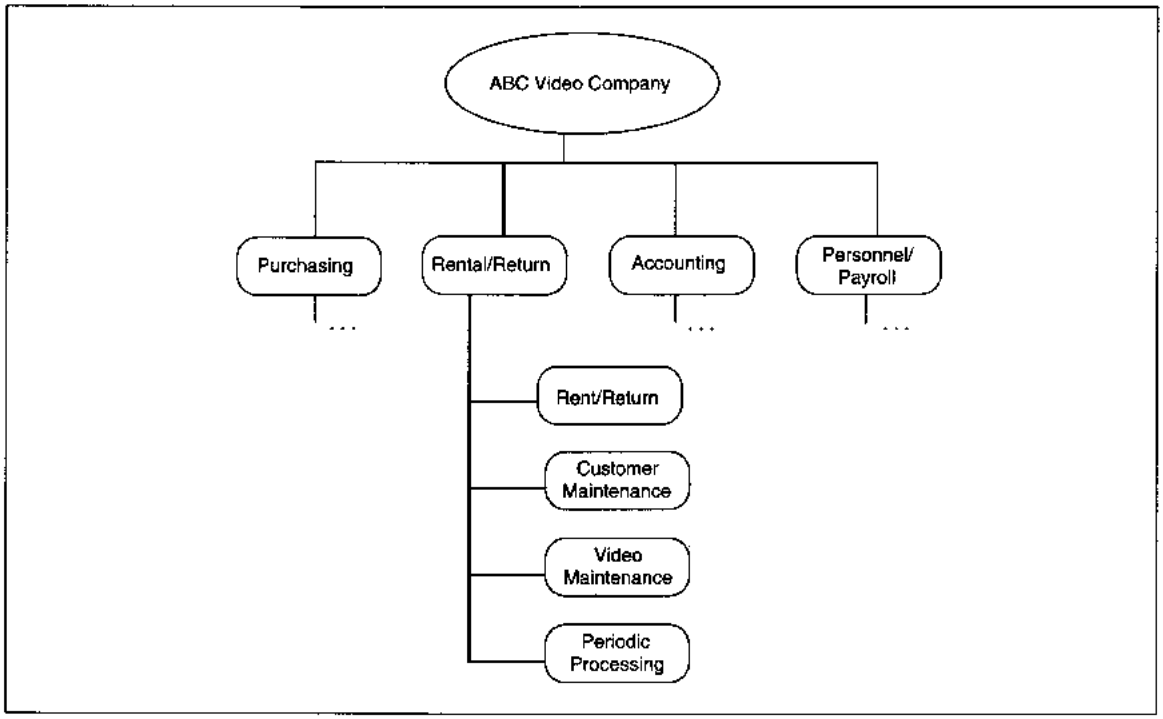


FIGURE 6-4 ABC Hierarchic Process Decomposition Diagram

assisting customers; they are all behind the counter doing payment processing for customer rentals. The lines, if any, form in the peak times and average two people per clerk. If a customer has a question, she or he waits until a clerk is free, then gets assistance and rental payment at the same time. On average, there are five people in the store at all times, with an average of 25 during the peak times. Of the 25 peak customers, 18 rent videos and seven leave empty-handed.

ABC's rental 'hit rate' of .72 (i.e., 18 of 25) is much higher than the industry average of .50.[1] Their single purpose may work against them for some customers who want full service electronic entertainment, and may work for them for other customers who only rent videos. The average age of an ABC rental customer is 22. Thus, the customer is slightly, but not significantly, older than the chain store's customers.

So far, the company contrast neither favors nor disfavors ABC over the chain store. Next, we compare the company's procedures for rental processing. The chain store requires a *subscription* to their company's services that includes the presentation of a valid driver's license and credit card to establish an account. To use the account, each family member is assigned their own number and given his or her own ID card. The ID card is presented at the time of rental and payment of all current and past charges is required for a rental to take place. The presence of a family member ID allows parents who get stuck paying their children's fees to track the guilty party. If two family members make rentals in the same day, the clerk may or may not mention that a rental already exists to the later person. There is no procedure for clerks to help customers control the number of rentals in one day, nor is there a way for previous rentals to be known.

1 The industry average is located by doing library research on the industry.

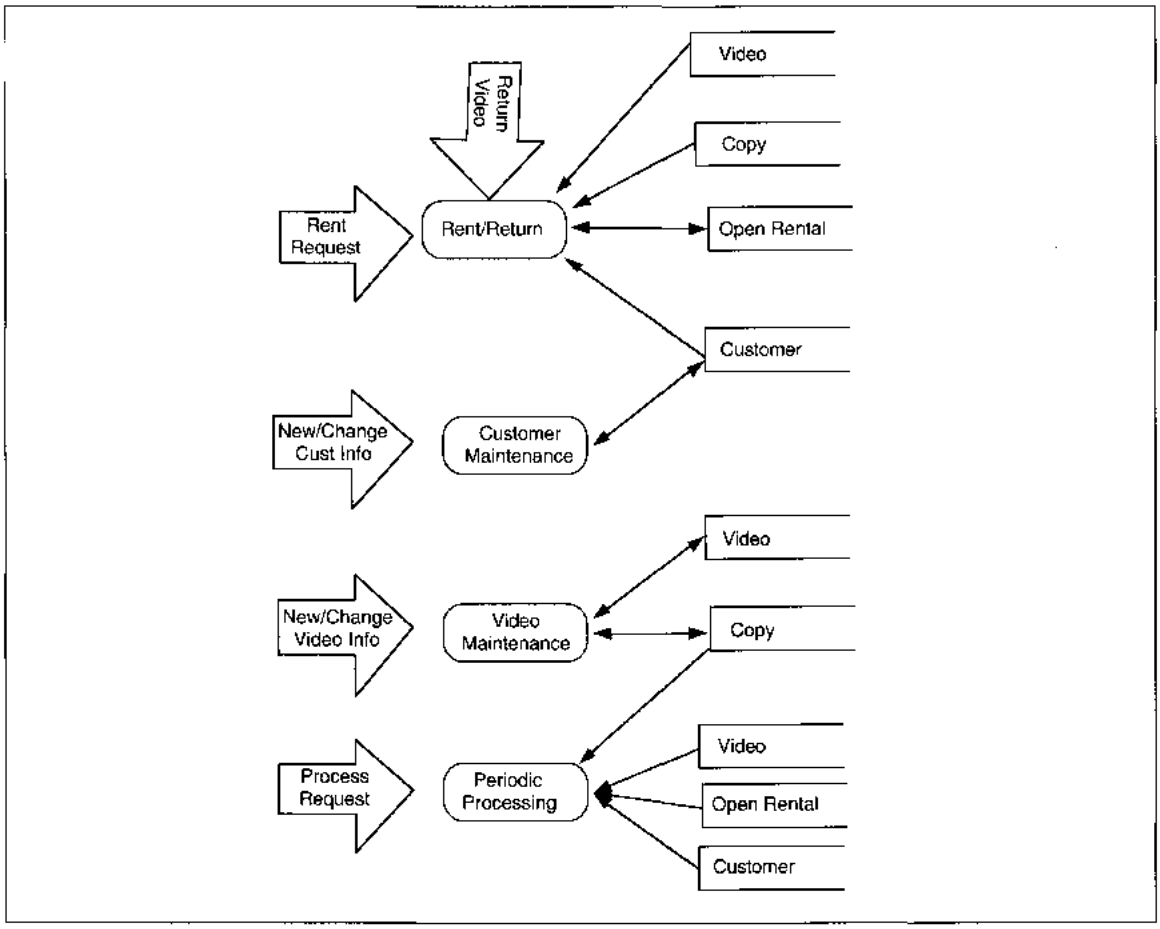


FIGURE 6-5 ABC Process Data Flow Diagram

ABC's expected rental processing is detailed in Chapter 2. Vic's vision of ABC's rental application does slightly favor ABC over the chain store. ABC will also assign family members their own IDs, but an ID card is not required of a customer. Rather, Vic envisions using the telephone number as the ID and asking the person for their name at the time of rental. A list will appear on the screen of all authorized renters for a given phone number with a sequential number the clerk selects beside each name. The procedures to accompany rental processing assume that customers *want to know* if a previous rental that day has occurred. Also, Vic envisions keeping track, electronically, of the previous rentals for a family and giving them the chance to stop a rental transaction on a previously viewed video. Thus, Vic's scenario has less bureaucracy, more service, and more customer-oriented clerical procedures than the chain store. These three improvements are the leverage points for ABC in its local market.

Next, we define other noncompetitive benefits of the application. The application eliminates many of the errors that can happen in a manual system of work. For instance, clerks can no longer decide who pays late fees by changing return dates. Customer cards, which can be lost, are eliminated and replaced by automated file records which can only be deleted by Vic. Both videos and customers must

be *on an automated file* to be eligible for rental processing.

The application will provide for automatic generation of end-of-day reports on receipts and transactions by clerk, by register, or by customer. If a discrepancy is found between receipts and money in the cash register, having a log of transactions that can be printed will assist the accountant in tracing errors. Both of these types of reports provide significant improvement over the current manual methods. Under the current method, receipts are tied to money in each register by sorting the paper copies of transactions and adding the totals. If there is an error, it is almost impossible to trace since no money is actually tied to an individual transaction. At the present, the accountant *writes off* errors.

Developing a list of the benefits for ABC's application is fairly easy because automation so improves a manual operation level task. Take the adjectives *faster*, *better*, *more* and, for each, define all the tasks or data that will be improved in some way relating to the adjective. For instance, processing an individual transaction will be faster because manual card lookup is gone, data entry is minimized to Customer ID and Video ID(s) with the computer retrieving and displaying all other information about each entity. Individual transactions will have improved data integrity by eliminating manual errors, such as writing the wrong amount, entering a wrong amount at the register, writing the wrong tape ID, retrieving the wrong customer card, and so forth. More information will be available for management use. For instance, end-of-day reports provide the accountant more information and Vic might develop ad hoc reports of all automated information. The benefits for ABC's rental application are summarized next.

- Simplify customer IDs—Less bureaucracy than competition
- Provide help to customers in finding tapes—More service than competition
- Give customers information on previous rentals the same day and on videos they have previously rented—More customer-oriented clerical procedures than competition
- Increase data accuracy for customers, videos, rentals
- Allow tracking of late rentals
- Allow accurate computation of late fees
- Increase speed of customer and video information retrieval
- Improve customer service
- Provide accounting record of transactions
- Allow tracking of transaction errors
- Decrease time for individual transactions through minimal typing
- Increase speed and accuracy of fee processing
- Decrease file update time
- Provide more accurate and timely end-of-day reports
- Improve customer satisfaction with overall rental process through the above changes

After general benefits are identified, they are made specific and quantified for evaluation of costs. The benefits listed above are specific enough to quantify directly (see Table 6-1). Quantification, though, requires detailed knowledge of the business and expected benefits. Vic is the business expert and he participates in the quantification activity. For each benefit, he is asked how much revenue (or expense) is related to each item for one occurrence of each benefit. For each, Vic is also asked the degree of certainty for the benefit and his estimate. The numbers provided are multiplied for the total number of each benefit expected. The degree of certainty (ranging from 0.0 to 1.0) is then multiplied by each total amount to provide a range of estimates for each. In the example shown in Figure 6-6, the *benefit* of *more*

Total revenues	$500,000
Losses from inaccurate data	2% of revenues
Dollar loss from bad data	$10,000
Certainty factor	80%
Benefit of more accurate data	.8 * 10000 = $8,000–$10,000

FIGURE 6-6 Example of Benefit Computation

TABLE 6-1 ABC Quantified Benefits

Benefit	Expected Increase in Revenue
Simplify customer IDs—Less bureaucracy than competition	$1,000
Provide help to customers in finding tapes—More service than competition	$1,000
Give customers information on previous rentals the same day and on videos they have previously rented—More customer-oriented clerical procedures than competition	$ 500
Increase data accuracy for customers, videos, rentals	$8,000–10,000
Allow tracking of late rentals Allow accurate computation of late fees	$10,000–15,000
Increase speed of customer and video information retrieval	$1,000
Improve customer service	$1,000
Provide accounting record of transactions Allow tracking of transaction errors Provide more accurate and timely end-of-day reports	$3,000–5,000
Decrease time for individual transactions through minimal typing	$1,000
Increase speed and accuracy of fee processing	$1,500
Decrease file update time	$5,000
Improve customer satisfaction with overall rental process through the above changes	$2,500

accurate data entry, Vic figures his current losses at 2% of total revenues of $500,000, or $10,000. He feels the $10,000 estimate is about 80% accurate. Stated another way, by eliminating errors in data entry, Vic will gain $10,000 with 80% certainty. Thus, the benefit to be gained from more accurate data entry is $8,000–$10,000.

Table 6-1 shows the benefits from the list on p. 156 with dollar values associated with them. For the benefits resulting in $1,000 increases in revenue, Vic was unsure that there was much tangible outcome, but estimated about $3, or one rental, per day. For the higher dollar estimates, he worked through the estimates in the same way shown above for increased accuracy.

Develop Alternative Solutions

The activities in developing alternatives include definitions of technical alternatives, and benefits and risks of each alternative.

Define Technical Alternatives

There are no specific, theory-based guidelines for developing technical alternatives. Rather, the technical alternatives within a specific business are explored to determine what is possible and practical. First, define the application concept (see Table 6-2). How up-to-date does information maintained by the application need to be? If the answer is four

TABLE 6-2 Steps in Developing the Technical Alternatives

- Define the overall application concept
- Evaluate usefulness of existing hardware/software
- If new equipment or software is needed:
  - Determine data sharing requirements
  - Determine the criticality of data to the company
- If shared or critical data, select equipment (either LAN or mainframe) and software that allow centralized control over data.
- If noncritical and nonshared data, select the smallest equipment that allows necessary level of control. In multilocation settings, consider decentralizing or distributing the application by duplicating equipment, application, or data in several locations.
- Define special hardware requirements and ensure that the special hardware works with the selected hardware/software platform(s).

hours or more, a batch application is sufficient. If the answer is between two and four hours, interactive data entry with batch updates throughout the day might be acceptable. If the answer is in the range from seconds old to four hours, an on-line application is also sufficient. If the answer is that the system user must react to all transactions as they occur, a real-time application is needed. On-line is the most frequently selected option.

Next, for individual processes, determine the concept at the lower level of detail. For instance, for reporting, should answers be developed as a report request is entered or can they be run overnight? Some reports might need to be on-line, others might be run in batch mode. The volume of print, estimated time for processing, and urgency of data all are used to select the concept for individual processes. For instance, an ad hoc report that generates 10,000 lines of print should not be sent to a display screen; rather, it should be printed. Also, a long report might be created at the time of the request, but sent to a print queue for convenience of printing. *The decisions made during feasibility are not expected to be permanent at this point, rather, you are estimating the concept to help in the evaluation of complexity of design.*

After the concept is developed, hardware and software are evaluated. If there is hardware and software already installed, investigate their use first. Can the application be developed for operation on the existing equipment? Can the existing software accommodate the application? Can the application coexist with other applications currently used? If the answers to these questions are "yes," the platform recommended is the existing equipment and software. If a "no" answer is given, then investigate new hardware or software as needed.

If no hardware or software are currently used, or the current equipment cannot be used to do the application, select the likely hardware platforms. First, determine whether the application users need to share information or not and how up-to-date the information must be. For instance, can copies of the application run in different locations with daily update of files, or must the users share all information throughout the day? Second, determine the 'corporateness' of the data. How critical to the organization is the application data? If the company depends on the data to stay in business, then a more centralized, controlled environment is required than if the data is not critical to the company.

The need for centralized control over data that is critical to the organization is one factor to consider in recommending a platform and environment for an application. The extent to which the company relies on application operability, the importance of data integrity, audit trails and security, and the ability of the environment to accommodate these needs are all assessed. Although there are no clear differences in application management between a LAN and a mainframe, software *does* make a difference. The levels of security, number of simultaneous users, size of database, locking of records for simultaneous update, and many other technical considerations differ widely across networks, operating systems, databases, and languages. When distribution is an alternative, the centralization issue becomes even more important to evaluate and resolve. Full discussion of the decision criteria for distributing data and applications are deferred until Chapter 10.

To determine hardware alternatives identify the smallest size computer possible that can accommodate the task, providing for data sharing and centralized control as needed. The cheapest and smallest platforms that meet the criteria are alternatives. For hardware we then ask if any other special purpose hardware is needed for this application. If other special purpose hardware is needed, enough research on the hardware should be done to determine what is required and whether or not it can be used with the identified alternatives.

From the hardware identification activity, the most likely platforms should be narrowed to two or three. The key factors in narrowing the selected platforms are reliability and flexibility. Portability might also be important, depending on the environment. **Reliability** is the extent to which the hardware, software, and application will be operational. **Flexibility** is the extent to which the hardware, software, or application can be modified easily. Hardware flexibility relates to the extent to which upgrades can be made, for example the number of additional boards, the maximum memory upgrade, the type bus, and type disk channel, to name a few. Software flexibility relates to package design and how often the vendor releases updates of new functions. Application flexibility relates to methodology, implementation language, and skill of the developers. Reliability and flexibility are important issues in, for example, selecting a PC workstation, because of the diversity and quantity of alternatives available. If you evaluate five different vendors of IBM PC-compatible equipment, you will have different reliabilities and flexibilities for each. But even more confusing is that five different configurations of a PC from *the same* vendor might also have five different reliabilities and flexibilities.

Portability is the extent to which the software can be moved to another hardware/operating system environment without change. The fewer changes when moving the application, the more portable it is. Portability is an issue when the application is developed in one environment (e.g., a LAN) and is ported or moved to another environment for operations (e.g., a mainframe). Portability is also important when an application is developed in one location and is implemented in multiple locations which may not have the same configuration. Multiple locations with heterogeneous environments are the norm in distributed applications.

Hardware alone rarely determines the recommended alternative. In addition to picking hardware platforms that can accommodate the needs for multiple, simultaneous users, you also choose the software most likely to be used in each environment. Again, these selections might change as the design progresses, but their purpose during feasibility is to allow assessment of skills, training needs, cost, and application design complexity.

In choosing software, you identify a programming language, database environment, and any special software needed. Each alternative is developed to solve the entire problem, meeting all requirements and as many optional requests as possible. Only the best alternative(s) for a given environment is considered. Two sets of alternatives illustrate this statement.

The first set of alternatives is for a mainframe environment using different operating environments. The first alternative (see Figure 6-7a) identifies an IBM mainframe, running the MVS operating system, and using IBM's DB2 for database and IMS/DC for telecommunications control. The second alternative (see Figure 6-7b) identifies an IBM mainframe, running the conversational VM/CMS operating system, and using a Focus database. Telecommunications control is hidden from the

Figure 6-7a. Alternative 1

Hardware:	IBM Mainframe 309x
Operating System:	MVS
Database:	DB2
Telecomm Control:	IMS/DC

Figure 6-7b. Alternative 2

Hardware:	IBM Mainframe 309x
Operating System:	VM/CMS
Database:	Focus
Telecomm Control:	SNA through VM

FIGURE 6-7 Two Alternatives Using Different Software

Figure 6-8a. Alternative 1

Hardware:	IBM Mainframe 309x
Operating System:	VM/CMS
Database:	Focus
Telecomm Control:	SNA through VM

Figure 6-8b. Alternative 2

Hardware:	IBM PC-Compatible
Operating System:	MS/DOS, Windows
Database:	Focus
Telecomm Control:	Novell Ethernet

FIGURE 6-8 Alternatives Using Different Operating Environments

application and is through VM (i.e., using VTAM and SNA). Both of these scenarios might be proposed, with the deciding factors relating to time of development and expertise of staff, rather than to the desirability of one environment over the other.

The second set of scenarios is for a network version of an application (see Figure 6-8a) versus a mainframe version (see Figure 6-8b). Both environments would use a database which is already available in-house. In this case, the decision relates to environmental and cost factors since both alternatives use similar database software. Then, reliability, flexibility, and portability are issues.

Estimate Benefits of Recommended Alternatives

Two kinds of benefit estimates are developed. First, the general benefits defined are analyzed to determine that they are (or are not) met by each proposed alternative. Second, new benefits that relate to a specific proposed alternative are defined. Again, benefits are context specific, relating to a given alternative for a given company at a given time.

The first benefits estimate is a tally of the number of general application benefits met and, if it can be determined, the effectiveness of implementation within the proposed alternative. Effectiveness, for our purpose, is the extent to which an alternative will implement the application requirements *more*, *better*, and *faster*. To measure the number of requirements met by each alternative, we simply count which are met in an implementation of each alternative.

To measure effectiveness, we need to determine the extent to which each requirement will be developed. This extent can only be defined in a specific context for a specific application. For instance, two requirements for ABC might be "Provide minimal data entry for customer and video identification" and "Use a scanner for data entry whenever possible" (see Figure 6-9). One alternative might assume the entry of scanned data only. A second alternative assumes the entry of scanned data while providing for keyboard entry in case of scanner hardware failure. A third alternative might assume the keyboarding of a minimal number of characters for each type of data. The first two alternatives meet both criteria. The third alternative does not meet the second requirement. Only the second alternative, however, provides both the requirement *and* a backup. The second alternative would be rated more *effective* in meeting the requirement than the others, while both the first and second alternatives meet the benefits. On a scale of one to three, the alternatives would be rated two, one, and three, respectively. In a different company with a different context, the same alternatives might be rated one, three, two respectively.

Define Risks

The purpose of risk assessment is to determine all the things that can go wrong. If you have heard of Murphy's Laws, you know they apply to applica-

Alternative 1:	Scan Data Entry
Alternative 2:	Scan Data Entry or Keyboard Data Entry with Minimal Typing
Alternative 3:	Keyboard Data Entry with Minimal Typing

FIGURE 6-9 Sample Evaluation of Alternative Effectiveness

TABLE 6-3 Sources of Risk

Source of Risk	Risks
Hardware	Not installed when needed
	Cannot do the job
	Does not work as advertised
	Installation not prepared in time
	Installation requirements (e.g., air-conditioning, room size, or electrical) insufficient
	Wiring not correct
	Hardware delivered incorrectly
	Hardware delivered with damage
Software	Not installed when needed
	Cannot do the job
	Does not work as advertised
	Contains 'undocumented features' that cause compromise on application requirements
	Vendor support inadequate
	Resource requirements are over budgeted, allocated amounts
Group	Key person(s) quit, are promoted elsewhere, go on jury duty, have long-term illness
	Skill levels inadequate
	Training not in time to benefit the project
Project management	Schedule not accurate
	Budget not sufficient
	Manager change
User	Quits, transfers, is replaced
	Not cooperative
	Not supportive
	Does not spend as much time as original commitment requested
Computer resources	Test time insufficient
	Test time not same as commitment
	Inadequate disk space
	Insufficient logon IDs
	Insufficient interactive time

tion development. The three most common of Murphy's Laws are:

1. If anything can go wrong, it will.
2. Things go wrong at the worst possible time.
3. Everything takes longer than it should.

Table 6-3 is a list of possible sources of risk. For each item on the list, you determine the likelihood of it occurring for this project. For instance, if you are using only existing equipment, you could skip the risks dealing with hardware installation problems. As sources of risk are identified, they should be

placed in a separate table and rated for likelihood of occurrence for each alternative. In addition, other possible risks for the project might be added to the list. For instance, if revenue for current year drops 25%, the company might not be able to afford the project.

ABC Video Alternatives

First, technical alternatives for developing ABC's rental application are developed. Next, benefits and risks relating to each alternative are estimated.

To develop technical alternatives, the application requirements should be listed as follows:

1. Provide add, change, delete, inquiry functions for customer, video, and rental information
2. Automate processing of rental transactions, including
   - Interactive processing and data display for all outstanding video rentals, including fees owing
   - The maintenance of customer history of rentals, rental history for each video tape, creation and change of rental transaction records
   - Monitoring of outstanding rentals by customer
   - Computation of late fees owing from prior transactions
   - The ability to create new customers as part of rental processing
   - The ability to add new videos to the system as part of rental processing
   - Query of any rental related information
3. Minimize data entry in rental processing by using bar codes or similar technology
4. Provide interactive, on-line updating capabilities for all files
5. Provide transaction logging for database integrity
6. Do daily backup of all files and application programs
7. Provide ad hoc reporting capability for all files and legal combinations of files (e.g., customer with video rentals with customer rental history)
8. Provide end-of-day reports of activity by transaction with summaries by transaction type (i.e., rental, late fees, other fees)
9. Provide for future growth of 15% per year per file
10. Provide for future growth in number of system users to be one every 18 months for five years. A total of nine concurrent users should be supported.
11. Provide SQL compatibility for future growth and compatibility between software applications
12. Provide mean time between failures (MTBF) of 1 year for hardware selection and mean time to repair (MTTR) of 1 hour in hardware maintenance contracts
13. Provide on-line processing for all functions from 8 A.M. to 11 P.M. daily

ABC has specific requirements that imply an on-line application, significant ad hoc reporting, and interactive processing with immediate file update throughout the day. Batch processing should be feasible as a background task to on-line processing since the on-line portion of the day is so extensive and there might be a problem trying to staff the batch hours. Beginning with a hardware platform, then continuing to software and applications, the proposed alternatives are defined. Only alternatives that can meet all requirements should be identified; however, if that is not possible, any feasible alternatives are identified and evaluated later. In ABC's case, only alternatives that can meet all requirements are identified.

In a small business, the two most likely hardware platforms are multiuser minicomputers or client-server local area networks. These are considered here. The competing hardware platforms are an IBM AS/400 minicomputer versus a token ring local area network (LAN). Each of these decisions requires a minianalysis of the alternatives in their respective environments that are beyond the scope of this text. To specify the LAN, for instance, requires comparison of options and costs of probabilistic versus deterministic networks, cabling requirements, network operating systems (NOS), network interface card

TABLE 6-4 Hardware Platform Estimates[1]

Client/Server Alternative

Item	Cost
Workstation (6)	$ 4,800[1]
Server	$ 2,000
Software	$ 3,500
Cable—Shielded Twisted Pair (STP)	$ 1,900
Network Interface Cards (7)	$ 1,000
Network Operating System (Ethernet), 6–10 stations	$ 2,500
Total	$15,700

Minicomputer Alternative

Item	Cost
Workstation (6)	$ 4,800
Minicomputer	$15,000
Software	$ 5,000 Plus $200/ month
Cable—STP	$ 1,900
Total	$26,700 Plus $200/ month

[1]Keep in mind that these are estimates for the sake of discussion and *not* real dollar estimates.

(NIC), compatible software, and so on. Both hardware platforms can be implemented successfully in ABC's environment, can support the volume of transactions, and can support the expected company and applications growth.

Once the platforms are identified, the hardware cost of implementing the application on the alternative platforms is estimated (see Table 6-4). From these estimates, the most likely (e.g., the cheapest) two to three alternatives are selected. Also, if there is doubt about the economic feasibility of the application, the client/user can determine whether to continue with the analysis or not. As Table 6-4 shows, the client/server LAN is cheaper than the minicomputer hardware alternative. Both alternative definitions exclude software for rental processing which is estimated separately because the option to purchase software versus custom development of software should be evaluated.

The client/server alternative is recommended to Vic and he approves although he is concerned about the cost. As a small business person, his company nets under $1,000,000 per year and, in ABC's case, is closer to $500,000. A rule of thumb in automation expenditures is to spend under 10% of net income. Vic's concern is that the total cost may exceed $50,000 and his financial risk becomes a problem.

The remaining estimates use only the client/server solution to develop software application alternatives. The choices are between purchasing a software package and developing a customized package for rental processing. Mary researches available software packages and finds that the cheapest one is VidRent[2] which costs $7,500 plus $1,500 maintenance per year (see Table 6-5). VidRent will be compared to building a customized applications using either SQL Server[3] or Focus. SQL Server is selected as representing software specifically designed to

2 VidRent is a fictitious name.

3 SQL Server™ is a trademark of Sybase and Microsoft Corporations.

TABLE 6-5 Alternative Software Packages

Software	Initial Cost	Maintenance	Maximum Number of Users
SQL Server™	$17,500[a]	$1,800/year	Up to 20
LAN Focus™ with SQL	$12,000	$1,200/year	Up to eight
VidRent	$ 7,500	$1,500/year	Any number of users on one LAN

[a]Keep in mind that these are estimates for the sake of discussion and *not* real dollar estimates.

take advantage of client/server environments. Focus is selected as representing software with which Mary's team has extensive experience. The costs of each alternative are completely different and provide for different numbers of users. These factors are kept in mind, but the requirements must be analyzed to determine if one software should be favored functionally over the others.

The requirements are reevaluated and rated for each development alternative as shown in Table 6-6. First, consider the softwares' capabilities. VidRent provides neither query capabilities nor historical customer or video processing. It also cannot create new customer or video records as part of rental processing. VidRent also does not provide transaction logging. If this package were chosen, these requirements would go unmet. Through discussion with the vendor, Mary determines that query processing can be done by using any software that can access ASCII files. Thus, the addition of dBase™ or Oracle™ or some other single-user package to provide Vic with query capabilities is a cheap alternative that adds about $1,200 to the alternative. This alternative is still limited in that querying would be limited to an off-line function when the on-line application was not in operation. This requirement is caused by the record locking scheme in VidRent. Also, the software package could be modified by Mary's group to provide the history processing desired by Vic, without violating the vendor warranty. Thus, VidRent's cost increases, and it is capable of doing most requisite processing (see Table 6-7).

Both Focus and SQL Server are fully capable of supporting the application. Both require complete, custom development of the application, but both provide application generators and have built-in query capability. A quick estimate by Mary based on her experience and without a detailed project plan is that the total development work would take about six-person months. At $150 per day, for a 26-day month, the custom software development will be about $23,400 (i.e., 6 * 150 * 26). Except for cost, there is no advantage or disadvantage to either package based on application requirements. SQL Server's license allows 15 concurrent users which is more than Focus.

Next, consider the organizational impacts of each package. Mary's team requires training for either VidRent or SQL Server. Training for SQL Server, which is supplied by the vendor, would not be charged to Vic since the knowledge is useful to the team after the rental application is complete. VidRent training, also from the vendor, would be paid by Vic. Training costs must be added to its cost (see Table 6-7).

Next, consider vendor reputation and market stability. SyBase and Microsoft, the vendors of SQL Server, are both relatively young companies, with Microsoft the current leader in software for the PC market. Focus' company, Information Builders, Inc., is over 15 years old and has enjoyed steady growth. Therefore, both vendors are expected to remain viable market forces for the foreseeable future. VidRent's vendor, VidSoft, is 5 years old and still is run from the owner's home. The company has grown steadily by selling to the single video store firms such as ABC, but the owner, Mark Denton, does not publicize his earnings.

In summary, SQL Server and Focus both meet all software requirements of the application; VidRent could be made to provide most requirements. Cost favors the VidRent proposal with a total estimated software cost of $22,000. At this point, Vic must decide how much he wants the custom features of his application and whether the compromises on querying and ease of processing are worth $13,000.

Vic and Mary discuss the alternatives frankly. Mary recommends not going with VidRent because of the company size, lack of features, and need for customizing for any features not already in the package. Vic is staggered by the cost of custom software development and is inclined to purchase VidRent and forget his grand plans. Mary reminds him that if he does not develop his application as envisioned, the competitive advantages might disappear. Vic eventually decides that he does want the application as currently defined and that he is not willing to compromise his vision in any way. Therefore, only SQL Server and Focus alternatives are developed further to determine the benefits and risks of the softwares.

Only general benefits are evaluated for each alternative; there are no apparent benefits of one

TABLE 6-6 Rating Software Development Alternatives

Function	SQL Server	Focus	VidRent
Provide add, change, delete, inquiry functions for customer, video, and rental information	Yes	Yes	Yes
Interactive processing and data display for all outstanding video rentals, including fees owing	Yes	Yes	Yes
On-line processing from 8 A.M. to 11 P.M. daily	Yes	Yes	Yes
The maintenance of customer history of rentals, rental history for each video tape, creation, and change of rental transaction records	Yes	Yes	Yes
Monitoring of outstanding rentals by customer	On-line	On-line	Off-line
Computation of late fees owing from prior transactions	Yes	Yes	Yes
The ability to create new customers as part of rental processing	Yes	Yes	No
The ability to add new videos to the system as part of rental processing	Yes	Yes	No
Query of any rental-related information	On-line	On-line	Off-line
Minimize data entry in rental processing by using bar codes or similar technology	Yes	Yes	Yes
Provide immediate file update	Yes	Yes	Yes
Provide transaction logging for database integrity	Yes	Yes	No
Do daily backup of all files and application programs	Yes	Yes	Yes
Provide ad hoc reporting capability for all files and legal combinations of files	Yes	Yes	Only with another package
Provide end-of-day reports	Yes	Yes	Yes
Provide for growth of 15% per year per file	Yes	Yes	Yes
Provide for nine concurrent users	15	10	Any number
Provide SQL compatibility	Yes	Yes	For ASCII files
Total requirements met out of 18	18	18	15

software over the other. The benefits of the application identified in an earlier step are compared to each proposed software alternative. As you can see from Table 6-8, the benefits are identical for each implementation.

Finally, risks of the alternatives are defined. The list of possible risks is customized for the application and each risk is assessed for probability of occurrence with a specific alternative (see Table 6-9). The table of risks is repeated here with an analysis of the two language environments. Hardware risks apply equally to both alternatives. Software risks vary because of differences in product knowledge by the development team, product functionality, and expected cost, all of which favor Focus.

TABLE 6-7 Total Estimated Cost of Software Alternatives

Software	Initial Cost	Purpose
SQL Server™	$17,500[1]	License fee
	$23,400	Custom software
		Total $37,900
LAN Focus™ with SQL	$12,000	License fee
	$23,400	Custom software
		Total $35,000
VidRent	$ 7,500	License fee
	$ 2,500	Database query software
	$ 5,000	Training
	$ 7,000	Customizing
		Total $22,000

[1]Keep in mind that these are estimates for the sake of discussion and *not* real dollar estimates.

TABLE 6-8 Benefits of SQL Server and Focus Alternatives

Benefits	SQL Server	Focus
Simplify customer IDs	Yes	Yes
Provide help to customers in finding tapes	Procedure	Procedure
Give customers information on previous rentals the same day and on videos they have previously rented	Yes	Yes
Provide data accuracy for customers, videos, rentals	Yes	Yes
Track and display late rentals	Yes	Yes
Compute and display late fees	Yes	Yes
Increase speed of customer and video information retrieval	Yes	Yes
Improve customer service	Yes	Yes
Provide accounting record of transactions	Yes	Yes
Allow tracking of transaction errors	Yes	Yes
Provide accurate and timely end-of-day reports	Yes	Yes
Decrease time for individual transactions through minimal typing	Yes	Yes
Increase speed and accuracy of fee processing	Yes	Yes
Decrease file update time	Yes	Yes
Improve customer satisfaction with overall rental process through the above changes	Yes	Yes
Total benefits met out of 15	15	15

TABLE 6-9 ABC Risks of Software Development Alternatives

Risks	SQL Server	Focus
Hardware not installed when needed	Low	Low
Hardware cannot do the job	Low	Low
Hardware does not work as advertised	Low	Low
Hardware installation not prepared in time	Low	Low
Hardware installation requirements (air conditioning or electrical) insufficient	Low	Low
Wiring not correct	Low–Medium	Low–Medium
Hardware delivered incorrectly	Low	Low
Hardware delivered with damage	Low	Low
Software not installed when needed	Low	Low
Software cannot do the job	Low	N/A
Software does not work as advertised	Low–Medium	N/A
Software contains 'undocumented features' that cause compromise on application requirements	Medium	N/A
Software vendor support inadequate	Low–Medium	N/A
Software resource requirements are over budgeted, allocated amounts	Low	N/A
Key person(s) quit, are promoted elsewhere, go on jury duty, have long-term illness	Low	Low
Group skill levels inadequate	Low–Medium	No
Training not in time to benefit the project	Low–Medium	N/A
Schedule not accurate	Low	Low
Budget not sufficient	Low	Low
Manager change	No	No
Vic quits, transfers, is replaced	No	No
Vic/clerks not cooperative	Low	Low
Vic/clerks not supportive	Low	Low
Vic does not spend as much time as original commitment requested	Low	Low
Test time insufficient	N/A	N/A
Test time not same as commitment	N/A	N/A
Inadequate disk space	N/A	N/A
Insufficient logon IDs	N/A	N/A
Insufficient interactive time	N/A	N/A

Once the benefits, risks, and alternatives are defined, they are evaluated to narrow the field to one (or two) proposed alternative(s).

Evaluate Alternative Solutions

The recommended alternatives are evaluated for technical adequacy, organizational feasibility, extent to which benefits are met, and severity of associated risks. In general, we select the alternative that meets the most requirements, yields the greatest benefit, and has the lowest associated risk. When these characteristics do not relate to the same technical alternative, one or two are selected for further analysis and the remaining alternatives are eliminated from consideration. In this section, we discuss technical, organization, benefit, and risk evaluations for narrowing the decision to one or two alternatives.

Evaluate Technical Feasibility

Technical feasibility assesses the technology, its maturity in the market, its availability to the company, and the likelihood of successful use. Technical feasibility is most important when using new technologies that are *leading edge*. You want to be *leading* the competition, not *bleeding*, when using new technologies!

The key questions used to evaluate technical feasibility are:

- Is the technology in use elsewhere?
- Is the technology used elsewhere for similar applications?
- How mature is the technology?
- How much industry experience is there with this technology?
- Are staff with experience using this technology easy to find?
- How does each alternative manage the application sources of complexity?
- Does the proposed alternative require any compromise of application requirements? What type of compromise and which requirement(s)?

Each question is evaluated for each technical alternative proposed. Any issues about a technology's ability to perform as required for an application should be identified. Objective answers that may not be what managers want to hear are required to adequately assess technical feasibility. Maintaining objectivity is difficult when market pressure to develop an application exists and managers *want* to develop an application.

To perform technical feasibility analysis, the technical alternatives are listed and compared across alternatives. Then, the application requirements are listed and evaluated for number of requirements met across the alternatives. The alternative meeting the most requirements is favored during this analysis. If there is a difference in the extent to which a requirement would be met, that information is noted in the analysis.

Evaluate Organizational Feasibility

Organizational feasibility is the extent to which the organization is ready to implement the proposed application. First, using the questions below, organization structure is assessed to define organizational changes required.

- Does the organization structure need to be changed?
- Do all groups that create the same information report to the same manager?
- Do user jobs require new procedures?
- Do user jobs require new work organization? For instance, do they move from individual assembly line-type arrangement to work groups?
- Do users have the required level of computer literacy?
- Do users have the required level of typing skills?
- Will users require training for the new application?
- Can training be done by other users?
- Are users involved in screen design, acceptance test design, and/or general application development?
- Does the IS staff know the problem domain?
- Does the IS staff know the software being used?
- Does the IS staff know the operating environment being used?

Organization structure is evaluated to determine if the people who have creation authority for data all report to the same management and that all departments and jobs that will be needed in the new application are defined or currently exist. Second, expected users are evaluated to determine the extent to which training is required to implement the proposed application. For instance, some computer literacy and typing skills might be required. If users must know how to turn the machine on and activate an application, but do not currently use computers, you might need to do a short questionnaire or interview users to determine their level of computer literacy. Any needs identified are added to the implementation plan as a task (and cost) of the proposed application. The goal of this first type of organization analysis is to identify user department changes and user requirements for training, both of which must be satisfied before the organization can effectively use the proposed application.

A second type of organizational feasibility assesses the readiness of the IS organization to develop the proposed application. When a custom development is being done by consultants, you evaluate their skills with the technology and similar problems to determine their readiness. The assessment determines staff skill with the hardware, operating environment, programming language, database, and similar environments. As with the user organization, feasibility, level of expertise and training requirements are determined. Technical staff training requirements defined during this assessment are added to development plans for cost analysis.

The last type of feasibility assessment, financial feasibility, is performed after a plan for the recommended alternative(s) is developed. Financial feasibility is discussed in a following section.

Assess Benefits

Benefits defined for the application in general, and for specific implementation alternatives, are assessed to determine which proposed alternative yields the outcome with the highest reward to the organization.

Benefits are tallied for each alternative. First, a simple count of the benefits for each alternative is done. Then, for benefits assigned monetary values, the amounts of increased revenues or avoided expenses are summed to provide a single dollar-value benefit for each alternative. If there are no alternative-specific benefits, the number and value of benefits are the same for all alternatives. If there are alternative-specific benefits, then one or several alternatives might be preferred. These are identified by this analysis.

Assess Risks

Similar to the benefits analysis, the risks of each proposed alternative are assessed to determine the alternative with the least risk. First, a simple count of the risks for each alternative is done. Then, for alternative-specific risks, the extent to which they are likely to occur is assessed. If there are no alternative-specific risks, the risks are the same for all alternatives. When the risks are not the same, alternatives with lower, less likely risks are preferred to alternatives with a high likelihood of occurrence. If a dollar value of exposure is assigned to the risk, it is considered, with lower values of risk preferred to significant potential losses.

Propose New Application

Next, the recommended solution(s) are defined in sufficient detail to allow project planning and financial analysis. The development plans include hardware, software, operating environment, development concept, technical feasibility, organization feasibility, benefits, and risks.

The proposal of the new application might document the recommendations formally to begin to develop the feasibility report, or may still be an informal collection of information that supports the remaining analyses. The formality of this gathering of information is decided by the Project Manager and SE, based on their confidence in their decisions. If they are fairly confident that no major changes will take place, they might develop final versions of documentation and begin an informal review of their findings and recommendations with users.

ABC Video Evaluation of Alternatives

The alternatives first are assessed in terms of the technical and organizational feasibility. Then, the benefits and risks of each are assessed. Based on

the differences between alternatives, a recommended solution is selected.

Both packages, SQL Server and Focus, appear capable of providing the complete application as envisioned by Vic. The implementation would probably be smoother with Focus given the high skill level of Mary and her staff with the product. SQL Server might have intangible benefits in that, if another store were opened, the software could easily communicate between stores, having been built specifically for distributed processing. This benefit is not immediate, however, and the current technical solution favored is Focus. Focus has a longer history, and is thus, a more mature product, has a large company backing it, provides all technical requirements for current and future plans; and is cheaper than SQL Server in the example.

From an organization perspective, neither product offers any distinct advantages or disadvantages. The staff at ABC would have to learn both products. Both vendors offer classes in the Atlanta area. The company does not need reorganization to accommodate the application regardless of software chosen. From the perspective of Mary's staff, Focus is preferred since they already have experience using it, but she feels confident that they could also build the application using SQL Server if desired.

The benefits analysis is simple in this case. The benefits do not favor either implementation scenario since they all apply to both. Thus, all benefits are expected to accrue from either implementation.

The risk analysis favors Focus over SQL Server slightly. The main difference in risk exposure is from the lack of usage experience of Mary's group with SQL Server. This lack of knowledge can only be partially removed by training. Experience in using the product is really required to develop knowledge of the 'undocumented features' and unanticipated limitations of the software. In this case, Focus is known to Mary's team and is therefore preferred.

In the example for ABC, both packages could probably be used with success in developing the ABC rental application. Both softwares appear capable of future growth and have apparent company stability. The cost differences favor a Focus solution, while the specific client/server orientation provides an as yet unneeded benefit to SQL Server. Vic decides in favor of the Focus solution, but is clearly unhappy with the overall cost of $50,700. Vic wants to continue with the planning and financial analysis for the application, but is also interested in some way to reduce or defer the development costs of Mary's team services for customized software. In any case, the Focus, LAN solution will be planned and evaluated financially in detail. Before we continue with ABC's problem, we first talk about project planning.

Plan the Implementation

Estimating Techniques

Users are easy to deal with when they feel you understand their problem, when they think you can improve their situation through automation, you can estimate how long the job will take, and you can estimate their costs. These are not easy items to know or to develop. When users are comfortable that they can afford and use the proposed application within a reasonable amount of time, they become the champions of the project, fighting for its development in the political environment of the business. Research shows that a champion provides a major contribution to application development success. In this section, we discuss the last two important issues to making the user feel comfortable: planning and costing the project.[4]

Accurate estimates are important to

- allow cost-benefit and other financial analyses
- allow hardware/software trade-off analysis
- provide a basis for management evaluation of multiple projects
- act as the basis for schedule, staffing, project management, and structure definition
- avoid problems such as contract renegotiation, overtime, user cost increases; or project costs increases

At the feasibility level, estimates should be within 20% accurate. This means that the estimates might be overstated or understated by 20%. Planning

4 All the methods in this section are based on methods discussed in Barry Boehm's book, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

should be redone at the end of the analysis phase, at which time the estimates should be within 10%. Again, planning at the end of design should refine the estimates to within 5%. The redefinition of costs is one activity that meets with resistance from managers who tend to *cast in concrete* the first estimate they hear. Part of the Project Manager's role is to educate the managers and users involved to understand that as the degree of uncertainty about project activities decreases, the certainty of time estimates and costs increases. Therefore, the plans should be redone at the end of every major phase of activity.

The planning methods discussed in the next section are ways to generate time estimates for the person-days of project work. These are then converted into costs by allocating an amount of money for each person required. Ultimately, the Project Manager and SE rely on their knowledge of the organization and salaries of individuals. Additional costs are allocated for computer resources, acquisition of hardware, software, or consultants, and other supplies needed to complete the application.

There are many different approaches to planning which are discussed in the first section below. After that, we take a practical, experience-based approach to developing a critical path plan. The experience-based estimates are then *reality checked* against two sets of algorithmic planning formulae. The two planning methods used are function points and the CoCoMo model. Both have known flaws. By combining planning methods rather than using only one, you improve the likelihood of more accurate estimates.

Planning methods are usually classified into categories for algorithmic methods, expert judgment, analogy, Parkinson, price-to-win, top-down, bottom-up, or function points. These are defined here, and several methods are discussed in detail because they are the most frequently used. Advantages and disadvantages of each method are summarized in Table 6-10.

ALGORITHMIC METHODS. An **algorithmic estimating** relies on one or more key formulae to develop an estimate of person-power required for project work. There are five types of algorithmic planning methods. The sequence in which they were developed and found to be inadequate is linear (see Figure 6-10), multiplicative (see Figure 6-11), analytic (see Figure 6-12), tabular (see Figure 6-13), and composite, which combines the others. All but the composite method are rarely used because they offer too simplistic a model of project work. The noncomposite methods do not support adjustment of the model for expertise of staff, tools used to aid development, or other factors that might alter the time and cost of development. All algorithmic methods suffer the same *fatal flaw* that they rely on some initial estimate that is difficult to guess and on which the accuracy of the entire estimate rests.

There are two key variables in the Composite Cost Model (CoCoMo): number of delivered source instructions and project *mode*. **Delivered source instructions** refers to lines of code used in a production version of an application and omits any modules or programs written to support the development effort. Since any sizable project has thousands of instructions, this term is expressed as thousands of delivered source instructions or **KDSI**. *Delivered* instructions are those that actually are in the finished product and excludes any code that is generated to facilitate project development. For instance, in a DBMS application, you frequently write programs to do a formatted print of the file that are not part of the finished application. These modules would be omitted from the estimate. The second important word is *source*. Source code means uncompiled, unlinked lines of code in whatever language is used. The implication is that some *compiled* language such as Cobol, Fortran, Pascal, or PL/1, is used. Control language code is omitted from KDSI, while the number of Cobol statements is reduced by a factor of .33 to compensate for the high percentage of nonexecutable code.

The model is based on three critical assumptions. First, it assumes that KDSI can be estimated with some accuracy. Second, it assumes that the waterfall life cycle approach is used. Third, the language of application development (Cobol, Pl/1, APL, and so on) is assumed to have no discernible impact on the amount of effort or staffing for a project. The latter two assumptions can be corrected for by the multipliers. The first assumption, that accurate estimates of KDSI are possible, is only true when projects are

TABLE 6-10 Advantages and Disadvantages of Estimating Techniques*

Method	Advantages	Disadvantages
Algorithmic	Objective, repeatable, efficient, analyzable formula Good for sensitivity analysis Objectively calibrated to experience	Subjective inputs Does not accommodate exceptional circumstances Assumes history predicts future applications
Expert Judgment	Assessment of representativeness, interactions, and exceptional circumstances can be factored into the judgment	No better than participants Biases, incomplete recall Representativeness of experience
Analogy	Based on experience	No better than participants Biases, incomplete recall Representativeness of experience
Parkinson	Might relate to experience	Reinforces poor practice
Price to Win	Often wins the contract	Produces large overruns Unethical misrepresentation of information
Top-Down	System level focus Efficient use of resources	Less detailed and stable than other methods Overlooks technical complexity
Bottom-Up	More detailed basis More stable than top-down Fosters individual commitment when individual estimates own work	May overlook system level complexity and costs Requires more effort than most other methods
Function Points	Objective, repeatable, objective inputs	Based on history Must be calibrated Focuses on application externals

*Adapted from Boehm, Barry W., *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981, p. 342.

similar over time, and accurate statistics of past project KDSI are maintained.

Project mode refers to a combination of size, staff, and technology. The three main project modes are organic, semidetached, and embedded (see Table 6-11). An **organic project** is developed by in-house staff, is small to medium in size, and uses existing, familiar technology.

A **semidetached project** is one that is developed by in-house staff and contractors, is intermediate to large in size, and uses technology that is familiar to some of the project team.

An **embedded project** is one that is developed by contractors, is medium to very large in size, and uses state-of-the-art technology which is new and unfamiliar to all project members.

The five project sizes referenced by CoCoMo are small, intermediate, medium, large, and very large. Each size has an average number of thousands of source instructions to which it relates (see Table

Effort = $Ao + A1X1 + \ldots AnXn$

Where An = Weight
Xn = Source of Cost n (e.g., Personnel time)

Ex.:

Effort = –3.6

+9 (2)	High Uncertainty of Requirements
+10.7 (2)	Unstable Design
+55.7 (1)	Concurrent Hardware Development
+15 (1)	New Technology
+29.55 (1)	Multiple Target Hardware Platforms
+2.2 (.6)	Percent I/O
+.52 (.4)	Percent Match Instructions
= 137.58	Person Months

FIGURE 6-10 Linear Estimating Formula and Example

Effort = $Ao\,A1^{x}\,A2^{x2}\ldots An^{xn}$

Where An = Source of Cost n (e.g., Personnel time)
xn = –1, 0 or 1 depending on presence of cost

Ex.:

Effort = .6 *

$*.95^{1}$	High Uncertainty of Requirements
$* 1.7^{1}$	Unstable Design
$* 5.5^{1}$	Concurrent Hardware Development
$* 1^{0}$	New Technology
$* 2.55^{1}$	Multiple Target Hardware Platforms
$* 100^{1}$	Person Months Test Code
= 1359	Person Months

FIGURE 6-11 Multiplicative Estimating Formula and Example

$N1N2N\log 2N / 2SN2$

Where:

N1	=	Number of Program operators (e.g., Add)
N2	=	Number of Program operands (e.g., Data Fields)
N	=	N1 + N2
S	=	Approximately 18
N2	=	ΣN2 usage, i.e., the number of time the operands are used in instructions
N	=	ΣN1 + ΣN2 usage

Example: If

N1	=	30
N2	=	1000
N	=	N1 + N2 = 30 + 1000 = 1030
S	=	Approximately 18
N2	=	ΣN2 usage = 2500
N	=	ΣN1 + ΣN2 = 1000 + 2500 = 3500

then $N1N2N\log_2 N / 2SN2$

= 30 * 2500 * 3500 $\log_2$ 1030 / 2 * 18 * 1000
= 75000 * 4.5 / 36000
= 9.1 Person Months

FIGURE 6-12 Analytic Estimating Formula and Example

Estimate number of functions by type.
Estimate number of LOC for each function.
Table lookup of productivity.
Sum all time.
Distribute according to table formula.

Type	MM/1000 LOC*
Math	6 MM
Report	8 MM
Logic	12 MM
Signal/Process Control	20 MM
Real-Time Control	40 MM

Example:

5	Math functions	=	2000 LOC
15	Reports	=	8000 LOC
25	Logic functions	=	5000 LOC
6	Signal control functions	=	1200 LOC
0	Real-time control	=	0 LOC
=	(2*6) + (8*8) + (12*5) + (20*1.2)		
=	12 + 64 + 60 + 24		
=	160 MM		

*MM = Person Months
LOC = Lines of Code

FIGURE 6-13 Tabular Estimating Formula and Example

TABLE 6-11 Three CoCoMo Project Modes

Organic	In-house developed
	Small–medium size
	Existing, familiar technology
Semidetached	Partially in-house and partially contractor developed
	Intermediate–large size
	Existing, familiar technology
Embedded	Contractor developed
	Medium–very large size
	State-of-the-art, unfamiliar technology

TABLE 6-12 Five CoCoMo Project Sizes

Size	Thousands of Lines of Source Code
Small	2
Intermediate	8
Medium	32
Large	128
Very Large	512+

From Boehm, Barry W., *Software Engineering Economics.* Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981, p. 75.

6-12). Tables of the estimates, completed for each of the standard sizes, are provided in Boehm's book. These sizes provide a guide for calibrating nonstandard KDSI estimates.

To use CoCoMo, the mode is defined, KDSI are estimated, the formula for the matching project mode is computed. Table 6-13 shows the CoCoMo 'basic' formulae for each mode. The appeal of such a simple model is obvious. The model is reusable, objective, and simple to learn and use. The model's major source of uncertainty is in the need for an accurate estimate of KDSI. This difficulty of accurately estimating KDSI should not be minimized.

TABLE 6-13 CoCoMo Basic Formulae

Mode	Effort	Schedule
Organic	$MM = 2.4(KDSI^{1.05})$	$TDEV = 2.5(MM^{0.38})$
Semidetached	$MM = 3.0(KDSI^{1.12})$	$TDEV = 2.5(MM^{0.35})$
Embedded	$MM = 3.6(KDSI^{1.20})$	$TDEV = 2.5(MM^{0.32})$

MM = Person Months
TDEV = Time of Development

From Boehm, Barry W., *Software Engineering Economics.* Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981, p. 75.

Next the multipliers are evaluated and used to modify the person-month estimate based on project specific factors (see Table 6-14). Risks, uncertainties, constraints, and staff experience are all evaluated to determine their potential impact on the schedule. The basic person-months estimate is multiplied by each relevant subjective multiplier to adjust for project contingencies.

Total months of effort is not very useful for a multiperson project unless there is also some way to know how much elapsed time the project should take and when to phase people onto and off of the project. CoCoMo provides these estimates. The second set of formulae are used to estimate total development time (TDEV) which accounts for multiple people working on the project. (Table 6-13 also shows the algorithms used to compute development effort.) To use these algorithms, you simply plug in the person-months value from the first formula into the TDEV formula matching the project mode.

Finally, the CoCoMo model includes a formula to estimate staffing levels over time in the shape of a Rayleigh (pronounced RAY-lee) curve. A Rayleigh curve (Figure 6-14) starts at some point above zero, increases to a high point, and gradually decreases to near zero. The formula for developing the number of people at any time requires an estimate of the time of the highest staffing level for the project (see Figure 6-14). This formula assumes a peak about one-third of the way into the elapsed time (TDEV).

The advantages of any formula for estimating is that it is objective and repeatable (see Table 6-10). Further, they are easily understood and require little effort to use. The disadvantages are that the formulae all require some initial estimate that is *hard* to develop and frequently inaccurate. The formula might not fit the project and may be complicated to learn.

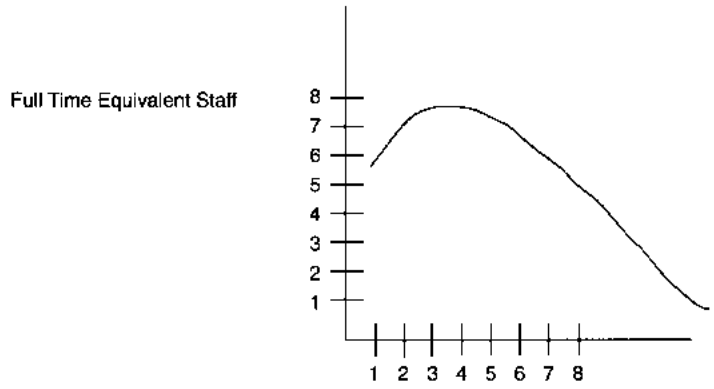
TABLE 6-14 Sample CoCoMo Multipliers

Type Variance	Range of Multiplier
Product	
Reliability	.75–1.4
Data Base Size	.94–1.16
Software Complexity	.70–1.65
Computer	
Execution Time	1.00–1.66
Memory Constraints	1.00–1.56
OS Volatility	.87–1.3
Turnaround Time	.87–1.15
Project	
Modern	.82–1.24
Practice	
Use of Software Tools	.83–1.24
Schedule Constraints	1.10–1.23
Personnel	
Analyst Capability	.71–1.46
Programmer Capability	.70–1.42
Application Experience	.82–1.29
Operating System Experience	.90–1.21
Programming Language Experience	.95–1.14
Rate Each Cost Driver on a scale of 0 (Not applicable) to 5 (Highly applicable)	
Multiply rating times multiplier to obtain final multiplier	
Multiply MM Computation by final multiplier	

From Boehm, Barry W., *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.

EXPERT JUDGMENT. **Expert judgment estimating** is a technique by which the Project Manager and SE use their experience to guide the development of the time estimates. Each task is defined in terms of the program types likely to result from the task. Then, using their experience, the PM and SE assign times to each program, adding design time and analysis time.

For instance, assume there are 15 report programs. If a batch Cobol report interfacing with a DBMS averages one week to code and unit test, 3–5 days of design, and 2–4 days of analyses, then 15 reports will average 15 weeks for programming and one week is allocated per program. The other phase estimates are similar. A range of 30–60 days of analysis and of 45–75 days for design are allocated for the 15 reports. Similar estimates are made



$$FTE_t = MM\left(\frac{0.15TDEV + 0.7t}{0.33(TDEV)^2}\right) e^{-\frac{(0.15TDEV + 0.7t)^2}{0.33(TDEV)^2}}$$

where MM is man-months
TDEV is total development time
t is the period for which the estimate is made
FTE_t is *F*ull *T*ime *E*quivalent staff in time t

*Adapted from Boehm, 1981.

FIGURE 6-14 Rayleigh Curve of Staffing Estimates

for batch updates, on-line queries, on-line updates, and so forth.

When all program estimates are complete, the entire group is summarized to develop a project estimate. These are then presented as a range of estimates with the lowest number representing the optimistic schedule, the average number representing the most likely schedule, and the highest number representing a pessimistic schedule.

Costs are similarly assigned. Each program type is used to define the skill level of the desired programmer. For instance, a junior programmer might be assigned to batch reports, a senior programmer assigned to on-line processing, and a mid-level programmer to on-line reports. The times for each program type and programmer type are summed and multiplied by the cost of that level person. Similarly, the level of analyst or programmer-analyst needed for analysis and design of the tasks is estimated. Finally, all costs are summed to develop a total cost for the project.

The advantages of expert judgment are the ability to factor experience into estimates, to tailor estimates to assigned personnel, and to develop estimates quickly and efficiently (see Table 6-10). The disadvantages are that the estimates are no better than the expertise of the PM and SE, they may be biased, are hard to rationalize, and not objectively repeatable. That is, the experience cannot be taught to others so two PM/SE teams estimating the same project will develop different estimates for the same problem. Finally, expert judgment is not useful in novel situations using new technology, methodology, or languages.

ANALOGY. Analogical estimating is similar to applying experience. In **estimating** by **analogy**, a recently completed similar project is selected to act

as a prototype baseline for developing cost estimates for a current proposed project. Costs are determined based on the match or mismatch of tasks and programs to the baseline. In other words, if a task is essentially the same, then the actual time of the task in the baseline project is used to estimate the actual time of the task for the proposed project. Analogy is applied to time, staff skill levels, and, eventually, resource, hardware, software, and other costs.

The advantage of analogy is that it is based on an actual, recent experience which can be studied for specific differences and only those differences require new cost estimates (see Table 6-10). The disadvantages of analogy are that the analogous project may not be representative of the proposed project, constraints, techniques, or functions. Some of the disadvantages can be reduced by matching project functions. This technique might work in large companies with many similar projects, but is not particularly useful in small companies, unique projects, or projects using new technology, methodology, or languages.

PARKINSON'S LAW. **Parkinson's Law**[5] states that "Work expands to fill the available time." Based on this law, any time can be allocated and that is the time the project will take (see Table 6-10). For instance, there are 6 people available for 6 months, therefore the project will take 36 person-months. This is a cynical view of estimating that reinforces poor development practices by random assignment of time and people.

There are obvious flaws to Parkinson's Law. This method is likely to be grossly inaccurate in estimates generated (see Table 6-10). If people are allocated because they are available and not because they are needed, their skills are likely to be wasted and the project is more likely to be late. This method is *not* recommended.

PRICE-TO-WIN. **Price-to-win** is a consultant strategy that uses a low estimate to obtain a job, with the implication that the time and cost will later be renegotiated. Like Parkinson's Law, this strategy is *not* recommended. Price-to-win leads to forced user compromise on application requirements to try to meet a cost/time estimate, gives the consulting company bad public relations, always requires staff overtime, and most always results in cost overruns for both time and money.

You might ask, Why would anyone ever use a price-to-win strategy? Unfortunately, historical estimates by IS personnel are not very accurate unless combinations of modern techniques such as CoCoMo and function points are used *and* few problems occur on the project. Following this logic, people who use a price-to-win strategy usually believe any estimate is good as long as they get the job, since there is little relationship between real and estimated costs anyway. Frequently, in government projects especially, the lowest bid wins the job. This logic of choosing the lowest bid leads to price-to-win estimates. This has led to problems for several government entities.

TOP-DOWN. Used with one or more of the other estimating techniques, **top-down estimates** use project properties to derive an estimate. Then total cost is split among the components. After a time estimate is derived, the 40-20-40 rule is applied to the estimate. According to the rule, 40% of project time is spent on analysis and design, 20% is spent on coding and unit testing, and 40% is spent on project testing.

The advantage of using a top-down approach is that, by focusing on global properties of the application, an estimate can be developed quickly—in a day or two. Using analogy to assess global properties, the proposed project is assumed similar to some other whole project. For instance, ABC's application is an on-line database application with create, change, delete, and query capabilities for all data, and an overall query facility for grouped data; system functions include start-up, shutdown, and monthly file maintenance processing.

The major disadvantage of a top-down approach is that the above description fits most on-line database applications (see Table 6-10). Such a high level focus cannot identify low level technical problems that drive up costs. For instance, in a complex database application, one particular data access need

5 Parkinson's Law was first published in 1957.

might require a month of design and prototyping time to prove that the concept works. This type of special process would be missed in a top-down estimate. Whole software components might be missed in the global assessment that, when developed, account for a disproportionate amount of time and cost. On balance, top-down estimates are less stable than more specific estimates.

BOTTOM-UP. The bottom-up approach takes the opposite view of an application from the top-down approach. Using a **bottom-up estimating** approach, each software component is identified and estimated, often by the person who would do the development. All individual component costs are summed to arrive at the estimated cost of the entire software product.

The bottom-up approach is as likely to miss components for development as the top-down approach (see Table 6-10). At the low level, integration work to combine modules and programs may not be estimated or is easily underestimated. Also, the bottom-up approach requires significantly more effort to develop because every module, program, screen, database interaction, and so on must be identified for estimating.

The advantages of the bottom-up approach are that the estimates are based on a more detailed understanding of the project than the other methods, and, when estimated by the person doing the work, the estimates are backed by a professional's commitment.

FUNCTION POINTS. The function point method takes an organizational history approach to estimating. **Function points** are a measure of complexity based on global application characteristics. A baseline developed by analyzing all previous applications is developed for each type application. The baseline number of function points is divided by the actual cost/time of development to get an estimate for one function point per application type (or language, or person-month). New applications are analyzed to determine an estimate of the number of function points in the project. Then, the base time and cost estimates for one function point are multiplied by the number of estimated function points for the proposed application to develop a total time and cost estimate.

Function point analysis rests on the ability of the project team to predict the inputs, outputs, queries, interfaces, and files. Figure 6-15 shows the counts and weights assigned for each type of I/O. Each item is counted and weighted for complexity. The weighted counts are summed.

Then a series of 14 questions to determine different types of application complexity are evaluated on a scale of zero to five to measure increasing importance of the item to the application (see Table 6-15). The answers to the 14 questions are also summed. The summed complexity weights and weighted counts are combined in one formula shown below to compute the total function points for a project.

Application Item	Count	Simple	Average	Complex	FP = Count * Weight
# Inputs (i.e., Trans Types)		3	4	6	
# Outputs (i.e., Reports, Screens)		4	5	7	
# Programmed Inquiries		3	4	6	
# Files / Relations		7	10	15	
# Application Interfaces		5	7	10	

From Pressman, Roger S., *Software Engineering: A Practitioner's Approach*, third edition. NY: McGraw-Hill, 1992, p. 49.

FIGURE 6-15 Function Point Weighted Count Table

TABLE 6-15 Function Point Questions and Rating Scale*

Rating Scale from 0 (No influence) to 5 (Essential)
Factor Questions:
1. Is reliable backup and recovery required?
2. Are data communications required?
3. Are any functions distributed?
4. Is performance critical?
5. Is operational environment volume high?
6. Is on-line data entry required?
7. Does on-line data entry require multiple screens or operations?
8. Is on-line files update used?
9. Are queries, screens, reports, or files complex?
10. Is processing complex?
11. Is code design for reuse?
12. Does implementation include conversion and installation?
13. Are multiple installations and/or multiple organizations involved?
14. Does application design facilitate user changes? How integral is ease of use?

*From Pressman, Roger S., *Software Engineering: A Practitioner's Approach*, third edition. NY: McGraw-Hill, 1992, p. 50.

FP = Total weighted count * (.65
+ (0.1 * Σ(complexity adjustments)))

Function points have become popular enough that several companies and software packages are available for developing function point estimates. In addition, tables of function points per number of lines of code are also available. For instance, 100 lines of Cobol is equal to 20 lines of Focus is equal to one function point. Translating function points into lines of code, then, requires a simple table lookup.

The appeal of function points is similar to that of CoCoMo. Any algorithmic method is likely to be easy to use, understand, and repeat (see Table 6-10). An algorithm gives the appearance of objectivity that other methods do not. Of course, the function point estimate has flaws similar to those of CoCoMo, too. Function points must be calibrated for the organization based on its history of project development. It assumes that history predicts the future. Further, it assumes similar technology and skills across projects. The model assumes that methodology and CASE have no impact on project development time.

To summarize, there are several useful methods of project person-month or lines-of-code estimating. The most popular are expert judgment, analogy, CoCoMo, function points, top-down, and bottom-up. All of these methods have advantages and disadvantages. If a history of projects and function points is kept, that appears to be the most accurate estimating technique at the moment. If function points are not calibrated to the company's history, no one estimating technique is better than any other. Rather, the methods might be paired or used several at a time to develop estimates that are closer to reality than estimates developed using any one method alone.

Planning Guidelines

In the absence of calibrated function points for ABC, we will discuss the use of several methods in developing a plan for an application. By combining the methods, the schedule and plan developed should be better than using any one plan on its own.

Several variations for combining estimating techniques are feasible. They are:

1. Estimate inputs, outputs, interfaces, queries, and files according to function point directions.
2. Answer 14 questions and estimate project complexity.
3. Compute function points.
4. Lookup lines of code per function point (FP) in language table and compute total lines of code (LOC) for the project.
5. Decide the CoCoMo mode.
6. Using FP LOC as input to the CoCoMo model, compute person months of effort.
7. Analyze multipliers and adjust the estimate.
8. Compute total development time and project staffing estimates using the other CoCoMo formula.

If the company uses function point analysis for its baseline, function point planning is the first type performed. Then, the plan can be compared to the CoCoMo model estimates to verify its goodness of fit. Alternatively, the project manager can develop a top-down plan while the SE and any other project staff working on the feasibility develop a bottom-up plan by using the following steps:

1. PM and SE together estimate the development approach and all functions in the application.
2. PM uses top-down analysis to develop a list of activities to be performed and the times for each.
3. From this list, deliverable products and a schedule are developed.
4. The list is analyzed to determine task dependencies, and a first-cut critical path chart is developed.
5. Concurrently with steps two to four, the SE analyzes each function bottom-up to determine the complexity, possible problems, nondeliverable programs, and amount of effort to be assigned to each technical task.
6. Any new tasks identified by either the PM or SE are added to the plan and estimated. The SE and PM compare and adjust their time estimates until they agree.

Another alternative is to combine expert judgment, analogy, top-down, and bottom-up to develop a first set of estimates. Then, these estimates are compared to the standard function point estimate for a *reality check*. If the expert estimate is more than 15% lower than the function point estimate, then the plan should probably be revised upward. In this section, we use expert judgment and analogy, using a top-down approach to develop the estimate, then do a bottom-up analysis of each piece to ensure they are all present.

The steps to developing a plan are:

1. Decide the Development Life Cycle (DLC), approach, and methodology.
2. For each phase, list the deliverable products that mark completion of the phase.
3. Decide on information gathering technique(s) and use of JAD, prototyping, or other variants to DLC.
4. Decide which products the technical project team members will develop and which the users will develop.
5. Define dependencies and develop CPM chart.
6. Assign times to tasks and compute total project time.
7. Estimate inputs, outputs, interfaces, queries, and files according to function point directions.
8. Answer 14 questions and estimate project complexity.
9. Compute function points.
10. Lookup lines of code per function point (FP) in language table and compute total lines of code (LOC) for the project.
11. Estimate productivity in LOC/month.
12. Compare FP number of person months to the estimated total time.
13. Adjust time estimates, as required, and complete the CPM diagram by adding times.

For instance, assume the waterfall is followed and the phases include Feasibility, Analysis, Design, Program Design, Code/Unit Test, System Testing, Acceptance Testing, and Installation. Then, list deliverable products. Phases might have more than one deliverable product. Products usually coincide with the ending of life cycle phases. Products for these phases include a feasibility report, functional requirements specification, design specification, program specifications, plans for testing, conversion, training, and implementation, operational documentation, and user documentation.

From the choices in Chapter 4, decide the approach to information gathering. If you use JAD, for instance, the amount of time allocated to analysis is less than if you use interviews over time. Decide the overall system design approach. Is prototyping needed? How involved will users be in the development process? How extensive will user training be? Will CASE be used? Which tool? (Some tools add analysis and design time, some reduce it). How extensive are documents expected to be? Is on-line

help software going to replace user manuals? Who is responsible for planning and executing the conversion? How much data *scrubbing* to remove errors from existing data is required? The answers to these questions increase or decrease the time allocated for each task.

Next decide which products the technical project team members will develop and which the users will develop. These tasks are estimated just as the technical team tasks are estimated, but they are also singled out for several reasons. First, the dependencies should clearly show the split of assignments for the technical team and users. Second, users should be allowed to comment on tasks for which they are responsible. The technical team usually takes responsibility for the tasks if the users will not take it.

Develop a list of tasks and define dependencies, developing a critical path chart for the project. Assign times to tasks. Compute function points. Using an estimate of LOC per month per person on the project, compute a total project time, and compare the FP estimate to your estimate. Adjust your estimate as required if it is more than 15% less than the FP estimate. In general, always use a higher estimate rather than a lower one. Project schedules have a way of losing time for meetings, nonproject responsibilities, and other legitimate, but nonproductive uses of time.

Now, let's go through each step to using combined techniques for estimating. To develop a critical path diagram, list the tasks on a sheet of paper. Begin with high level tasks, or tasks of a single phase, adding lower level tasks as they come to mind. Development of the task list requires some experience and is always done more easily by several people rather than one who is likely to forget some critical task. The task list, in critical path method terms, is called a **work breakdown.**

Define durations for each task. Durations may be an absolute number or a range of time. The critical path method recommends the identification of optimistic, likely, and pessimistic estimates. Then, the weighted formula ((Optimistic + 4(Likely) + Pessimistic) / 6) is applied to develop one number for use in financial analysis and software planning tools. Use either method for developing the time. Planning software packages allow early, most likely, and latest possible dates to be entered. For some software you enter the project completion date and the software computes the early and late dates for tasks based on their durations.

Extend the times to develop dates at which each task is expected. A work breakdown shows the earliest start and end dates for each task, plus the latest start and end dates per task. The early dates assume that each preceding task took the minimum estimated number of days. The latest start and end dates assume that each preceding task took the maximum estimated number of days.

Next, create the CPM chart (see Figure 6-16). List all tasks on a piece of paper. Draw lines from later tasks to early tasks on which they are dependent. By dependent tasks, we mean those tasks that cannot be begun until information (or products or approvals) from the previous task are complete. The early task *feeds* the later one.

When the diagram is complete, compute the time to complete each *leg* of the diagram. The leg with the longest time is the critical path, that is, the tasks on which meeting the deadline for the project depends. If any one of the critical path tasks is late, the project will be late. When monitoring the project, the critical tasks get priority. When assigning staff to tasks, the critical tasks should be assigned the most experienced and skilled personnel.

Some sensitivity analysis on critical path and on task dependencies might be done, if using an automated tool for the analysis. Manual analysis is so time-consuming that it may not be worth the effort. The impact of different end dates is analyzed. For instance, if the user were to mandate a date two months earlier than the estimated end date, what is the impact on the project and tasks? Does the critical path change? Can other tasks, not fully analyzed, be made more parallel? Can any dependencies be removed by altering the plan or tasks? If the project suffers penalties (loss of revenue) from not meeting deadlines, the risks for each task might be reassessed to ensure that nothing is missed. The project manager continues this type of analysis until he or she is comfortable with the result.

After the critical path is identified, staff should be assigned to each task to complete project planning.

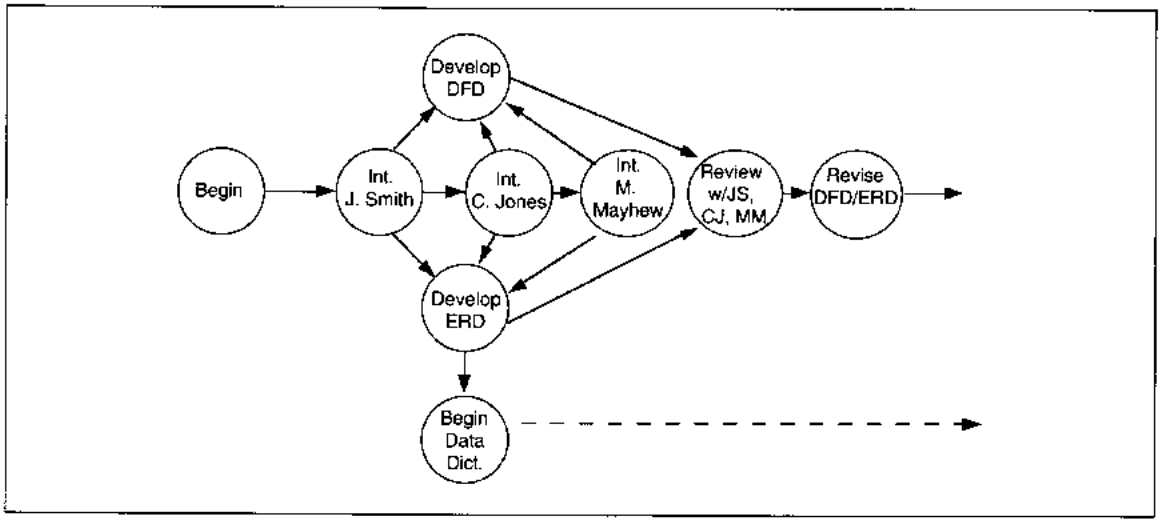


FIGURE 6-16 Sample Critical Path Method Chart

Assign people to minimize the amount of slack time for which they have no assignments, but allow some slack time in case problems arise. Assign the critical tasks first, allocating them to the best, most experienced people. A general rule of thumb is that, in absence of artificially short deadlines, people can be assigned to develop a whole *leg* of the critical path. The purpose for assigning sequential tasks in a leg are to leverage the knowledge gained from early tasks to later tasks, and to provide each individual a sense of contribution to the overall project by allowing them to take responsibility for a large *chunk* of work.

When the estimates are complete, develop a function point estimate, or have someone else do it in parallel. Weight the FP estimate by the answers to the 14 questions. Lookup the lines of code (LOC) per function point (FP) in a table (see Figure 6-17).[6] Estimate your productivity in LOC per month; for instance, 1000 LOC/Month for a 4GL is not uncommon. If your company keeps statistics, use its historical numbers for project type and language. Compute total person-months for the project using the formulae in Figure 6-18. Compare the FP estimate to your estimate and adjust as needed. Don't just blindly take the higher number. Rather, a difference means that information was interpreted dif-

6 Refer to Capers Jones' 1986 book, *Programming Productivity*, for extensive tables with this information.

Lines of Code/FP	Language
25	4 GL
25	SQL
100	Cobol

FIGURE 6-17 Example of LOC/FP for Different Languages

Number of Lines of Code per Function Point * Number of Function Points = Total Lines of Code

Example 25 LOC/FP (4GL) * 100 = 2500 LOC

Total Lines of Code / Lines of Code per Month = Number of Person Months

Example 2500 LOC / 1000 per Month = 2.5 Person Months

FIGURE 6-18 Function Point Computations for Total Person Months

ferently by the two methods of estimating. See if you can find what is different and which estimate is more realistic.

Use the 40-20-40 rule to check if the effort *looks* like it is reasonable across the phases. Analysis/design should be about 40% of effort if manual and 55% if using CASE. Code/unit test is about 20% effort if manual and 5% if a CASE tool generates code. System testing should be 30–40%. Testing estimates are usually low. If testing is the difference, ask if there is some reason to be optimistic, for instance, a skilled programmer. If the difference cannot be found, and the percentages are allocated about right, then changing your estimate is a judgment call.

For manual allocation of staff to a project, a list of tasks in CPM *legs* should be created and a person's name assigned to each task. This allows easy tracking of assignments and dates at which people rotate on and off the project. If using an automated tool, allocation of staff usually requires entry of the person's name and assignment of tasks by CPM ID. In either case, as people are assigned to tasks, note who they are and when they begin (and end) project work. Make sure you do not change the critical path by the assignment of personnel to overlapping or conflicting duration tasks.

Upon completion of task assignments, a Gantt Chart is developed to summarize the project. A **Gantt Chart** shows the entire set of project tasks, people assigned, and completion times estimated for the development effort (see Figure 6-19). A list of people and amount of time assigned to the project is created for use in the costing activity.

	Day																	
Scheduled Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Interviews																			
J. Smith		SB	SB				S												
C. Jones				SB	SB		B												
M. Mayhew					SB	SB	S												
Develop DFDs		S	-	-	-	-	S												
Define Data and ERD		B	-	-	-	-	B												
Review and Revise DFD and ERD							SB	SB											
Begin Data Dict.		SB	-	-	-	-	-	-	-	-	-	-	-	-	-	-	SB		
Define Problems w/Current System								SS											
Define Business Opportunities									BB										

Legend:

B = Barbara James, SE
S = Stan Smits, PM/SE

Where initials alternate, both Barbara and Stan participate in the activity.

FIGURE 6-19 Sample Gantt Chart

ABC Video Implementation Plan

ABC's rental application is a fairly average project with no obvious complexities, no state-of-the-art technologies, and a single, small organization. Mary, the PM, and Sam, the SE, decide to use a combination of analogy, top-down, and bottom-up and to check their estimate with function points based on the estimate of 25 LOC/FP for a 4GL. Before Mary and Sam begin, they first decide their approach and assumptions on which the estimates are based.

The project is expected to be implemented on a Novell ethernet LAN using PCs as workstations and a superserver (50 Mhz, 486-based machine). The software environment will be some SQL language with custom application software. There will be four main files, corresponding to the four main entities in the ERD. The main processing centers around rental activity with standard maintenance procedures for the other files. Other files, which will be maintained during rental processing, include history and an end-of-day summary of transactions. The application will accommodate up to ten concurrent users for all processing.

If two people are estimating, as Sam and Mary are, a good approach is to split the two types of estimates between the individuals. Sam would do one and Mary the other. Then they compare and rationalize their work.

First, we develop a function point estimate for the work. The function point estimate (see Figure 6-20) shows that the project is not very complex in any of the key inputs or outputs. The weighting questions identify the on-line, interactive, and multiuser characteristics as contributing the greatest complexity to the application. The total function points are esti-

Application Item	Count	Simple	Average	Complex	FP = Count * Weight
# Inputs (i.e., Trans Types)	5	3	(4)	6	20
# Outputs (i.e., Reports, Screens)	6	4	(5)	7	30
# Programmed Inquiries	6	(3)	4	6	18
# Files / Relations	8	7	(10)	15	80
# Application Interfaces	(0)	5	7	10	0
				Total	148

Factor Questions	Score
Reliable backup and recovery	4
Data communications	0
Distributed functions	0
Critical performance	4
High volume operations environment	4
On-line data entry	5
Multiple data entry screens or operations	5
On-line file update	4
Complex queries, screens, reports, or files	0
Complex processing	4
Reusable code design	0
Conversion and installation	4
Multiple installations and/or multiple organizations	0
User change; ease of use	3
Total	37

FP = Total Weighted Count *
(.65 + (.01 * Σ(Complexity Adjustments)))
= 148 * (.65 + (.01 * 37))
= 148 * (.65 + .37)
= 151 Function Points

FIGURE 6-20 ABC Function Point Estimate

mated at 151. Carrying the FP analysis through, at 25/LOC per function point, there are about 3775 LOC (i.e., 25 * 151) for the project. At a productivity rate of 2000 per month, the total number of person months for the project is about 1.9 months (i.e., 3775/2000). The estimate of 2,000 LOC/month is a company statistic based on the average productivity of each of the project participants.

Mary, in parallel, creates a task list which she converts into a work breakdown. The work breakdown identifies the tasks to complete the project, and the optimistic, likely, and pessimistic times for each task (see Table 6-16). The most likely time for each task is then computed and a total time for the project is estimated.

At this point, the two sets of estimates should be compared. The FP estimate suggests 1.9 person-months, while the work breakdown estimate of 172 hours translates into slightly under one month (25 days). The FP estimate is almost twice as high. Let's see where the differences might lie. At the end of Table 6-16, the total times for each phase are shown with percentages of the total computed for each number. The percentages do not follow the 40-20-40 rule closely. The realistic estimate shows 46% of time for analysis and design, 32% for coding and unit testing, and 22% for system testing. The estimate for system testing is low relative to the rule while the other estimates are somewhat inflated. Mary knows she and Sam are the only two people who are expected to work on the project and she based her estimates on their ability to debug and test quickly. But even she cannot defend this low number to Sam. Sam also points out that, if Vic wants much documentation, her estimates for all the tasks might be low. Mary has assumed that Vic, being a small company owner, will opt for less documentation to save on the expense.

On the other hand, Sam identified several complexities with which Mary takes issue, in particular with the difficulty of on-line update and the difficulty of interactive programming. Both of these were given a '5' rating of complexity. Mary feels that if the application were on a mainframe and using mainframe software and tools, the fives would be justified. Since the application platform is a LAN with which they have extensive experience, she feels that the highest rating should be a four. This would then reduce the FP estimate. Both Mary and Sam discuss their estimates, defending their reasoning processes and subjecting them to criticism by their partner. In the end, they confirm with Vic that he does want only minimal documentation, and they decide to split the difference on their estimates adding a total of 90 hours to the project. Of that time, 18 hours (20%) is allocated to code/unit test and the remaining 72 hours (80%) to testing of the project. The final estimates would then show code/unit test time of 73 hours (28% of total) and testing time of 110 hours (42% of total). While these percentages are now slightly skewed away from analysis and design, which is now 30% of the total, these percentages are in line with the 4GL need to do less analysis and design. The total estimated project time used in the financial estimates will be 262 hours or 1.5 person-months.

The final work breakdown is converted into a CPM diagram to identify the critical path of work (see Figure 6-21 for the Analysis CPM). Based on the critical path, contingencies are planned to ensure meeting of the schedule. Figure 6-22 is a Gantt chart for analysis showing how Mary and Sam split their responsibilities.

If project planning software were used, the CPM is built first, then selection of an option converts the CPM into the work breakdown. To create either diagram, the tasks and durations must be known. Sophisticated software supports the insertion of a start date for the project and, based on the optimistic and pessimistic task durations, and on the dependencies from the CPM, the software computes all the dates for the project.

Evaluate Financial Feasibility

Financial Feasibility Analysis

Financial feasibility analysis evaluates the firm's ability to pay for a project, and compares recommended alternatives to determine which is more economically attractive. In general, projects are economically feasible when the sum of all IS projects plus the proposed project is less than 10% of firm net

TABLE 6-16 ABC Work Breakdown with Durations

Task: Analysis	Optimistic	Likely	Pessimistic	(O+4L+P)/6
Define Customer Maintenance Processing	2	3	4	3
Define Video Maintenance Processing	2	3	4	3
Define Rental Process	1	2	3	2
Define Return Process	1	2	3	2
Define How Intertwined	2	3	4	3
Define History	1	2	3	2
Define EODay, Audit, Trans Log	2	3	4	3
Define Cust Create, Video Create in Rental	1	2	3	2
Define Error Msgs, Abort Procedures	1	2	3	2
Define Screen Contents	2	4	6	4
Define Flow of Processing	1	2	3	2
Define Start-up/Shutdown	1	2	3	2
Define File Purge	.5	1	1.5	1
Define Backup/Recovery	.5	1	1.5	1
Define Conversion/Training	1	2	3	2
Analysis Total Time	**19**	**34**	**49**	**34**
Task: Design	Optimistic	Likely	Pessimistic	(O+4L+P)/6
Cust Maint Process	2	3	4	3
Video Maint Process	2	3	4	3
Rent/Return Includes: Display, Data entry, Retrieval, Payment, Accounting, File Update, History, EOD, Audit, Controls	7	11	21	12
Screens	10	14	16	15
Start-up/Shutdown	4	6	12	6
Backup/Recovery	1	1	1	1
Conversion, Training	2	5	8	5
Design Total Time	**28**	**43**	**66**	**45**

income. This uses industry averages as the guideline. To compare alternatives, several methods discussed in this section are used.

Cost-benefit analysis is the comparison of the financial gains and payments that would result from selection of some alternative. The analysis facilitates comparison of alternatives for one project or alternative projects.

Criteria used in alternative comparisons might be maximizing benefits, ratio of benefits to costs, net

TABLE 6-16 ABC Work Breakdown with Durations (*Continued*)

Task: Code/Unit Test	Optimistic	Likely	Pessimistic	(O+4L+P)/6
Cust Maint Process	2	4	6	4
Video Maint Process	2	4	6	4
Rent/Return Includes: Display, Data entry, Retrieval, Payment, Accounting, File Update, History, EOD, Audit, Controls	8	14	28	15
Screens	5	10	15	10
Start-up/Shutdown	8	10	12	10
Backup/Recovery	1	2	3	2
Conversion, Training	5	10	15	10
Code/Unit Test Total Time	**31**	**54**	**85**	**55**

Task: Testing	Optimistic	Likely	Pessimistic	(O+4L+P)/6
Scaffolding	2	4	5	4
Screen test	2	4	6	4
Subsystem Test	7	14	21	15
System Test	7	14	21	15
Testing Total Time	**18**	**36**	**53**	**38**

Project Totals by Phase	Optimistic	Likely	Pessimistic	(O+4L+P)/6
Analysis Total Time	19 19%	34 20%	49 19%	44 20%
Design Total Time	28 29%	43 26%	66 26%	45 26%
Code/Unit Test Total Time	31 32%	54 32%	85 34%	55 32%
Testing Total Time	18 19%	36 22%	53 21%	38 22%
Project Total Time	96 100%	167 100%	253 100%	172 100%

benefits, minimizing costs for given level of benefit, or maximizing project internal rate of return. The most popular criterion is maximizing net benefits, which requires analysis of the present value of benefits and costs.

Three types of costs are considered: acquisition, development, and operating costs are all considered in the development of the cost-benefit analysis. Several different sources of costs relate to each of these cost types:

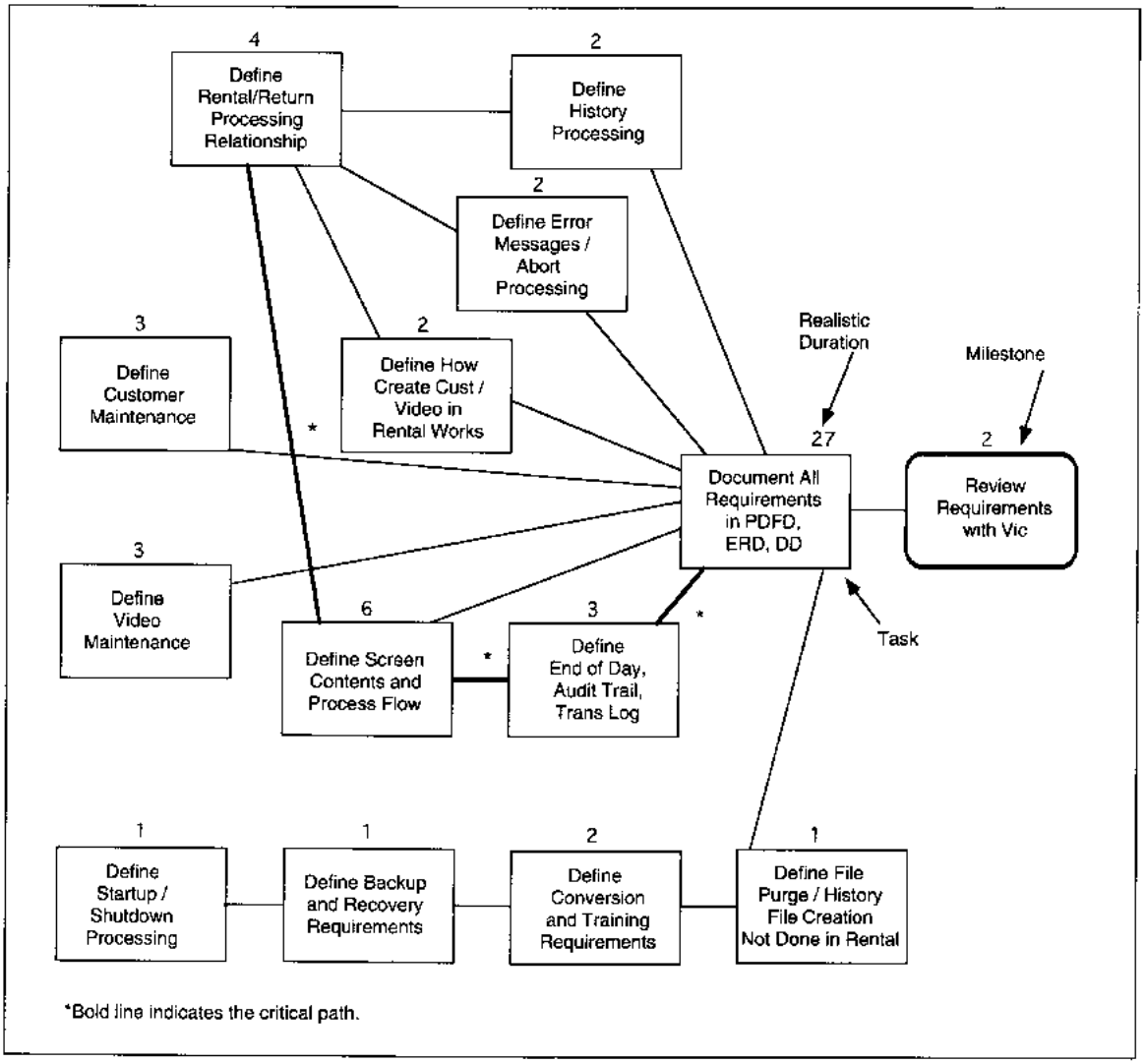


FIGURE 6-21 ABC CPM Chart for Analysis Activities

- Acquisition Costs
  - Consulting
  - Equipment
  - Software
  - Site preparation
  - Installation
  - Capital
  - Management staff assigned to acquisition
- Development Costs
  - Application development
  - Education of personnel
  - Testing
  - Conversion
  - Losses relating to changeover, downtime, reruns
  - Aggravation cost
- Operating Costs
  - Personnel allocated for maintenance
  - Hardware operating expense (e.g., air conditioning, electricity, etc.)

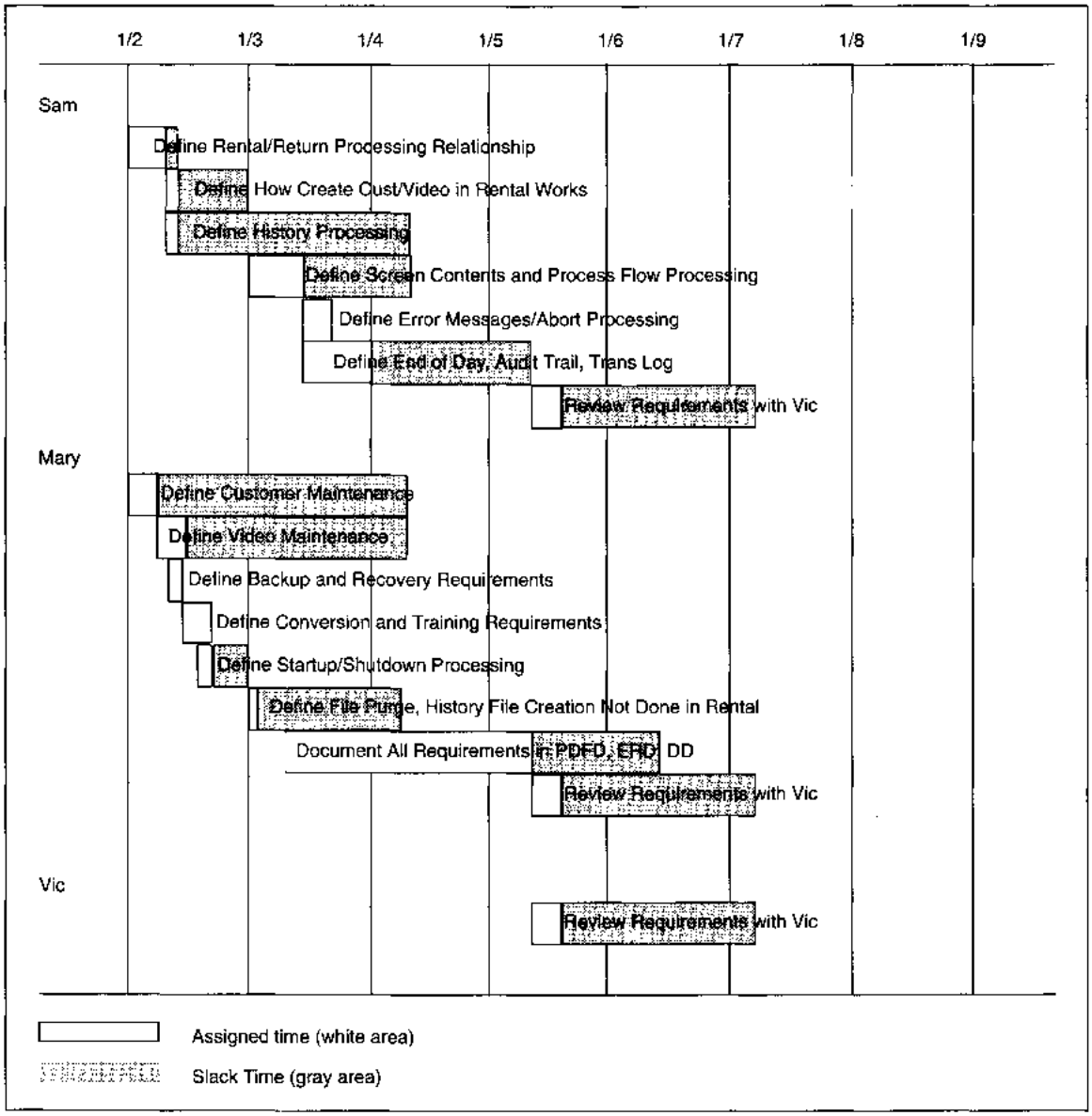


FIGURE 6-22 ABC Gantt Chart for Analysis Phase

Lease/rental costs
Depreciation on related capital acquisitions
Operating personnel overhead

In general, any time you spend money, a cost is generated. Whether the money is for salaries, personnel benefits, copy machine rental, PC acquisition, operating system acquisition, DBMS acquisition, and so forth, a cost is generated. The breakdown of costs into acquisition, development, and operating categories allows managers to do sensitivity analysis on alternatives. For instance, Alternative A might have a high acquisition cost relating to hardware site preparation and expense, whereas Alternative B has

none. If the benefits are greater with Alternative A, we might ask if the acquisition of hardware is justified by the extra benefits relating to Alternative A.

All of the costs of each alternative are assembled according to type for the analysis. Depreciation schedules, leasing schedules, and any ancillary information relating to how costs are generated over time are also used in the analysis.

Similarly, all information about benefits expected from the application are assembled for the analysis. Benefits are identified as 'one time' or as continuous improvements. If a stream of revenues is generated over time by the application, these are identified as annual revenues.

The net present value formula is applied to the benefits and costs to develop a net present value for the application (see Figure 6-23). The formula accounts for the time value of money in computing the net benefits over costs. If inflation or fluctuating interest are expected, the interest rates might be changed for each time period to account for such fluctuations. Keep in mind that exactly the same analysis is required for all competing alternatives to ensure consistent NPVs. The example shown in Figure 6-23 shows a project for which the benefits outweigh the costs; such a project would be desirable.

$NPV = \Sigma((B_t - C_t)/(1 + d)^t)$

where: t is the time period, varying from 1 to n
d is the discounted interest rate
B is the value of period benefits
C is the value of period costs

Example: $d = .08$

t	B_t	C_t	$(1 + d)^t$
1	0	50,000	1.0000
2	10,000	5,000	1.0800
3	30,000	5,000	1.1664
4	50,000	5,000	1.2597

$NPV = -(50,000/1)$
$+ 5,000/1.08$
$+ 25,000/1.1664$
$+ 45,000/1.2597$

$= -50,000 + 4,629 + 21,433 + 35,722$
$= \$11,784$

FIGURE 6-23 Net Present Value Formula and Computation

The problems arise when a project does not generate a favorable NPV, but *numbers alone do not express project value.* Benefits may be insufficient to pay for the project. For instance, in complying with government regulations, there may be no specific benefits to the company. Similarly, when responding to a competitive need, the benefits might not outweigh the costs, but the cost of not doing the project might be the loss of the business. Start-up companies frequently build applications to support anticipated work; the applications might not be profitable until they are several years old. Benefits from such applications are difficult, if not impossible, to quantify because of the uncertainty associated with a new business. Finally, companies wishing to gain significant competitive advantage must frequently undertake a financially unjustifiable project to obtain their goals. American Airlines, for instance, in developing their $1 billion airline reservation system was betting that their ability to gain market share would outweigh their expenses. The financial analysis could not justify the project because of the high level of intangible benefits and the difficulty in estimating their worth. The risk paid off, but could just as easily have backfired. That is the nature of risk and why good managers develop skill in knowing when such a risky project is worth attempting.

Make/Buy and Other Types of Analysis

Other types of analysis that might be developed are make/buy, internal rate of return, and payback period. Each of these uses NPV as a starting point for determining the value of a project. Each develops a different analysis. Make versus buy decisions evaluate two types of development alternatives. First, make/buy compares the value of a customized application to the purchase of a software package. This sounds like a simple comparison, when in fact, it is not. Purchasing software for a complex application usually requires customizing and alteration. Packages are rarely used *off-the-shelf.* Consequently,

the analysis concentrates on the extent to which changes to the package are required and the cost of purchase plus changes versus the cost of custom development.

Second, make/buy is also used to compare the competitive value of building a software product internally versus development by a consulting firm. Occasionally companies which charge for in-house IS development services begin to overcharge their users. Users are then justified in obtaining competitive bids from consulting companies and using their services when the cost is less.

Internal rate of return (IRR) is a financial analysis of NPV such that positive cash flows (i.e., benefits) are equated to negative cash flows (i.e., costs). This means that the *d*, discount rate, in the NPV formula is found. This gives the true cost of funds for this particular project. When projects have similar NPVs, an IRR analysis identifies differences in cost of money based on when the cash flows are generated that might differentiate the alternatives.

Payback period is the number of years required to recover the investment (acquisition and development) costs from projected benefit cash flows. The payback period might decrease revenues for the time value of money or might use a simple analysis of payback. Payback analysis is popular because it is easily understood. It can discriminate against projects which have a long lead time to realizing benefits, but should not be the primary criterion for project selection decisions. In the example shown in Figure 6-23, the payback period would be 3 years and 2.4 months. This number is arrived at by identifying $10,000 in year 4 as contributing to the payback along with all benefits in years 2 and 3. 10,000 is 20% of 50,000, the fourth year's projected return. Therefore, 20% of 12 months is 2.4 months. The payback, rounded, is 3 years and 3 months.

Document the Recommendations

The documentation of the feasibility study pulls together all information relevant in developing the final recommendation. The purpose of the *summary* document is to provide managers a basis for deciding whether or not to continue with the development effort. With this thought in mind, the feasibility document should contain mainly supporting diagrams, lists, and summary analyses. Text should be kept to a minimum to explain the attached diagrams and analyses. An outline of a feasibility document is provided in Table 6-17.

TABLE 6-17 Feasibility Report Outline

1.0 Management Summary

2.0 Current Environment
- 2.1 Business Environment
- 2.2 Work Procedures
- 2.3 Evaluation of Strengths and Weaknesses of Current Procedures

3.0 Proposed Solution
- 3.1 Scope of Proposed Solution
- 3.2 Functional Requirements Overview

4.0 Technical Alternatives
- 4.1 Alternative 1
  - 4.1.1 Description of Alternative
  - 4.1.2 Benefits of Alternative
  - 4.1.3 Risks of Alternative
- 4.2 Alternative 2 . . .
- 4.n Alternative n

5.0 Recommended Technical Solution
- 5.1 Comparison of Alternatives
  - 5.1.1 Technical Comparison
  - 5.1.2 Benefits Comparison
  - 5.1.3 Risk Comparison
  - 5.1.4 Recommendation and Risk Contingency Plan

6.0 Project Plan
- 6.1 Critical Path Chart
- 6.2 Staffing Plan

7.0 Costs
- 7.1 Cost of Recommended Alternative Hardware/Software
- 7.2 Projected Staffing Cost
- 7.3 Analysis of Alternatives (if necessary)

The Management Summary section is the most important because it is the only item read by most of the audience. Therefore, it should be brief, less than two pages, and should summarize the remainder of the document. In particular, the cost, NPV, other financial analyses, scope, purpose, technical recommendation, and importance of the project to the organization are highlighted in the summary section. All organizations involved in the development effort and the nature of their involvement should be highlighted.

The remaining sections summarize each of the main activities completed during the feasibility study. The current environment and proposed alternatives are described in sufficient detail to give the reader an understanding of the differences proposed. This section identifies hardware, operating environment, software, items for custom development, and requirements met by the alternative. Benefits and risks associated with each alternative are also listed and discussed to trace the reasoning leading to a selection.

The section on the recommended technical solution is more detailed than the alternatives discussion and discusses different topics. The tasks, key features, and development life cycle, methodology, and concept are highlighted in the proposed application section. In addition, the discussion lists constraints, assumptions, level of security, recovery, and auditability for the recommended solution. A contingency plan for minimizing the probability and for dealing with risks of the recommended alternative are detailed. Potential impediments to successful development, such as decisions or information not currently available, are identified. Ideally, the person responsible for resolving the outstanding issues is named and dates for resolution are identified.

The project plan section summarizes the planning effort. A critical path chart and staffing plan are presented with any attendant assumptions and requirements. Finally, the costs of the recommended alternative(s) and the financial analysis are detailed. Any assumptions, for instance, the discount rate for NPV, are listed. If sensitivity analysis was performed, the extent to which the estimates are sensitive and the source of sensitivity are identified. Sources of sensitivity might include, for instance, interest rates, economic fluctuation, or the presence of a key salesperson.

AUTOMATED SUPPORT TOOLS FOR FEASIBILITY ANALYSIS

There are two classes of tools that support the work performed during feasibility analysis: planning tools and analysis tools. Analysis tools can span any of the three methodologies covered in this text and are discussed in the respective methodology analysis section.

The planning tools might include project estimating products, project scheduling products, risk analysis products, or spreadsheets for financial analysis. Spreadsheets are general purpose and are not discussed here. Estimating products are based on an algorithmic method from those discussed above. Products based on CoCoMo estimating, Rayleigh curve, and function point techniques are included in the list. The tools assume that the underlying input information, for instance, KLOC, is known by some other, unspecified technique.

Planning products assume that a work breakdown with task duration assignment exists. The work breakdown planning tools support the definition of tasks, task interrelationships, assignment of staff, determination of early and late start dates, expected end dates, and cost of resources. From this information, the tool can generate Gantt Charts, critical path networks, cost summaries, and manpower planning guides. There are many good project management software products of both types on the market, several of which are listed in Table 6-18.

Two risk analysis products are included in the summary list. These products walk you through the assignment of risk types, probability of risk occurrence, and cost of the risk to develop a monetary value of risks related to the project. The cost of risk is factored into the financial analysis. More products of this type should be expected to be available as companies become more sophisticated in

TABLE 6-18 Automated Tools to Support Project Planning

Product	Company	Technique
DEC Plan	Digital Equipment Corp. Maynard, MA	CoCoMo Based Estimation Tool
ESTIMACS	Computer Associates, Inc. Long Island, New York	Function Point estimates extrapolated to include staffing, cost, risk, hardware configuration, and cost estimating
Harvard Project Manager	Harvard Graphics Corp. Boston, MA	Pert, CPM, and Gantt Charts Resource Allocation and Tracking
MacProject	Apple Computer Cupertino, CA	Pert, CPM, and Gantt Charts Resource Allocation and Tracking
ProMap V	LOG/AN, Inc.	Risk Analysis
RISNET	J. M. Cockerman Associates	Risk Analysis
SLIM	Quantitative Software Management	Costing Software based on Rayleigh curve and LOC
SPQR/20	Software Productivity Research, Inc. Cambridge, MA	Multiple choice approach to function point estimation
Time Line	Symantec Software Cupertino, CA	Pert, CPM, and Gantt Charts Resource Allocation and Tracking
WINGS	AGS, Inc. New York, NY	Pert, CPM, and Gantt Charts Resource Allocation and Tracking

their assessment of the risk associated with capital projects.

SUMMARY

Feasibility analysis is an important activity that gives a development project a scope and refined definition of application purpose, while providing information that allows the determination of technical, organizational, and financial readiness of the organization. The steps to performing feasibility analysis are: collect data, define scope and functions, define technical alternatives, define benefits and risks of each alternative, analyze organizational and technical feasibility, select technical alternative(s), define project plan, assess financial feasibility, and select final alternative. Data collection most frequently uses interviews or JAD-like sessions to define current work environment, problems, and desires for the new application. From the information collected, the team and user define the scope of the activity, including all departments involved. Then, the functions to be kept from the current work environment

and functions to be added to provide the new functionality are defined at a high level.

Technical alternative definition begins with an assessment of the project's criticality to the organization and the need for different departments to share data. Based on that information, existing computer resources are analyzed to determine their usefulness for the proposed application. If existing resources are not adequate, new computer equipment, software, or packages are defined for acquisition. In general, the smallest size computer (or LAN) that can do the work and provide a migration path for growth is selected. Distributed resources might be identified as an option but are not fully analyzed at this time. Several technical alternatives are developed and analyzed to select one or two that meet the most requirements, provide for the greatest benefits, and pose the fewest risks.

The next activity is to define a project plan. There are many different estimating techniques for projecting time to complete a project: algorithmic, top-down, bottom-up, price-to-win, Parkinson's Law, expert judgment, function point analysis, and analogy. Of these, CoCoMo and function point are the most popular when a history of project development is maintained by a company. Function point analysis complements CoCoMo in developing an estimate of LOC. CoCoMo can use the LOC estimate as input to its formulae to develop total person-month, total development time, and project staffing estimates. Parkinson and price-to-win are *not* recommended. When other techniques are used, they are best used in combination. So, top-down, bottom-up, and expert judgment might be combined to develop best guesses of the time and effort involved in a development project. The project plan is used to develop personnel costs and computer resource usage. These and the other costs are factored into the financial feasibility assessment.

Financial feasibility techniques most commonly used include net present value analysis which accounts for the time value of money, internal rate of return which identifies the real interest rate of a project, and payback analysis which identifies the time at which net revenues equals net costs of project. Financial analysis also supports the comparison of make versus buy alternatives for a project. Two types of make/buy analysis can be developed. First, custom development of software versus purchase of a package can be evaluated. Second, in-house versus contractor development can be evaluated. Finally, alternative selection is based on financial value of the alternative(s) when more than one technical alternative for a project exists. Also, from the financial analysis, managers can evaluate several different projects using an objective method and can identify the project with the fastest, strongest returns.

REFERENCES

Albrecht, Albert J., and James E. Gaffney, "Software function, source lines of code and development effort prediction: A software science validation," *IEEE Transactions on Software Engineering*, November, 1983, pp. 639–648.

Boehm, Barry W., *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

Charette, Robert N., *Software Engineering Risk Analysis and Management*. NY: McGraw-Hill, 1989.

Collins, Eliza G. C., and Mary Anne Devanna, eds., *The Portable MBA*. NY: John Wiley & Sons, 1990.

De Marco, Tom, "An algorithm for sizing software products," *Performance Evaluation Review*, ACM SIGMetrics Publication, Vol. 12, #2, Spring–Summer, 1984, pp. 13–22.

Gause, Donald C., and Gerald M. Weinberg, *Exploring Requirements Quality Before Design*. NY: Dorset House Publishing, 1989.

Jones, Capers, "Program Quality and Programmer Productivity: A Survey of the State of the Art," Presentation through Software Productivity Research, Inc., Boston, MA: March 15, 1989.

Jones, Capers, *Programming Productivity*. NY: McGraw-Hill, 1986.

Kendall, Ken E., and Julie E. Kendall, *Systems Analysis and Design*, 2nd Ed. Englewood Cliffs, NJ: Prentice-Hall, Inc. 1992.

King, John L., and Edward L. Schrems, "Cost-benefit analysis in information systems development and operation," *Computing Surveys*, Vol. 10, #1, March, 1978, pp. 20–34.

Rubin, Martin S., *Documentation Standards and Procedures for On-line Systems*. NY: Van Nostrand Reinhold Company, 1979.

KEY TERMS

algorithmic estimating
alternative approaches
analogical estimating
application leverage point
benefit
bottom up estimating
business leverage point
CoCoMo estimating
competitive environment
contingency planning
cost/benefit analysis
critical success factor
customer environment
delivered source instructions
Delphi method of estimating
discounted cash flow
embedded project
expert judgment estimating
feasibility
financial feasibility
flexibility
function point
function point analysis
Gantt Chart
goal
imaging
industry environment
intangible benefits
internal rate of return analysis
KDSI
leverage point
make/buy analysis
net present value (NPV)
objective
organic project
organizational feasibility
Parkinson's Law
payback period analysis
pert chart
platform
portability
price to win strategy
project mode
project plan
quick analysis
reliability
risk
risk assessment
semidetached project
tangible benefits
technical feasibility
top down estimating
vendor environment
work breakdown
work flow management

EXERCISES

1. Using Table 6-6 as a guide, develop a CPM for the design phase of ABC's project. While you do the diagram, reason through the dependencies. Assuming Sam and Mary do the project alone, how should the work be allocated between them to (a) allow Mary to do project management tasks, and (b) leverage the work they did during analysis?
2. Using Table 6-6 as a guide, develop a more detailed task list for some phase or portion of a phase (e.g., all rental/return processes, or conversion/training). Then, develop an estimate of the work based on your expertise and the idea that you would perform the work. How does your estimate differ from Table 6-6? Why? Are the differences completely justifiable? Present your estimates to a group of classmates and provide your reasoning for the changes.

STUDY QUESTIONS

1. Define the following terms:

Benefits	Function point
Net present value	Leverage point
Risk	Technical feasibility

2. Why is feasibility analysis performed?
3. What are the three main types of feasibility and why are they important?
4. List the steps to performing feasibility analysis.
5. What are the main data collection techniques used during feasibility analysis?
6. What is a leverage point?
7. How do business and application leverage points differ? How do they complement each other?
8. List five sources of benefits.
9. Discuss the differences between tangible and intangible benefits.
10. List five sources of risk and give an example of each.
11. Why is risk analysis performed? What do you do with the risks once they are identified?
12. How are technical alternatives generated?
13. Once technical alternatives are complete, how are they assessed? What is the basis for selecting one alternative as the preferred one?
14. Compare the advantages and disadvantages of algorithmic, function point, and combined top-down, bottom-up estimating.
15. What is the major weakness of CoCoMo estimating?
16. What is the major weakness of function point estimating?
17. Why do we have so many estimating techniques? Is one better than another?
18. What is the major financial analysis used to analyze project alternatives? Why is it the preferred method?

19. What is the purpose of make/buy analysis?
20. Describe the two types of make/buy analysis.

★ EXTRA-CREDIT QUESTION

1. The Office Information System described in the Appendix is an application that automates the support division of a large company. The units involved include a typing pool, copy center, print shop, and graphic arts department. Other projects are being developed in the IS Department that will cost approximately $2.4 million per year, and an additional $1.5 million in operating expenses.

   The proposed budget for the OIS is $200,000 for a Cobol, mainframe application using a DBMS to store the data. Is this a reasonable amount? Develop one to three alternatives that are more financially attractive. One of the alternatives might be on the mainframe but can use different resources; at least one alternative should use different technology. Who should develop the application? Under what circumstances would you recommend to do/not do the application?

PART III

ANALYSIS AND DESIGN

INTRODUCTION

Analysis is the act of defining *what* an application will do. **Design** is the act of defining *how* the requirements defined during analysis will be implemented in a specific hardware/software environment. The next eight chapters define and describe functional analysis and design. Each set of analysis-design chapters uses major representation techniques from the methodology class it presents. In a traditional application development, there are many more analysis and design activities than we address here (see Tables III-1 and III-2). Most of these topics should already be part of your knowledge base from a systems analysis and design course. Many activities we *do* cover in this text are also in a systems analysis and design course. The difference is that here we develop three methodologies instead of one as in systems analysis. In this text, we concentrate on the activities which differ across the methodologies. Chapter 13 summarizes the similarities, differences, and automated support across the methodologies. It also discusses the future based on current research in methodologies. Chapter 14 discusses the forgotten activities in most methodology-related books and many systems analysis texts. These activities include human interface design, input/output design, conversion design, and user documentation design.

At the end of the next eight chapters, you should be able to do the following:

1. Understand the conceptual foundations of the three classes of methodologies and how they are similar and how they differ.
2. Represent the functional requirements of an application using each of the three methodologies.
3. Be able to translate a functional requirements definition into a SQL-based design for an application using each of the three methodologies.
4. Compare the advantages, values and disadvantages of methodologies' uses for analysis.
5. Develop a critical understanding of the difficulties of translating what users want into representations that convey meaning.
6. Know some computer-aided and organizational supports for completing analysis and design work.

TABLE III-1 Representative Project Development Analysis Activities

Recurring activities/tasks
- Initiate phase
- Plan next phase
- Prepare report
- Review phase products

Analysis Phase Activities

Initiate hardware/software evaluation (as required)
Initiate prototype development (as required)

Define current system (as required)
- Document and files
- Data elements
- Compile data dictionary
- Processing
- Controls
- Volumes and timing
- Interfaces with other systems
- Responsibilities
- Work distribution
- Operating costs

Assess current system
- Review project objectives and scope
- Compare system in operation with recommended solution
- Identify opportunities for immediate improvements
- Assess organizational design appropriateness for application

Define proposed application's business requirements
- System concept and overview
  - Major functions
  - Scope
  - User organizations involved
  - Interface organizations
  - Interface application systems
  - Context diagram
  - System concept—technology (i.e., DBMS, LAN, distribution plan, etc.)
  - Major issues, unresolved problems that might hinder application development
  - Schedule summary by phase
  - Staffing summary by phase
- Assess proposed system requirements
  - Identify alternatives for system design [e.g., database environment(s), hardware platform, software platform, special technology, packaged software, 4GLs, user software (e.g., Lotus)]
  - Discuss and, as necessary, reassess technical, organizational, and economic feasibility as each relates to the alternatives identified

Define processing requirements
- DFD (or analogous graphic for the methodology)
- Steps (i.e., procedures to be followed; should match methodology)
- Required sequences of processing only
- Constraints (e.g., timing, memory, concurrency, other applications, etc.)
- Accuracy (e.g., to *x* decimal place, or timing as of *y* minutes)
- Formulae
- Performance criteria (e.g., volume, timing, response time)
- Inputs—name, source, frequency, volume, data elements, media
- Outputs—name, purpose, frequency, screen format, copies, elements, sequence, media
- Database—data requirements as expressed in methodology, relations, user views, organization, required reviews, access, security
- Reports—name, purpose, destination, frequency, form/screen, data elements, sequence
- User acceptance criteria

Define interface requirements
- Identification—name of interface, sending system/organization, receiving system/organization
- Responsibility/approvals
- Interface schedule—testing schedule and responsibilities, conversion schedule and responsibilities, delivery to production
- Requirements
  - Inputs—name, purpose source, frequency, media, form #, components using each input, data elements, data controls, data descriptions, formulae for computation
    - Input layout—data direction, terminal devices, comm software, time outs, modem requirements, line use, data characteristics, line characteristics, line protocol
  - Output—name, purpose, frequency, format/screen #, copies, elements, sort sequence, media, component generating the output, source of data and name, data description, layout (transmitted output should have same information as input layout above)

TABLE III-1 Representative Project Development Analysis Activities (*Continued*)

- Files—system name, system ID, file name, file ID, type of file (I/O), purpose, source, update cycle, sequence, frequency, volume, growth, media, usage (R, W, R/W), retention characteristics, security, blocking factor, file records types, components using file, file control characteristics
- Record description—record name, file ID, record type (fixed, variable, spanned), record size, update cycle, form # for input, data elements and characteristics (definition, purpose, use in computation, formulae, precision, edit criteria, defaults, required/optional data, etc.)

Define control requirements
- Batch totals, item counts
- Hash totals, record counts
- Operation intervention and inquiry logs
- Exception reporting and responsibilities
- Processing controls—equipment failure
- Document control (e.g., for prenumbered checks)
- Transaction logging and on-line controls

Define security and backup requirements
- Recovery requirements data criticality, recovery plan in event of emergency
- Password and internal security checks

Define conversion requirements
- Data clean-up
- Clerical effort
- Systems effort—automated and manual files to be converted
- Volume and growth of files as it impacts conversion
- Alternatives for implementation
- Overall conversion timing requirements
- Conversion impact on user areas
- Conversion impact on operations
- Facilities alteration/site preparation
  - Changes or additions to desks, tables, work spaces, cabinets, charts, etc.
  - Forms, tapes, manuals, etc.
  - Construction—walls, floors, ducts, etc.
  - Cabling and electrical—outlets, switches, cables, lighting, other wiring, etc.
  - Safety—extinguishers, alarms, first aid kits, etc.
  - Security—badge-entry, guard service, etc.
  - Environmental—air conditioning, humidification, dust, etc.
  - Maintenance—cleaning, equipment maintenance, etc.
  - Contingency—disaster plans, backup procedures, etc.

Define training
- Type of training, recipients, and details for all training, including but not limited to on-line data entry, remote location data input, native language manuals, general introduction to new system

Define system acceptance criteria
- Test data input by user
- Parallel runs
- Pilot runs
- Phased cutover
- Depending on acceptance criteria, include the following:
  - Amount of test data to be entered, and number of clerks involved
  - Size of pilot parallel (e.g., number of accounts, cycles, etc.)
  - Length of time
  - Performance criteria
  - Impact on clerical staff
  - Impact on operations

Define hardware
- Acceptable limits of downtime
- Average or maximum terminals down at the same time
- Inquiry response time
- Update response time
- Batch turnaround time
- Maximum percent of transmission errors
- Backup 'firedrills' plan and frequency
- Maintenance/reliability
- Peak and average time requirements
- Geographic constraints on terminal location
- Purchased hardware required cost/benefit analysis and RFP selection process
- List of hardware for this system, type, location, 'ownership,' system role, backup, criticality (This list should include terminals, PCs, controllers, modems, transmission lines, mini-computers, workstations, mainframes, peripherals, disks, CDs, tapes, etc.)

Define software/system/misc.
- Volume of each transaction type
- Growth
- Delivery time constraints

(*Continued on next page*)

TABLE III-1 Representative Project Development Analysis Activities *(Continued)*

- Number of reruns
- Backup 'firedrills' plan and frequency
- Distribution of output messages
- List of hardware for this system, type, location, 'ownership,' system on which it runs, backup, criticality [This list should include DBMS, operating system, LAN, communications, remote access (e.g., Carbon Copy), on-line help, etc.]

Define initiate request for proposal (RFP)
- Determine criteria for decision
- List requirements for proposal
- Select vendors
- Prepare RFP report

Define data requirements
- Data dictionary should be an appendix to documentation if it is not automated. For automated application documentation, print the information from the dictionary. For manual applications, include the following for each data element:
  - Field name, alternative name, description, purpose, use in computation, use in determining conditions (with other fields), code reference, length, decimal positions, type, unit of measure, optional/required, allowable values (range, code structure, meaning of values), default value, external data source

In this section, we introduce the general characteristics of analysis and design that all methodologies have in common.

APPLICATION DEVELOPMENT AS A TRANSLATION ACTIVITY

The process of building applications is a series of translations. Historically, we first examine and translate the current physical system to develop an abstract, logical definition of the current system (see Figure III-1). Then, with the application users, we define the requirements of the new logical system which retains the aspects of the old system while incorporating the new requirements defined by users. The new logical system definition is the basis for translating to a working physical application.

This historical strategy is useful only sometimes. The strategy works when a new application will maintain 50% or more of the old application's functions. For example, we might redevelop an accounting application to move from batch to on-line, but to perform all the same functions. Another use of this strategy is when study of the old application can save time in providing code tables. For instance, state abbreviations, zip codes, and customer name abbreviations all might be retained from an old application.

In many situations, however, the existing application is antiquated, full of obsolete design or riddled with errors. To study it is to learn erroneous design and procedures that must be unlearned. Why learn it in the first place? Rather, a frequently better approach is to begin analyzing the requirements of the new application. This is called 'essential' system analysis[1] and requires only that you, the analyst, attend to what relates to the new application. The old application or procedures may be studied for specific information, code tables, or crucial steps in the process; but in general, the old application and procedures are ignored.

The essential approach is used in this text. We ignore the details of the manual method of performing rental processing because the computerized method will completely replace the manual method. The major value of studying, for instance, what manual forms are filed and when they get retrieved, is to help get a sense of file processing in the new application. When the old procedures are being replaced, you may want to use the old methods as a way to confirm your thinking *after* you have developed the application concepts.

Whichever analysis method you use, translations performed during analysis all have the following five

1 See McMenamin and Palmer, 1984.

TABLE III-2 Representative Design Phase Activities

Recurring activities/tasks
- Initiate phase
- Plan next phase
- Prepare documentation
- Review phase products

Design Phase Activities

Initiate business system design

Design functional outline
- Review business functions
- Review interface requirements
- Develop alternative functional outlines
- Select best alternative
- Design data structure/database
  - Normalize, optimize, then . . . denormalize as required
- Design interprogram flows and controls

Design input, output, and data
- Design output screens/documents
- Design input requirements/screens
- Design screen dialogue and system navigation

Design processing
- Design computer processing
- Design noncomputer processing

Design controls
- Describe business control procedures
- Define security and backup procedures

Design business system test plan
- Identify acceptance criteria
- Prepare tentative user acceptance strategy
- Identify critical resource requirements
- Prepare testing overview
- Develop system test plan

Complete business system design
- Complete data dictionary with elements, processes, messages, objects, modules, files/relations, data flows
- Define proposed organization
- Review conversion requirements
- Prepare operating schedule
- Perform program design as outlined below

Evaluate business system design
- Assure technical, operational, and economic feasibility
- Review risks

User procedures
Define manual procedures
- Define user manual procedures
- Define computer operations manual procedures
- Prepare manual procedures test plan

Complete forms, documents, and screens
- Prototype forms, screens, reports
- Complete input documents/screens
- Complete output forms/screens
- Complete screen designs, error codes, screen interaction process

Develop training
- Determine pedagogical training requirements
- Determine training methods
- Prepare training sessions and software
- Prepare training schedule
- Pilot test training

Prepare for installation
- Prepare and test user manual
- Verify readiness of user environment
- Train user personnel
- Test manual, backup, and disaster procedures

Design the physical database
- Define user views
- Define logical DB to DBMS
- Map logical DB to media, deciding specific access method, extra space allocation, algorithms, etc.
- Build and test a sample DB
- Work with test planners to build the test DB environment
- Work with conversion team to implement the production DB environment

Build conversion subsystem
- Work with user to translate and validate current data
- Specify, write, and test conversion programs
- Train conversion personnel
- Execute conversion plan to build permanent DB

Program design
Develop modular program structure
- Study data structure
- Develop logical program structure
  - Complete methodology-related graphics
  - Specify subprograms, modules, functions
- Document programs/modules individually and as a collection. Pay special attention to document intermodular relationships and message passing between programs

(*Continued on next page*)

TABLE III-2 Representative Design Phase Activities (*Continued*)

- Develop and unit test physical code
  - Implement programs top-down using stubs
  - Prototype as needed
- Plan program testing
  - Prepare program test plan
  - Create program test data
  - Create test dialog for single user, multiple users, multiple functions
- Similar plans for subsystem, system, stress, multiuser, and acceptance testing are required and planned at this point. (If the application is on a tight deadline, testing and immediate conversion to production can be planned and implemented together.)
- Define program development plan
  - Determine development method
  - Define development sequence
  - Revise schedule and budget for programming phase
- Create source library members
  - Write record descriptions for source library (This is not done if an active dictionary is used or if the dictionary for the DBMS monitors all interactions to the database. Instead, copy books or analogous code are included to describe user views.)
  - Write standard program code to source library
- Refine operational requirements
  - Revise computer run procedures
  - Produce tentative production control cards (JCL)

common subactivities (the activities are summarized in Table III-3).

1. **Identification**—Find the focal *things* that belong. Identification, for instance, in the definition of the new logical system requires finding requirements. *Things* to be found include, for instance, entities, objects, relationships, functions, processes, and constraints.
2. **Elaboration**—Define the details of each *thing* identified. For instance, a requirement might be to provide consolidated customer account information for ad hoc reporting. During elaboration, you seek to answer questions like:

   What information should be consolidated about a user? Does it currently exist?
   What does ad hoc mean to the user?
   What type of queries do the users ask now?
   What types of questions do the users *want* to ask that they cannot ask now?
   What kinds of data analysis do users need?
   What form (for example, screen or paper) does output take?
   Where (geographically) are the users asking the questions?
   Where (centralized/distributed/decentralized) is the data? and where should it be?

3. **Synthesis**—Build a unified view of the application, reconciling any parts that do not fit and representing requirements in graphic form. The representation can be either manual (i.e., on paper) or automated, using computer-based tools.
4. **Review**—Perform quality control. At the end of the phase (either analysis or design), reanalyze feasibility, schedules, and staffing. Revise them as needed based on the more complete, current definition of the new application.
5. **Document**—Create useful documents from graphics and supporting text either manually or with computer-based tools.

Each of the three methodologies begins analysis by defining requirements, but each has a different starting (and ending) perspective for its analysis process. Similarly, for each of the other analysis activities, the results of the activity differ because the perspective at the start focuses your attention to different aspects of the application.

Keep in mind that even though we discuss these methodologies as fairly linear, sequential processes,

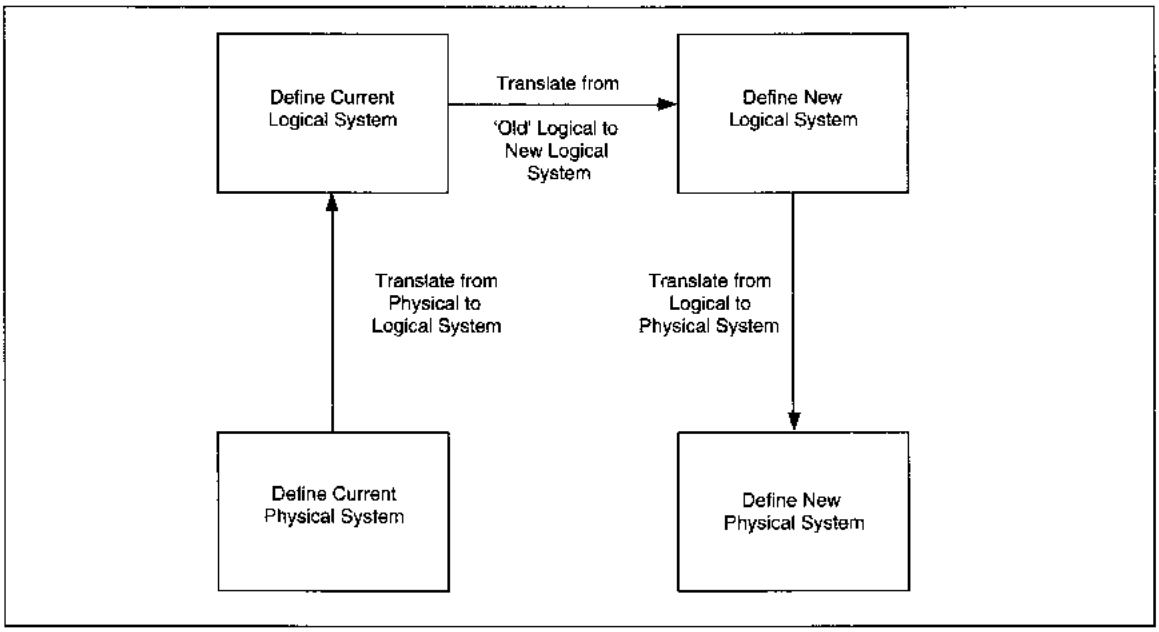


FIGURE III-1 Application Development Translations

they are not. You get application requirements in a nonlinear fashion, usually through interviews. Frequently, you get high-, low-, and medium-level information all at the same interview. Your job, as the SE, is to make sense of the information received. The sense-making activity is part of the process of building your mental model of the application domain. Since you receive information at different levels over time, your mental model of the domain gets fleshed out at different levels over time, too. You constantly have to reevaluate the information you currently have against new information to determine if adjustments to the current mental model are necessary.

A second point about the nonlinear aspect is that specification and implementation are never *really* separated completely in your thinking process. In systems analysis class you usually learn not to think about the language or implementation environment while you are performing analysis. You are told only to think about functional requirements. You must think of the implementation environment periodically in the real-world, however, because some desired function might not be able to be done (or done easily) in the planned environment. When expensive or complex functions are requested, you must alert the user/sponsor to be sure they agree with the desired function. An expensive change is one that adds more than 10% to the cost of the application. A complex change is one that convolutes an otherwise simple process (see Example III-1).

Just as analysis is a translation activity, so too, is design. The goal of design is to map the functional requirements from analysis into a specific hardware and software environment. In design, the same five general subactivities are done, but they have different definitions.

1. **Identification**—Design is the act of mapping *how* logical requirements will work in the target computer environment. This means that we identify the system design structure (if not already decided). The system structure is the underlying design approach. Possible approaches include the following:
   - Batch, on-line (portions of complete), or real-time
   - Which functions are connected and how . . . how the application will work in the production environment

TABLE III-3 Summary of Analysis and Design General Activities

Activity	Analysis	Design
Identification	Find the focal *things* that are in the application. This includes, but is not limited to, entities, objects, relationships, functions, constraints, data elements, control, legal requirements, etc.	Refine the system concept and apply it to the functional requirements. Identify any compromises of requirements that might be necessary to work around implementation environment limitations. Define the general standards and rules for the implementation environment to which all remaining work must adhere.
Elaboration	Define the functional details of each *thing* identified. Users provide definitions for all terms and describe all procedures, formulae, and processing. This elaboration is independent of hardware, software, or location.	For each function, map the function to the hardware and software environment. Identify reusable modules. Finalize details of message processing and intermodule communications.
Synthesis	Develop a unified view of the application. Develop and document a representation of the application. Graphics, tables, and other techniques are preferred representations.	Develop a unified mapping of the application to the intended hardware and software environment. Determine geographic and package locations for all data and processes. Graphics, tables, and other techniques are preferred representations.
Review	Review and walk-through the analysis with peers and project members. Walk-through the analysis with users. Review and revise schedules and costs as necessary.	Review and walk-through design components, test plan, conversion plan, and DB design, with peers and project members, program specifications with the programmer and other peers, and screenswith users. Review and revise schedules and costs as necessary.
Document	Develop 'final' forms of graphics and supporting text for *all* analysis activities.	Develop 'final' forms of graphics and supporting text for *all* design activities.

- General user interface as menu-driven, windows-icons-menus-pointers (WIMP), command-driven
- Mode of operation, that is, is user an expert, novice, or somewhere in between

2. **Elaboration**—Each requirement from the analysis phase is expanded into greater detail and mapped to hardware and software within the system design structure. Questions relate to:

   How should the database be designed to provide, for instance, the best possible response time with the greatest efficiency?

EXAMPLE III-1

CARTER CORDUROY—YOU SHOULD HAVE ASKED AGAIN

Carter Corduroy, a $100 million company, wanted to install an integrated database application to perform order entry, inventory control, and manufacturing control. During the analysis of the application, George Dare was the user contact who approved all requirements, acted as liaison to the rest of the company, and provided many requirements.

The analysis phase of the project completed on time and all ten project team members felt they had a good understanding of the process required and what the resulting application would do. The two people preparing most of the documentation and all of the program specifications were Maria Martinez, SE/project manager for 10 years who had done two other such integrated order-inventory systems, and Charlie Chou, SE with 12 years of experience who had developed applications using all of the software involved.

During the middle of the analysis phase, the systems manager was replaced with a newly hired person, Robert Blake. Mr. Blake came from a larger fabric manufacturer and wanted to make a name for himself quickly in his new environment. He quickly forged a liaison with Harry Crater, the plants' manager. The application would be installed in his two finishing plants: one in Virginia and one in Arkansas.

Crater and Dare were political enemies. Dare had once worked for Crater and had not gotten along with him. Dare was young and highly proficient at his job and soon surpassed Crater. Crater now reported to Dare for purposes of developing the application—the biggest in the company's history.

These circumstances did not affect the application team until late in design, after programming had begun. Six weeks before the application was supposed to go into production, Dare was on vacation. Crater had a validation meeting for reporting requirements with Martinez and Chou. At the meeting, he said that planned reports could not identify 'reworks,' goods that were defective and reentered into the finishing process a second time. He was adamant that he must have some way of knowing if a lot of goods were a 'first work' or a 'second work.' It was the first mention of anything other than one-time-through manufacturing. Maria said this constituted a major change to the requirements and a nontrivial change to all programs already begun. It was so significant that the end date of the project was in jeopardy. She decided to examine the specific impact, then talk to George about the change.

Mr. Blake heard of the meeting and, that afternoon, began pressuring Maria and Charlie to 'do what Crater wants.' After all, he was the real user.

Maria talked to the team and asked for an assessment of effort to change their programs to allow the same lot order to be processed more than once. She and Charlie then did their own assessment. The team was unanimous. The change would add four to six weeks time for programming and testing, all documentation would have to be modified, and all databases would be changed. In short, the change could add as much as $90,000 to the $225,000 contract—a 40% increase. Maria decided to speak to George before committing to the change.

Mr. Blake coerced the team, as their immediate boss no matter who the user of the application was, to begin work on the change. When George got back, he was immersed in another special project that was

(*Continued on next page*)

EXAMPLE III-1

CARTER CORDUROY—YOU SHOULD HAVE ASKED AGAIN, *Continued*

taking most of his time. When Maria finally got to him, he said, "Yeah, if Blake approves and Crater insists, we probably need it." Still, Maria had doubts.

She put the changes with cost estimates in a memo to Blake. He never signed-off on the change, but verbally agreed again. The application was three weeks late when everyone at Carter exploded. Suddenly, no one remembered that the application would be late. No one remembered being warned that this one, small change would cause so many problems. Maria was to blame for a poor design that could not be made to work. Crater now said that he 'requested' the change but that it was not absolutely 'necessary.' Blake forgot the conversations, memo, and approvals. Dare was furious because his special project was now overbudget and late.

When the written memo and other documentation from the meetings held at the time were produced, Dare's comment to Maria was, "You are the expert, you should have asked again whether or not the change was necessary. You were the only one who knew how big it really was!"

In the end, the application was put into production with only one run through the finishing plant per work order. Reworks were assigned a new number and tracked as if it were the first time through the process. The costly change and insufficient whistle-blowing by the project manager led to unhappy clients, overworked project team members, and a less than optimal application. Could they have been avoided? Probably not. The client should have been made to realize the magnitude of the change, however. Maria and Charlie should have been more insistent on a detailed review of the request and sign-offs for this major change.

How should programs be packaged to fulfill processing constraints? Examples might be to provide five-second response time; to provide completion of reporting within a three-hour period daily; or to provide 24-hour access to information that is up-to-the-minute.

Other elaboration activities to be decided include common routines for commonly used processes. For instance, how will screen processing be performed? Will each programmer write his or her own version of screen interface or will there be common modules for screen interactions? The scope and details of system 'utility' programs to be used by all programmers are defined.

The last major elaboration activity is to examine the application constraints. We ensure that each constraint is considered in the design and that processing is within the prescribed limits.

3. **Synthesis**—Build a unified physical design of the application, reconciling any parts that do not fit and representing requirements in more detail. We may add functions to the application that are environment specific. For instance, in a mainframe IMS database environment, applications require user views, data base definitions (DBDs), data control blocks (DCBs), and data service blocks (DSBs). These control blocks are not required if using dBase IV on a PC. The representation can be either manual (i.e., on paper) or automated, using computer-based tools.
4. **Review**—Perform quality control. At the end of the phase, conduct a design walk-through, comparing design to logical requirements to validate completeness and correctness. Rean-

alyze schedule and staffing for coming stages of implementation, testing, conversion, training, and turnover, revising them as required.

5. **Document**—Create useful program specifications and an overall design document. The design document describes the database, application structure, constraints, and so on. Graphics and supporting text document the design. The program/module specifications include the details of processing, all interface designs, and any specific information required to develop the program.

As in analysis, these activities vary by methodology because the ending point of analysis, which provides the input to design, is different. However, the intention of all methodologies is to define the application such that programming and implementation can be started after the design is complete. Program/module specifications, in some form, are the desired output of the design phase.

Keep in mind that even though we discuss design as a straightforward mapping of 'what' to 'how,' it is not a one-to-one mapping. You might need to compromise analysis requirements during design. **Compromise of requirements** means that they may be rescoped, manipulated, dropped, or otherwise changed to fit the environment's limitations.

Prototyping is an important activity in design to minimize the amount of requirements compromise that takes place. Especially when you use a package or language for the first time, prototyping should be used because prototypes frequently find the language's limits. You must verify that the application structure and concept can be implemented using the software as planned. Frequently in a PC environment, you will find you are bumping into language/package limitations that cause you to rethink the design. Vendors call this process 'work around.' You are finding a way to **work around** the built-in limits of the language. Vendors will usually help find a work around if the application cannot be built in known ways. They also challenge users to find work arounds and broadcast them to others who have similar problems.

The linkage between analysis, design, and program design is looser or tighter depending on the methodology and implementation environment. For instance, data information required differs if we use dBASE IV[2] or if we use IMS DB/DC.[3] Level of requirements detail differs if we use the Focus[4] language or if we use C-language.[5] Where possible, we point out specific instances of these linkages.

You, as the SE, must constantly check your mental model of functional requirements when building a mental model of *how* they will be implemented. Do not be afraid to try different ways of thinking. Frequently the old way was not too good. We get trapped in our thought processes and don't even remember to do the **out of the box** thinking[6] that is necessary for innovative designs.

Before we discuss methodologies, some organization and automated supports that facilitate application development regardless of methodology are discussed.

Organizational and Automated Support

Organizational innovations that are useful with all methodologies are joint user-IS application development activities, user managed application

2 dBASE IV is a trademark product of Ashton-Tate, Inc.

3 IMS DB/DC is a trademarked mainframe product of the IBM Corporation. IMS, Information Management System, is a hierarchic database product. DB stands for *data*base; DC abbreviates *d*ata communications.

4 Focus is a trademarked database, query, application generator, expert system product of Information Builders, Inc. Focus is thought of as a 4th-generation language because of its powerful query capabilities.

5 C-language is a trademark product of Bell Labs; C++ is a trademarked product of Borland International; and there are other versions of C-language.

6 Out of the box thinking means to rethink the entire process as if the current methods, procedures, and policies did not exist. Put yourself in the shoes of a caveman (or an intelligent child) who just walked into the company, and redesign the work as they might. Question everything. For instance, who says you need to keep a copy of an order? What is the *real*, i.e., legal requirement?

development, structured walk-throughs, and data administration. The goal of these organizational innovations is to speed the development process, foster user participation, and improve the quality of the resulting application. Automated support for structured analysis and design comes from computer-aided software engineering (CASE) tools. Each chapter will identify CASE tools that relate to the phase and activities. In this section we describe the characteristics of CASE tools and the ideal CASE environment.

Joint Application Development

Several techniques have been developed to describe the joint, intensive definition of application requirements—**Joint Requirements Planning (JRP)**, **Joint Application Design Development (JAD)**,[7] and **Fast-Track**.[8] They are all similar in that the goal is a collaborative, user-IS definition of application requirements. The planning and execution of a joint session are also similar. The differences are the level of participants, subject matter, and level of detail of the discussions. These are more fully described below.

JRP is an executive level user-IS activity to identify overall requirements at the enterprise level. Fast-Track and JAD both are designed to produce a functional requirements specification. If a JRP report exists, the Fast-Track/JAD uses the JRP report as constraining or defining the business environment within which the application is defined.

JRP, Fast-Track, and JAD activities are

- designed to shorten the application development process
- productivity tools
- structured to improve the quality of the application development deliverables.

These characteristics of the joint development activities can also provide opposite results if the sessions do not adhere to the guidelines defined by their developers. However, these techniques *do not substitute* for experience, good project management, or knowledge about the application! Even with user involvement in analysis and design, application developers *must* develop knowledge and shared mental models of both the application and problem domain. One purpose of the joint sessions is to be sure of a common mental model for all participants.

Requirements for a joint session relate to:

- the team
- the session
- joint structured process
- the meeting facility
- documentation tools.

The Team

The team is composed of client representatives, facilitator, systems representatives, and support personnel (see Table III-4). The clients must include decision makers at a high enough level to resolve conflicts and make decisions that affect the scope and content of the application. They must also be at a low enough level to be conversant and able to explain the daily functions and procedures. Finally, clients must represent *every* functional area affected by the application. You must also keep the number of client participants less than 15 and ideally between three to four people. The more people, the longer the process and the more difficult the decisions. Ideally, the whole session team is about seven people.

Systems representatives should include the project manager, an SE, and one to two analysts with technical expertise. The systems representatives must be able to assess feasibility of requested requirements and the expected complexity of implementing the requirements in the target environment. The main role of the system representatives is to learn the problem domain area during the sessions and ensure accurate problem restatement in system terms.

The facilitator is a specially trained individual who runs the session. The facilitator has several roles:

- Elicit information from participants
- Keep the meetings moving

7 JRP and JAD are design techniques of the IBM Corporation.

8 Fast-Track is a design technique of the Boeing Computer Company.

TABLE III-4 Joint IS-User Team and Responsibilities

Role	Job Title	Responsibilities
Facilitator	Consultant IS Manager Senior SE Facilitator	Elicit information. Keep meeting moving. Minimize monopolization by one or few individuals. Identify and resolve conflicts. Maintain professional atmosphere.
User	Manager Professional Clerk	Make decisions about compromises, changes, or other aspects of the application requirements that require managerial approval. Participate in and contribute to discussions about requirements. Provide information, requirements ideas, and suggestions on the meeting topic. Maintain open, professional atmosphere. Interpret and explain application problem domain to IS personnel.
IS Representative	Project Manager Project Leader SE Systems Analyst	Learn the application problem domain. Assist in interpreting requirements into graphical representations. Determine technological capabilities and limitations as they relate to the application requirements. Interpret and explain technical IS domain to users.
Support	Secretary Systems Analyst	Take notes as requested. Plan for coffee, meals, etc. Act as liaison with outside world. Take notes as requested; assist in transcribing and documenting daytime work.

Keep the discussion from becoming monopolized by one individual
Identify and resolve conflicts
Keep the meeting on a business (rather than personal) level.

Frequently in joint sessions, organizational disagreements on goals and objectives arise. Such conflict is to be expected and is normal. The facilitator's job is to identify and ensure resolution of disagreements during the sessions. The conflicts are potentially explosive and can lead to personal conflicts. The facilitator must recognize such situations and defuse them. Occasionally, defusing means asking for a participant to be replaced.

The facilitator is a cheerleader, meeting leader, and ring leader who keeps the session moving. Usually facilitators are senior staff from the information systems organization who already know how to develop application requirements, but who are specifically trained to facilitate joint user-IS sessions.

Finally, support personnel are individuals who take notes during the day and provide liaison with the outside world. The notes include data-related

information and process-related information. Data information includes identification, naming and definitions of entities, elements, and entity relationships. Process information includes decision rationales, process identification, procedural details of processes, and policies that constrain processes. The actual results of the data and process discussions are reflected around the room (see the photo in Figure III-2) on flip-charts, blackboards, and other visual aids that are always accessible to the entire group.

A second kind of support is administrative assistance, which includes documenting the information during evening sessions, coordinating coffee and meals, and ferrying messages to and from work for participants.

Preparation

A meeting to prepare session attendees should be held for *all* participants. The primary purpose is to give participants a list of tasks to complete before they attend the joint session and to train participants in the completion of the tasks.

The meeting includes an orientation, document examples, data requirements, and training in development of graphical techniques being used to document processing. The orientation discusses the expectations of the organization and normal results of such sessions. Then participants are given an overview of the joint structured process: what it is, how it is conducted, proper behavior, and decision-making necessity. The scope and purpose of the application are discussed and agreed upon again by all participants. If there is disagreement or problems with the scope, they are revised at this meeting so everyone has a shared understanding of what work functions and information are in, and what are not in, the application.

If data flow diagrams are the graphical technique being used, for example, the users are trained to develop a context diagram and first-cut data flow diagram of their current job. The list of tasks for data flow diagrams would include the following activities:

- Define the scope and functions of your position
- Document the 'what is' in a data flow diagram
- Try to draw a context diagram of all the departments, groups, and applications with whom you exchange information in your job
- Define all data used in your job

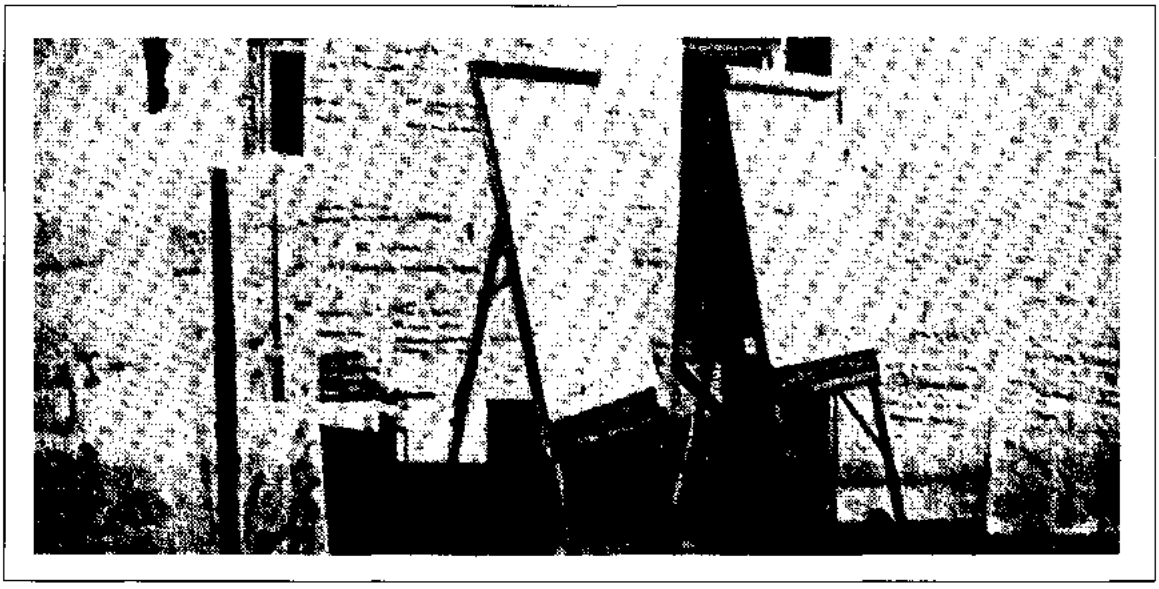


FIGURE III-2 Photo of Joint User-IS Session Room

- Collect statistics—how often, how much, when—for all data and processes
- Collect samples of all input and output documents.

Frequently these sessions are taught by an in-house facilitator, but they may be taught by a consultant who knows the techniques.

The Joint Structured Process

The ideal joint user-IS session is full-time, off-site, lasts three to five days, and has five to nine participants. All of these ideal characteristics can be loosened somewhat and still maintain the momentum that comes from intensive work sessions. The idea is to do the work intensively and quickly because no one has time to spend in months of meetings. Participants become very close and frequently become good friends as a result of working together. At best, the users and IS team realize they are business partners in the application development and that relationship prevails throughout the project's life.

Joint sessions are divided into mainly daytime and nighttime sessions. The word *mainly* is used because the activities can be done at any time. In general, daytime, when people are most alert, is devoted to creating new information; evening is devoted to documenting the new information.

During the day, activities are the following:

- Confirm business functions
- Identify and analyze specific requirements (processes by function, inputs and outputs for each process). For each process, identify what is done, how frequently, exception and error processing, periodic processing, problems with current procedures, policies that might need to be changed, and any new business requirements relating to the processes.
- Identify general requirements for the application. For data, how accessible and accurate does the information need to be? Can it be accurate as of close of business yesterday or must it be up to the minute? Can answers take one or two hours, or must the answer be within seconds?

  Application constraints are a second type of general requirement. Constraints place limits on the application. For instance, upper bounds of cost and time are allocated for development, hardware, software, language, or DBMS. These constraints are general, but they place strict boundaries on how the application will be designed. They also identify, to the technical staff, activities that need to be further elaborated during the detailed design to accommodate the implementation environment. Constraints from the first chapter discussion also apply. They include time, pre- and postrequisites, structural, control, and inferential constraints.
- Identify the likelihood of requirements change over the next three to five years. If requirements are identified as changing within the expected implementation time of the project, then the expected requirements become the current requirements for the application.

  For instance, users may currently need data up to the close of business yesterday. They discuss the industry as moving rapidly toward instant access of up-to-the-minute information and expect this requirement within 12–18 months, and the application will be implemented in 12 months. Build the new requirement into the application now to be an early leader and avoid costly redevelopment of the new application.
- Have the support staff record all processes, functions, data, outputs, data elements, terms of processing, names given to items, and so on.

Figure III-3 shows the first-cut data flow diagram developed by an accountant in a major company for a JAD/Fast-Track session. The *user*, after one training session, developed a DFD that was about 90% correct. Figures III-4 and III-5 show the related Level 0 and Level 1 diagrams, respectively, from the JAD which had minor changes during IS design. Figure III-6 shows the DFD level 2 as decomposed by the project team during design. Only one of the processes changed: General Ledger was elaborated to be Accounts Receivable and Accounts

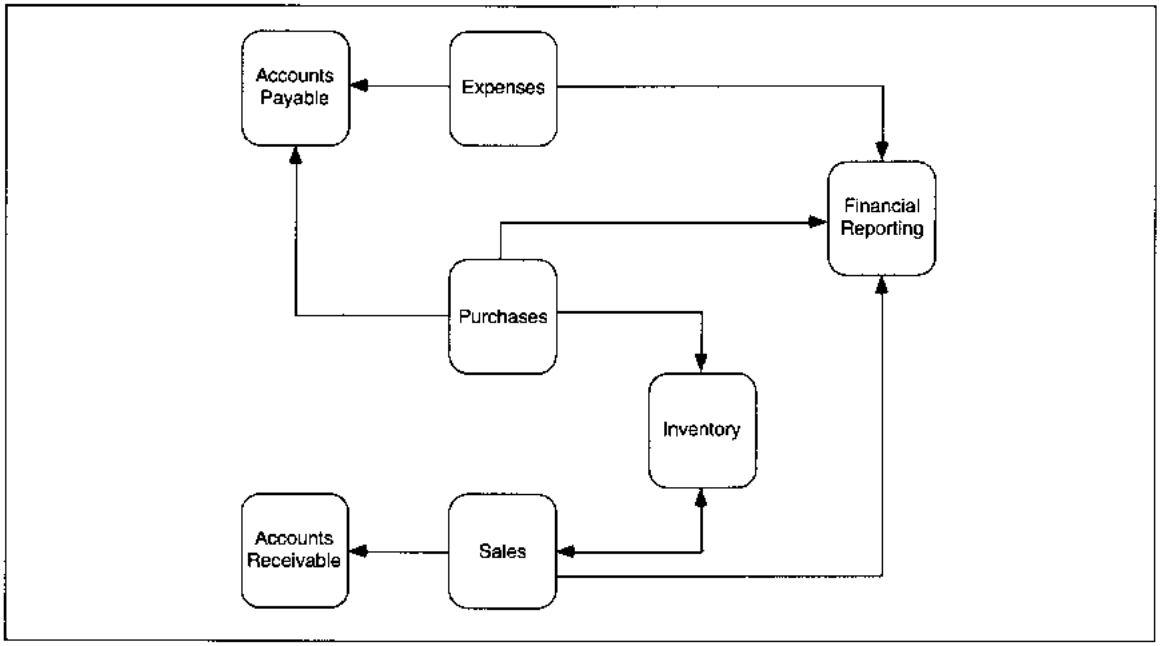


FIGURE III-3 User-Developed First-Cut DFD

Payable. The other changes were to files and external entities.

The evening sessions do the following:

Define all elements and terms
Document all processes
Draw formal DFDs
Document general application requirements and write an executive summary
Review documentation output of other mini-teams.

The group works together during the day to create information. In the evening, the group splits into mini-teams to perform one of the above activities. Documentation should be done using automated tools, including word processors, CASE tools, or other automated support tools that might be available. The goal is easily modifiable documents that can be formatted and printed.

When the mini-teams complete their work, they jointly review each others' work products. This review fosters the shared common view of the application and ends the participants' day with each having a clear sense of what was accomplished.

The Meeting Facility

The location should be at least 20 miles from the main work site of the participants to minimize interruptions and preclude people being pulled out of the sessions. The facility should provide above average meeting, sleeping, and eating arrangements in the same building. Phone access must be available but must be removed from the meeting room(s). The facility must provide computer accessibility. The location must be easily accessible for managers, who are not participants, to attend sessions for resolving conflicts. The facility must allow use of walls in the meeting room. The room should be equipped with flip-charts, overhead projector, markers, slide projector, and other meeting equipment as needed.

Documentation Tools

Documentation tools should include some word processing capability, dictionary support, and some graphical form support. All of these should ideally be in a computer-aided software engineering (CASE) tool. The CASE tool should allow customized reports of the information and should

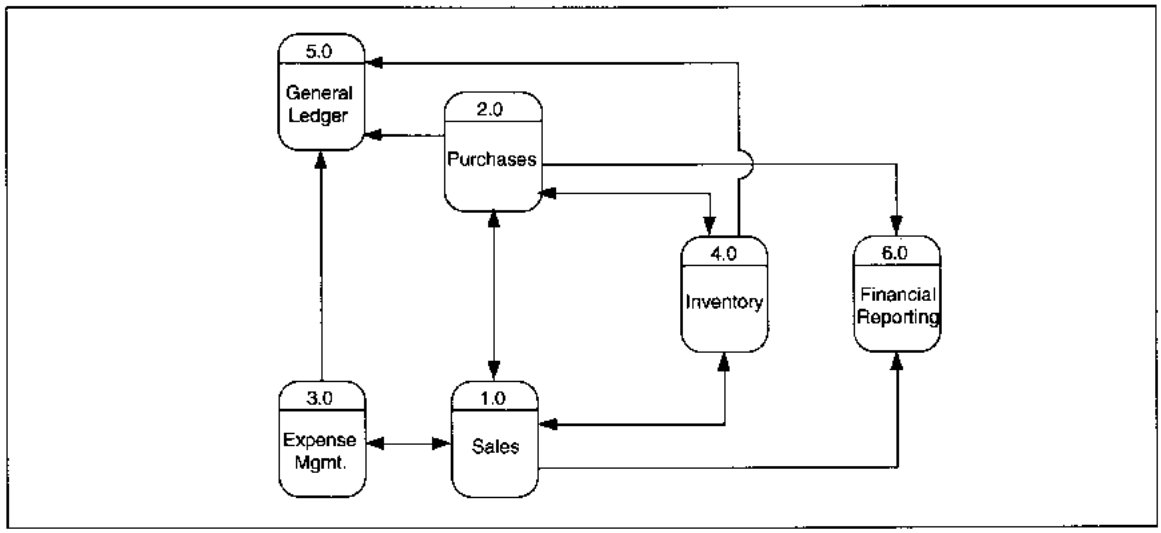


FIGURE III-4 JAD Team First-Cut and IS Final DFD Summary Level 0

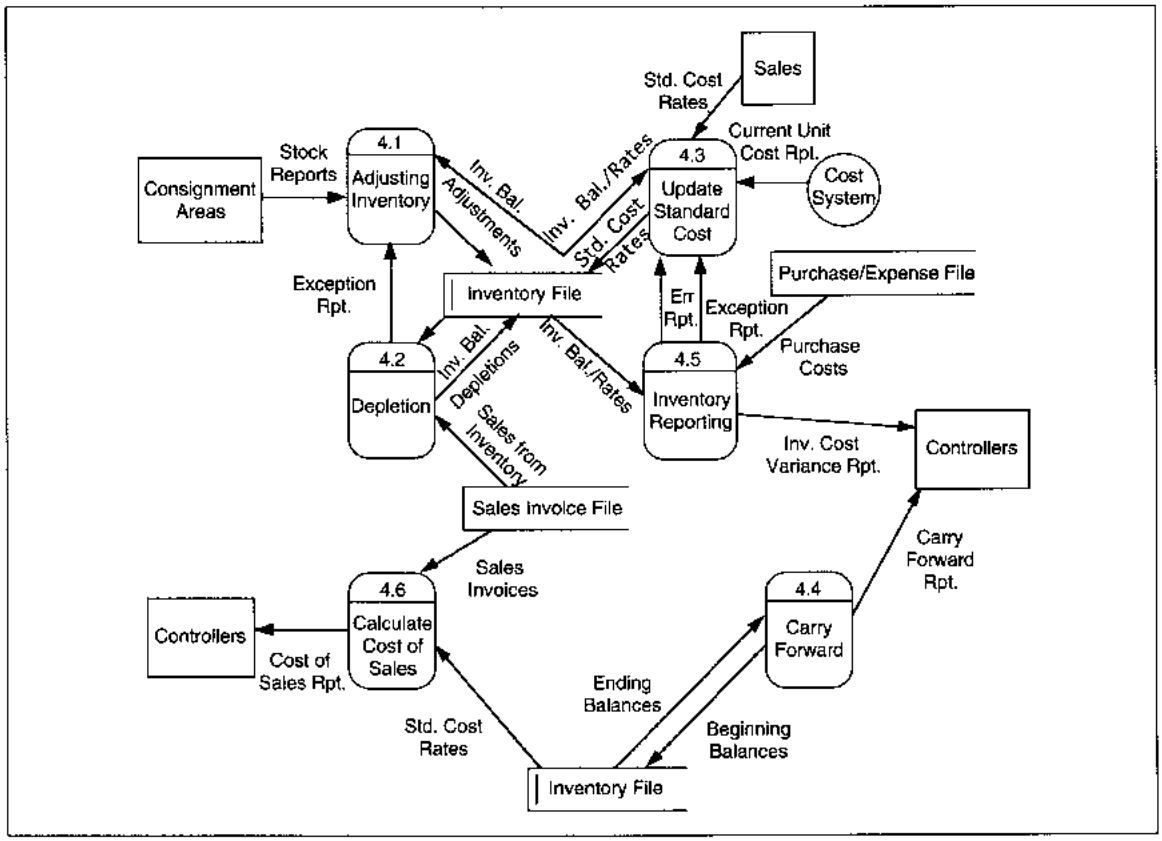


FIGURE III-5 JAD Team First-Cut and IS Final DFD Level 1

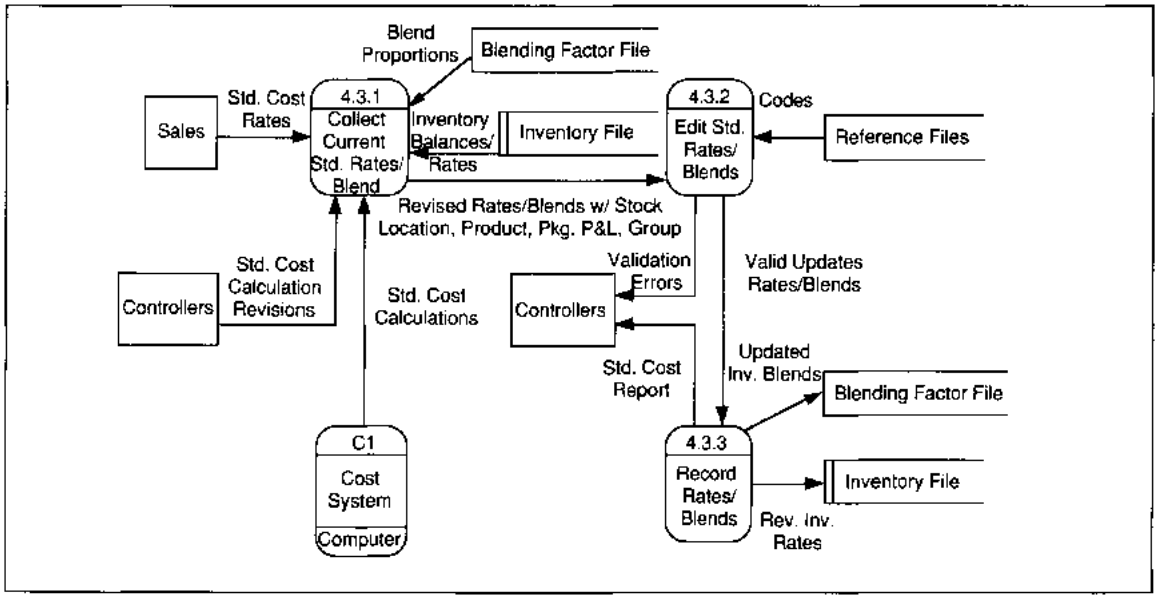


FIGURE III-6 IS Level 2 DFD for Updating Standard Costs

provide some intelligence on checking and cross-checking both completeness and accuracy of the information entered.

At a minimum, word processing should be provided via some tool such as WordPerfect,[9] Word Star,[10] MS Word,[11] and so on that allows graphics to be imbedded in text, creates tables easily, and does full text manipulation.

An **active data dictionary** is desirable for documenting the objects (e.g., entities, files, flows, objects) and object relationships defined during the sessions. An active dictionary is one that allows custom report development, provides intelligent assessment of completeness, and identifies potential duplicates based on name and definition. If a passive dictionary (i.e., has only vendor reports and no intelligence) is an option, you are better off using a word processor to document the information.

A graphical drawing tool is the third type of software needed. The tool should allow the type of drawing you are using with your methodology. An automated graphical tool is preferred to manual drawing because automated drawings are more easily changed and maintained. The joint groups frequently do several iterations of a drawing before they are satisfied with the result.

To summarize, joint user-IS sessions are a way to obtain quick results with a high degree of user participation in the development of requirements plans and application requirements. Joint sessions are intensive and require high commitment from participants. The rewards are a user-centered requirements document that frequently leads to more satisfied users and high user involvement throughout project development.

User-Managed Application Development

Joint sessions are designed to bring users and IS personnel together with the underlying understanding

9 WordPerfect is a trademark of WordPerfect, Inc.

10 Word Star is a trademark of Word Star, Inc.

11 MS Word is a trademark of Microsoft, Inc.

that users will always know more about their jobs than IS people. Joint sessions foster commitment to the IS development effort and give users a sense of participation. The user aspects of application development should not stop there. A **user manager** should be appointed for the application and should be the person ultimately responsible for the successful completion of both the application software and the organizational changes that accompany a new application.

The need for user-centered design seems obvious. **User-managed applications** foster a sense of business partnership; **IS-managed applications** foster a sense of them-and-us. User-managed applications provide a regular, natural communications line between the technicians and users; IS-managed applications provide a way for IS people to only talk among themselves. User-managed applications tend to require less IS involvement in application training, because users do their own training; IS training is notoriously condescending, inappropriate, and ineffective. Users 'own' the application and train their own staffs.

Not all is rosy with user managed applications. If the IS project manager is not used to working for a user, she or he will have to adjust some aspects of work. For instance, conversations will use business terms rather than technical terms. Variances in time and budget will require explanation and discussion. Rather than running the whole show, the project manager is clearly relegated to a supporting roll and only manages the actual software development.

User-managed development can also be subverted by unsupportive IS personnel. For instance, user teams can meet to develop functional requirements, but IS teams may not use them. IS groups have been notorious in ignoring user requirements. The comment heard is, "They can tell me anything, I'll give them what I want." The attitude is that mere users could never define as good a system as an IS person. How someone who does not know the business could make such a statement defies logic, but it is made. IS developers frequently need indoctrination that the business partnership aspect of application development *does* extend to the users.

Structured Walk-Throughs

Have you ever had a program bug that you spent hours trying to locate? You give up in frustration and turn to a friend for help. The friend takes a sideways glance and says, "Oh yeah, this period is out of place." Just like that, your hours have been a waste. That type of easily seen error is not a fluke. Your friend is not necessarily a genius, just as you are not necessarily stupid for not finding the error. The phenomenon at work is that you are too close to the problem to see the 'big picture.' At some point, we all reach this stage regardless of where on a project we work. Walk-throughs were designed to formalize the 'friendly review' described above.

A **walk-through** is a semiformal presentation of some work product for the sole purpose of finding errors. Work products might include all or part of the following:

- Functional requirements specification
- Project plan
- Design specification
- Logical or physical database design
- Program specification(s)
- Program code
- Test plan
- Test design.

This list is not complete. Its purpose is to give you an idea of the range of items that can be the subject of a walk-through. Virtually any work product, or piece of a work product, can be reviewed using the walk-through technique.

Ideally, a walk-through should not be scheduled for more than two hours at a time. If more time is needed, then additional walk-throughs are scheduled. Like all rules of thumb, this one is frequently broken. Participants who do not work on the development team sometimes have a difficult time walking-through application requirements in bursts. When they focus on the application, they like to see everything at once. So, occasionally you might have a marathon session that runs a whole day.

Walk-throughs are formalized in that there is preparation, a team with members having different responsibilities, and a process. Preparation for the

session is as follows: The team is identified and approved by an SE or project manager. The day, place, and time are agreed upon. A memo of meeting details is sent to all participants several days in advance. *Attached to the memo is the work product* to be reviewed.

All participants are expected to review the work product, annotating questions and potential errors in the margins. They must come to the session already having some understanding of the work product.

Participants in a session include the facilitator, work producer, one or two peers who are on the same project, one or two outsiders, and a scribe. Ideally, the number of participants is between five and seven. The **facilitator** is much like a JAD facilitator. He or she keeps the meeting moving, makes sure no personal or blaming remarks are allowed, and maintains focus on the work product.

The producer presents his or her work. First, an overview focuses attention on the purpose of the product. Then, the work is reviewed in a page-by-page or line-by-line manner following the logic of the document. The peers and outsiders are there to question the correctness, completeness, efficiency, and effectiveness of the product. Questions, comments, or errors are discussed as the presentation is made. When an issue is raised and appears legitimate, the scribe notes the problem and its location (see Table III-5).

Possible 'outsiders' who might attend a walk-through include representatives from auditing, quality assurance, operations, or other project teams who need to approve or work with the final product.

After the session, the scribe types the notes and presents a memo to the author for resolution. The author then responds to each item (see Figure III-7). If an item is an error, the response details how and when it was fixed. If the item is an efficiency or effectiveness issue, the response describes what research was done and the resolution. Depending on the extent of problems or the importance of the product, another walk-through might be held. Usually, if the products are for analysis or design, two or three walk-throughs are held. If the product relates to program or test design, then the number of walk-throughs is determined by the number of errors. With less than 10 errors, only one walk-through would be needed.

Data Administration

Data administration is the management of data to support and foster data sharing across multiple divisions, and to facilitate the development of database applications. The principle activity for the organization is the development of a **data architecture** which depicts the structure and relationships of major data entities, such as customer, vendors, and orders. A data architecture is similar to the frame of a building. Once the frame is constructed, the siding and façade are added. The frame provides the skeleton to which the other substructures, such as electrical wiring and plumbing, are added. In information systems, the data architecture defines automated and nonautomated data and how they are used in the organization. The architecture provides a 'frame' for defining new applications and documents all data uses and responsibilities for existing applications.

The other major organization level activity is defining, with users, data that is 'mission critical' for the organization. **Critical data** is defined as that data required to maintain the organization as a going concern. As such, critical data is subject to management and standards through the data administration function. Noncritical data is data that, while useful, is not required to maintain the organization in event of a disaster. Noncritical data does not require the same degree of management as critical data.

At a more detailed level, data administrators develop, administer, and maintain policies and standards regarding data definition, sharing, acquisition, integrity, and security for the corporation's data resource. Data administration provides guidance to project teams on storage, access, use, disposition, and standardization of data. Data administrators are responsible for maintaining corporate definitions in addition to the creation and maintenance of the data architecture representing the enterprise.

Historically, the motivation for data administration relates to a maturing organization. When DBMS software was installed in most organizations, a **database administration (DBA)** group was created to

TABLE III-5 Example of Errors Found in Walk-Throughs

Walk-Through Type	Representative Errors Found
Feasibility	1. One of organization, technical, or financial analyses is missing. 2. Financial analysis has mathematical errors. 3. Typos or poor English render the document (or some part) incomprehensible. 4. Analysis contains incorrect information.
Analysis	1. Data elements for data store, file, or other structure are incomplete. 2. Data items do not have formal names or names do not conform to standards. 3. Subsystem specification unclear. 4. Obvious 'holes' in the system as specified. 5. Graphical representations contain syntactical errors or confusing, ambiguous terms. 6. Nature of application interfaces not fully specified.
Logical Data Model	1. Logical data model (LDM) is not in third normal form (3NF). 2. Names do not conform to standards or are ambiguous.
Design	1. Mapping to implementation environment does not include all functional requirements. 2. Implementation as specified will be difficult to operate, maintain, or implement. 3. Design is incomplete . . . one or more screens are missing, screen dialog is incomplete, allowable navigation not provided, etc.
Physical Data Model	1. Physical mapping does not provide necessary user views and security simultaneously. 2. Numerous user views may be unwieldy in implemented environment. 3. Physical model does not provide growth anticipated.
Program Specification	1. Program specification does not clearly say what the program is to do. 2. Program specification does not map to design or functional requirements. 3. File requirements not specific . . . missing user view, copy lib name, JCL, etc. 4. Logic specification incomplete. 5. Faulty logic. 6. Access control for secure data not present.
Acceptance Test Plan (This could be any test plan)	1. Test plan does not test that all requirements are met. 2. Test case x data cannot perform as specified. 3. Missing/erroneous predicted results for reports, screens, file contents, or messages. 4. Missing on-line test dialog for single user functions. 5. Missing scenario and test dialogs for multiuser test. 6. Results predicted cannot be attained with current test design. 7. Test for breach of security missing. 8. Specific audit control tests missing/faulty.
Code	1. Logic error—missing, extra, or wrong logic. 2. Nonstructured format will make maintenance difficult and expensive. 3. Comments do not identify module linkages. 4. Comments on user view copy books do not clearly identify the database, user view, or JCL. 5. Access control for secure data not present. 6. Control totals for end of program counts missing. 7. Format error on report. 8. Misspelled word on screen, report, etc.

Consolidated NY Bank
InterOffice Memo

DATE: December 7, 1992

TO: Ms. Sandra Jones,
Walk-Through Facilitator

FROM: Mr. John James,
Producer

The following table includes all errors found during the Requirements Walk-Through on December 1 (see H. Hines, Scribe memo of 12/2). Each item has either been resolved or found not to be an error as indicated. One item, #5, identified an audit problem for which I am awaiting Audit Dept. resolution. They are supposed to respond by next Friday, December 11. Since we decided not to have another walk-through, I will proceed with finalizing the analysis phase.

Error #	Error Page	Description	Resolution
1	2	Overview inconsistent in treatment of errors for transactions.	Rewritten
2	10	System access code design not clear.	The lack of clarity was deliberate to prevent general access to security procedures. The group felt that the document should contain all of the information. Upon reviewing this request with Mr. Fields, Project Manager, we decided, for security reasons, not to include the information. Mr. Fields has a detailed description of security procedures and the document now refers individuals requesting the security information to him.
3	63	Test of screens is incomplete.	Missing information was added.
4	125	Security for accounting data not clear.	Same as #2.
5	127	Interface to accounting system has inadequate control counts and security.	Referred to the Audit Dept. for recommended action.

FIGURE III-7 Sample Walk-Through Error Resolution Memo

maintain and monitor the DBMS' use. There was no necessity for other data-related organizations because applications, for the most part, were isolated from one another and data sharing across organizational boundaries was low. Most industry followed this pattern of development.

In the normal process of maturation, companies realized that sharing and consolidation of databases across organization boundaries was desirable. The need to share data frequently accompanies the realization that individual division and work groups have their own vocabulary which often overlaps or conflicts with the vocabulary and terms used by other work groups. When divisions automate data, they incorporate local rules, policy, and definitions in their applications. Data, while having the same

name, then, may have several different meanings, uses, formats, and connotations across an organization. Conversely, data may have different names but the same definitions. This lack of consensus about terminology and data characterizes predata administration organizations.

The lack of consensus about data definitions leads to the realization that data standards pertaining to definitions, usage, ownership, security, access, and maintenance are not only desirable, but mandatory, in large-scale development of shared databases. This need for standardization increases with the recognition of data as a shared resource of the organization.

A formal data administration function is needed to define and manage data company-wide. Data administration requires recognition and commitment to the notion that data is a resource of the corporation. As a critical corporate resource, data requires the same careful planning and on-going management as cash on hand, office equipment, or personnel.

Commitment to DA is sometimes difficult to develop because data are fundamentally different than other resources. Data are abstract and nonphysical, do not decay, and are easily replicated as the need arises. They are also subject to different confidentiality, accuracy, and access requirements. Data are all of these things. In service industries, especially, information is a primary product, and the quality of the data resource directly affects the company's bottom line and how customers perceive the quality of service delivered. Data administration consolidates information across the organization to simplify the development of applications to service customers.

Benefits of data administration outweigh the frustrations and difficulties of establishing the function. Some of the benefits include:

1. Creating and documenting a data architecture leads to formal recognition and agreement of business rules and relationships which are inherent in the data. This agreement improves communications and understanding of corporate data.
2. By defining and documenting data only once, efficiencies are realized throughout the system development life cycle. All subsequent application—using previously defined data items—identify data required and obtain access to already automated data. The data design and documentation phases are shortened. Edit routines are reused, just like the data definitions, and ultimately the cost of program code is reduced.
3. Data administration leads to faster response to changing business conditions. The development of applications to support new products, for instance, can be speeded due to fully specified definition of data required to support a product.
4. Data administration provides a means for deciding what data must be controlled as part of the corporate data resource, and what data can be user-owned and controlled (including data that is off-loaded to PCs and LANs).
5. Data administration maintains definitions of *all* data in the corporation regardless of hardware platform or criticality. The central repository for this information, then, becomes the focal point of data-related activities.
6. By fostering data sharing, the cost of creating, sorting, updating, and backing up multiple copies of the same data items is reduced, if not eliminated. That is, we only introduce *planned data redundancy*. Just as DBMSs allow us to minimize intraapplication data redundancy, DA allows us to minimize interapplication data redundancy.

In summary, the creation of data administration is recommended to guarantee minimal redundancy, shared understanding of data item definitions, and a managed approach to providing for future database environments. Data administration should not be confused with DBA data management which includes physical DB design, disk space allocation, and day-to-day operations support for actual databases.

Data administration has numerous interfaces both within and outside of the IS area. Therefore, data administration interfaces occur at all levels of all divisions specifically to perform user liaison and application liaison.

User Liaison

The data administration function works with business areas to define the data which that area uses to perform its function. All data, whether it is under the control of a current information system or not, is subject to data administration review. Thus, all data on any hardware platform is subject to review. During the review, critical data entities and data items are defined, maintained, and managed by data administration. Applications with critical data will be required to comply with standards on data, access, and security.

The person performing user liaison must be able to understand and converse in business terminology, not technical jargon. He or she should have problem-solving and analytical skills but also should have excellent communication/negotiation skills, user orientation, and understanding of the role and functions of data administration. The individual must be able to translate user data, definitions, and rules into information in the corporate data repository.

Application Liaison

Data administration works with application project teams to define the data requirements of the application. The data administration analyst identifies what data is already automated and works with the project team to define logical descriptions of the data. The DA analyst, DBA, and project analyst together transform the logical database definition into a specific database's logical definition. The DA analyst down-loads the data definitions from the corporate central repository for use by the project team and DBA. DBA then works to develop a physical database definition of how best to store the data.

In project-oriented work, the project analyst and DA analyst reconcile all data requirements with existing information in the corporate repository. For instance, if a team needs a "plan" field, but their definition varies from that of the corporate definition, one of three actions is possible:

1. The corporate definition is changed to accommodate the new information.
2. The application redefines its use to be consistent with the corporate definition and usage.
3. A new data item is defined by the project analyst and DA; the new item is entered into the corporate central repository by the DA.

The skills, then, needed to perform application project liaison include analytical, communication, problem-solving, negotiation, data analysis, and modeling skills.

Where in the Organization is Data Administration

Ideally, the recommended organizational location of the DA function is independent of the corporate IS area, reporting to the president of the business entity it supports. DA affects and interacts with all departments and areas of the organization, including all of the application development groups as well as users, regardless of organizational position or hardware platform. The DA group could be part of an internal consulting/technology-related organization whose mission is to provide services across the entire organization. The DA group should be neutral about hardware, software, development, or management of applications as long as the data is not defined as critical.

CASE Tools

Computer-aided software engineering (CASE) is the automation of the software engineering discipline. You will find descriptions of ICASE, Upper CASE, and Lower CASE. These are variations on the theme with 'I' standing for 'integrated,' 'Upper' standing for conceptual or logical design only, and 'Lower' standing for programming support only. While these differences do exist, this text concentrates on CASE tools that support at least the analysis phase and may support others; they are all called 'CASE' here. We will identify which phases are now supported (of course, this might change by publication time).

The typical CASE environment includes a repository, graphic drawing tools, text definition software, repository interface software, evaluative software,

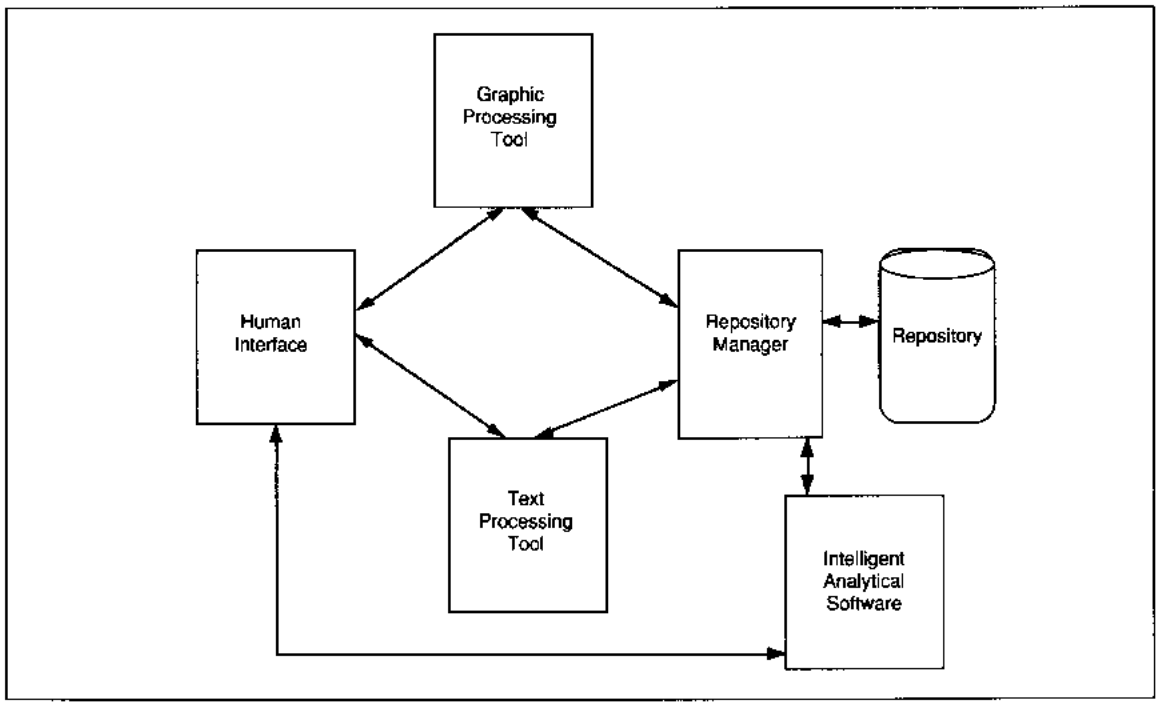


FIGURE III-8 CASE Architecture

and a human interface (see Figure III-8). A **repository** is an active data dictionary that supports the definition of different types of objects and the relationships between those objects. Graphic drawing tools support the development of diagram types and evaluates the completeness of the diagram based on predefined rules. Text software allows definition of names, contents, and details of items in the repository. The interface software is the interpreter which determines the form the data should take (either graphic or text). Evaluative software is the intelligence in CASE. Evaluative software analyzes the entries for a diagram or repository entry and determines if they are lexically complete (i.e., conforms to the definition of the item type), and if they are compatible with other existing objects in the application. The human interface provides screens and reports for interactive and off-line processing.

In this section, we discuss the characteristics of the ideal CASE environment. This is just an ideal and is the author's own invention.[12] No commercially available products and no research prototypes are known to embody this ideal.

The ideal CASE should provide complete automated support for the entire project life cycle, beginning with enterprise level analysis and working through to maintenance and retirement. The ideal CASE then becomes the focal point for all work that takes place in software engineering, and the work of the SE concentrates on the logical aspects of design. The ideal CASE tool would provide for the technical, data, and process architectures of the organization, project planning and monitoring,

12 The ideal CASE in this section is partially the result of research done with Judy Wynekoop, UT San Antonio and Nancy Russo, U of Northern Illinois, published in Wynekoop and Conger [1991], Conger [1989], Conger and Russo [1990]. It also results from 10 years of frustration in using CASE tools and waiting for vendors and researchers to build decent products.

group work on applications, application and manual procedure definition, normalization of data, DB schema generation, generation of bug-free code in a user-selected language, automatic testing of generated code against the application logic, and intelligent assessment of completeness and correction along the way. Really advanced CASE would recognize components already in the repository for reusability of analyses, designs, and code.

The repository of CASE determines both what is supported and, to some extent, how much support can be provided. The repository is something of a *super dictionary* that captures and maintains meta-data. Meta-data is information about data (see Chapter 1). For example, a data element in an application is data, and its attributes constitute the meta-data that would be stored in the dictionary. Attributes of an element include, for instance, data type, size, volume, frequency of change, and edit criteria. A **CASE repository** acts as the DBMS for the engineering effort, provides the capability for expanded meta-data capture, and maintains all components and their interrelationships.

The ideal repository should allow customizing of the methodology supported and enforcement software that can evaluate the correctness of user-defined repository entries. To do this requires some decoupling of the repository from a specific methodology and an abstracting of methodology compliance rules within the repository. These are not trivial tasks! This decoupling would allow organizations to adopt and use the components of methodologies that work for them, and ignore those that don't. The initial sacrifice for this capability will be less intelligence. But, decoupling the intelligence from a specific methodology and type of repository entry will also allow customizing of evaluation software and enforcement of local rules.

Intelligence in CASE comes in two major forms: intelligence of the interface and intelligence of the CASE product itself. The interface should provide both novice and expert modes of operation. It should allow work to be saved and restarted as part of the functionality. The tool should be customizable by individual users. For instance, if I want yellow print on a blue background, and I call a data flow diagram a DFD, I should be allowed to change the defaults to use my terms and formats.

Alternate forms of inputs should be reflected throughout the diagram sets. This means that if a user enters entities and attributes in a repository, when she or he moves to developing a graphical entity-relationship diagram, the information in the repository should be reflected on the diagram.

Intelligence of the CASE product includes analysis within and between both diagram types and repository entries. Ideally, application *A*'s requirement that conflicts with enterprise goal *Z*, should be flagged for management consideration.

The ideal CASE should allow users to separate and integrate different applications easily. For instance, the company may want to document already operational applications and begin to manage them electronically. Users defining a new application may want to integrate it with an old application. They should be allowed to create an integrated third definition that highlights the overlaps, redundancies, inconsistencies, and other problems that the integrated pair have.

According to the 40-20-40 rule of systems development, 40% of project time is used for analysis and design, 20% is devoted to programming, and a full 40% is devoted to testing.[13] The current direction of vendors is to eliminate code, thereby cutting 20% off development time. But, the ideal CASE would cut the 40% devoted to testing as well. The urgency for CASE testing tools is low relative to other current concerns (like getting the products to work bug-free). At some point in the 1990s, vendors will begin to provide testing support in their CASE environments. Ideally, such support will include black- and white-box tests with human intervention allowed but not required. Black-box testing is for correctness of output based on inputs; white-box testing is for specific logic paths in a program. Intelligent software will analyze the type of process and determine the most appropriate testing strategy. Additional intelligent software will develop test data based on logical requirements, conduct the tests, and maintain test results. Test results will be integrated

13 Pressman [1987].

across test runs, phases of testing, versions of the software, and even hardware platform environments. When bugs are found, backtracking to find its source, possibly across modules, will be provided. Since the software built the bugs, it should be able to fix them; but, if the source is a logical, human specification, notice to the SEs will require correction of the errors.

Future products will eventually tackle the remaining 40% of project time by providing intelligence to identify reusable components of applications. Reusability of designs will have the most payback but is also the most difficult. Initially, reusable code modules will be enabled, then reusable designs, and finally, reusable logical analyses. Code reusability recognition should be available in CASE tools by the mid-1990s; the others will take until the turn of the century to surface.

This description of ideal CASE characteristics concentrates on what CASE should do rather than on what it currently does. For that, we discuss CASE as it supports each methodology and phase of development in the coming chapters. Although CASE and artificial intelligence (AI) are both in their infancy, the developments described above are currently feasible with current state-of-the-art technologies. The CASE repository will become the hub for all of the work that takes place in IS organizations. The limits to CASE intelligence that can be built are only due to human limitations.

SUMMARY

In this section preview, we identified the major activities of analysis and design. Analysis identifies *what* the application will do; design describes *how* the application will work in production. Both analysis and design have the same five generic activities: identification, elaboration, synthesis, review, and documentation. These activities are constrained and guided by a methodology. Each methodology takes a different perspective of an application leading to different phase-end results.

The organizational supports facilitate application development regardless of methodology. Organizational supports described in this chapter included joint requirements definition, joint application design, user-managed application development, data administration, and walk-throughs.

Software support that most facilitates application development is computer-aided software engineering (CASE). The ideal CASE environment has both expert and novice modes, can be customized for hybrid methodology use, and provides many additional intelligent functions beyond analyzing completeness of work. Future environments will identify reusable components of previous work to further reduce application development time.

The next six chapters discuss the analysis and design phases using the following example methodologies:

- Process—Structured Analysis (Chapter 7) and Design (Chapter 8)
- Data—Information Engineering—Business Area Analysis (Chapter 9) and Business System Design (Chapter 10)
- Object—Object-Oriented Analysis (Chapter 11) and Object-Oriented Design (Chapter 12).

Chapter 13 summarizes and compares the methodologies and their CASE support. Chapter 14 discusses forgotten activities of systems analysis and design.

REFERENCES

Blum, B., *Software Engineering: A Holistic View*. NY: Oxford University Press, 1992.

Conger, S., "The active dictionary in a CASE environment," *Data Base Management*, #25-01-20, NY: Auerbach Publishers, 1989, pp. 1–12.

Conger, S., and N. Russo, "A Taxonomy of Applications: A Framework for Selecting and Designing Methodologies," Georgia State University Working Paper #90-0201, 1990.

Couger, J. D., M. A. Colter, and R. W. Knapp, *Advanced System Development/Feasibility Techniques*. NY: John Wiley & Sons, 1982.

McClure, C., *CASE is Software Automation*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

McMenamin, S. M., and J. F. Palmer, *Essential Systems Analysis*. NY: Yourdon, Inc., 1984.

Pressman, R., *Software Engineering: A Practitioner's Approach*, 2nd ed. NY: McGraw-Hill, 1987.

Olle, T. W., J. Hagelstein, I. G. MacDonald, C. Rolland, H. G. Sol, F. J. M. Van Assche, and A. A. Verrijn-Stuart, *Information Systems Methodology: A Framework for Understanding*. Workingham, England: Addison-Wesley, 1988.

Swartout, W., and R. Balzer, "On the inevitable intertwining of specification and implementation," *Communications of the ACM*, Vol. 25, #7, July, 1982, pp. 438–440.

Wynckoop, J. L., and S. Conger [1991], "A review of computer-aided software engineering research methods," in *Information Systems Research: Contemporary Approaches and Emergent Traditions*, (H-E. Nissen, H. K. Klein, and R. Hirschheim, eds.). NY: North-Holland, 1991, pp. 301–326.

KEY TERMS

active data dictionary
analysis
CASE repository
compromise of requirements
computer-aided software engineering (CASE)
critical data
data administration (DA)
data architecture
database administration (DBA)
design
document
elaboration
facilitator
Fast-Track
identification
IS-managed application
joint application design (JAD)
joint requirements planning (JRP)
out of the box thinking
repository
review
synthesis
user-managed application
user manager
walk-through
work around

CHAPTER 7

PROCESS-ORIENTED ANALYSIS

INTRODUCTION

In this chapter, we review process-oriented analysis using structured analysis following DeMarco [1979], Yourdon [1989], and McMenamin and Palmer [1985]. Structured analysis was the first well-documented, and well-understood method of describing application problems. While the techniques have changed as our understanding and application types have changed, the techniques will remain useful for many years to come. This material should be a review, and for that reason, you might want to skim or skip it altogether. You might rate your knowledge by tracing the development of the ABC Rental Processing case. If you understand *and can reproduce* the work, skip the chapter.

CONCEPTUAL FOUNDATIONS

Structured analysis (and design) follow the architectural notion that "form ever follows function."[1] Functions of an information system are the processes that transform application data. Therefore, we emphasize processes and the flows of data into and out of those processes in **structured analysis**.

Structured analysis also is based on **systems theory** which assumes inputs are fed into processes to produce outputs. To complete the **systems model** (see Figure 7-1), there must be some sort of feedback to eliminate system entropy, that is, to keep the system from 'running down.'

To conceptually analyze complex systems as we have in IS, pieces of a problem are analyzed in isolation. We might look at inputs, outputs, and processes separately, then integrate them to produce a unified system. As system processing gets more complex, we study pieces of processes separately then integrate them. The pieces of the processes

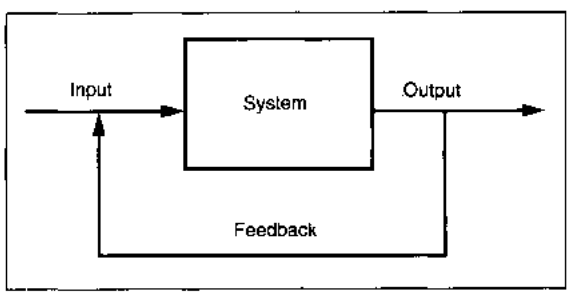


FIGURE 7-1 Systems Model

1 Sullivan, Louis, "The Tall Office Building Artistically Considered," *Lippencott's Magazine*, March, 1896.

must themselves be self-contained, small systems. These smaller systems comprise a hierarchy of system components, such that a component at any level is itself a system of components. Each *system*, regardless of level, has its own inputs, processes, outputs, and feedback. At the lowest level of the hierarchy are the **elementary components** which can no longer be subdivided and retain their system characteristics.

Structured development provides heuristics, guidelines, and diagram sets for dividing an information system into a hierarchy of logical component parts.

SUMMARY OF STRUCTURED SYSTEMS ANALYSIS TERMS

Structured analysis begins with two assumptions. First, we assume that we are most interested in what the application is *to do*. That is, what are its functions or processes? A **function** or **process** is some activity that transforms an input data flow into an output data flow. Second, we assume that we will treat the problem in a top-down manner. In top-down analysis, we analyze the external interfaces of the application first, then high level functions, and finally, lower level functions.

At the highest level, we define the scope of project activity. The **scope** defines the boundaries of the project: what is *in* the project and what is *outside* of the project. We document the scope of the project in a context diagram. A **context** defines a setting or environment. In structured systems analysis, the **context diagram** defines the interactions of the application with the external world. External world interactions occur between external entities and the application via the data flows that pass between them. An **external entity** is a person, place, or thing with which the application interacts, such as

Accounts Receivable Application
Citibank
Customer
Customer Service Department
Medicaid Processing Application
Medicaid Administration
The Federal Reserve Bank
The Internet (or other public network)
U.S. Internal Revenue Service

A **data flow** is data or information that is *in transit*. A data flow might be a piece of paper, a report, a diskette, or a computer message. Data flows in a diagram are directed arrows that depict data movement from one place to another.

A context diagram depicts the scope of the project, using circles, squares, and arrows. A large circle designates the application (see Figure 7-2). Squares identify external entities with which the application must interact. Directed lines (i.e., with arrows) are the data flows which indicate movement of data between entities and the application.

At the next lower level of analysis, we look inside the circle representing the application to define the major functions and files. Again, the functions are the major transformations triggered by input data flows to create output data flows. **Files** or **data stores** are relatively permanent collections of data. *Data flows* are distinct from *data stores* in their time orientation. Data flows are temporary and cease to exist once they are acted upon by a process. Data stores are persistent and maintained over time. Data stores may represent one or more data structures.

A **data flow diagram (DFD)** (see Figure 7-3) is a graphic representation of the application's component parts. Notice in Figure 7-3 that the entities and data flows from the context are all present. Also notice that data flows may connect processes to other processes, data stores, or external entities. Data stores and external entities do not interact directly with each other. If we compare the context to the data flow, we can perform quality assurance for completeness and consistency. *Completeness* checking ensures that *all* data flows and entities are included. *Consistency* checking ensures that *only* expected data flows and entities are included and that they are in the correct locations in the diagram set.

We do several iterations of DFD process analysis. At the highest level of analysis, the DFD is said to

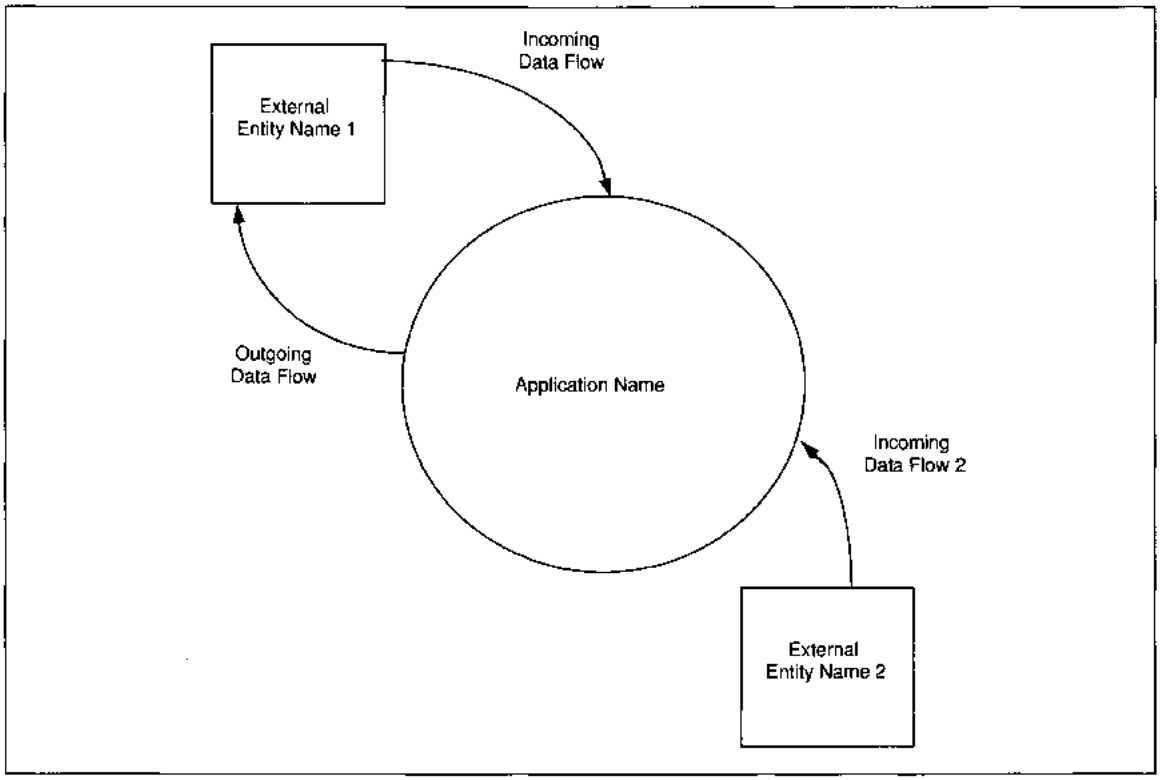


FIGURE 7-2 Sample Context Diagram

describe **Level 0** of the application. Each iteration is a *deeper* level of analysis to look into the processes from the previous level, analyzing the subprocesses, their constituent data flows, and their data stores. We link DFD levels through the process numbering scheme (see Figure 7-4). For example, process 1.0 from the level 0 diagram is decomposed into processes 1.1, 1.2, 1.3, and so on to describe the **Level 1** DFD. In Figure 7-4 Process 1.0 is decomposed into two subprocesses. Notice that a new file and an entity are other details added to the diagram. Level 1 DFDs may be further decomposed. To continue the example, process 1.1 might be decomposed into processes 1.1.1, 1.1.2, 1.1.3, and so on, until we reach the primitive, basic level. The **primitive level** is the level of each process at which no further decomposition can be done without fracturing the function. In other words, the decompositions at each level fully define the function, but may not define all of the functional details. At the primitive level, all files, flows, entities, and individual functions have been defined. There is no *right* level of definition; level is usually related to the type of application and target implementation language. You may do only two or three levels of decomposition for a nonprocedural, fourth generation language; you may do six or seven levels of decomposition for assembler or low level procedural languages (e.g., COBOL or Pascal).

The **structured decomposition** technique is a mechanism for coping with application complexity through the principal of 'divide and conquer.' A large, complex application problem is divided into its parts for individual analysis. Each part is further divided and individually analyzed. Complexity is reduced by allowing us to analyze small parts of the problem in isolation. The difficulties in structured decomposition are in correctly identifying the

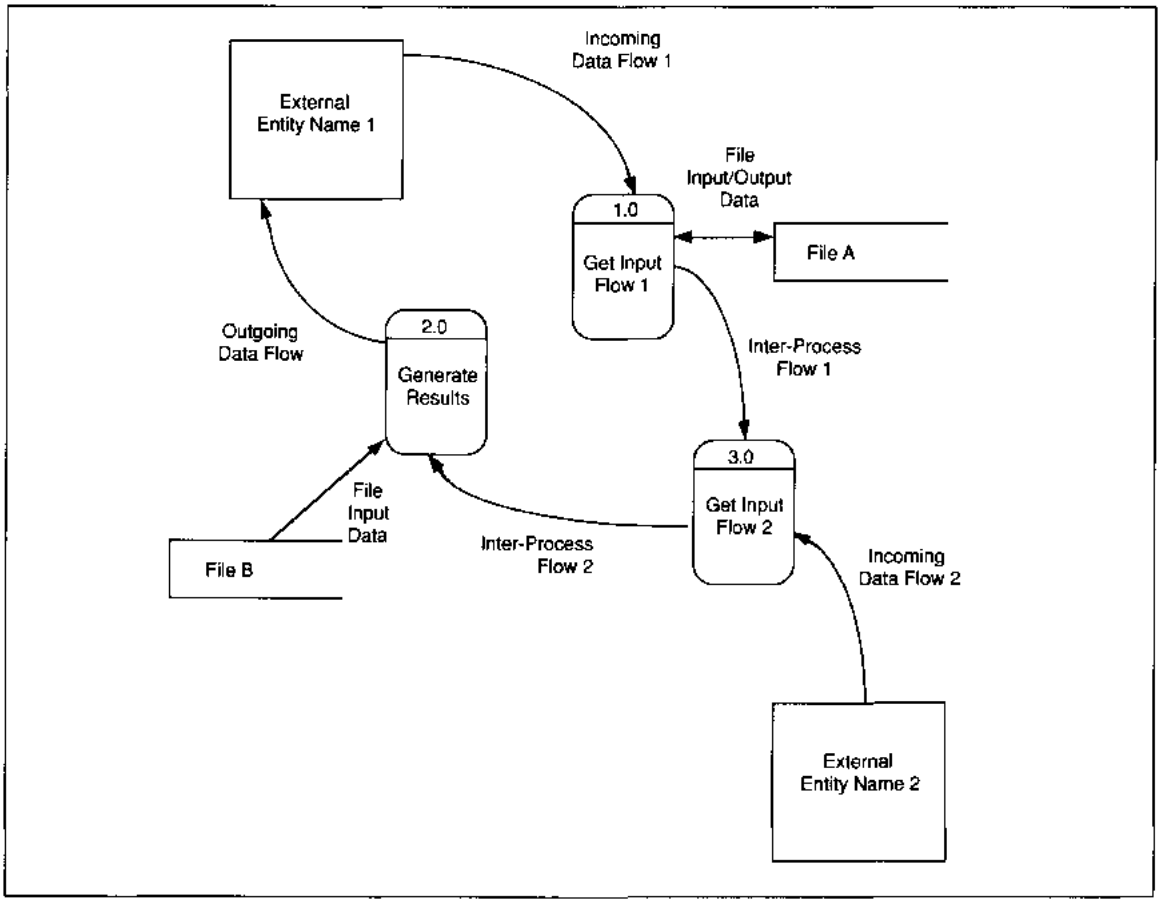


FIGURE 7-3 Sample Data Flow Diagram

isolated parts, and keeping the level of abstraction consistent.

After each analysis, the current level of DFDs is balanced with the previous level. **Balancing** is the act of checking entities, data flows, and processes across the levels of the diagram set. All entities and data flows from the higher level processes *must be* in every more detailed diagram. The names of entities and data flows must be consistent across the levels of the diagrams. We also balance processes. Lower level processes 'explain' or provide the details of higher level processes. Lower level processes are checked to be sure that they all relate to one, and only one, of the processes named at the higher level. They are then checked to be sure that they are in the diagram set for their related higher level process. When complete, processing is fully documented in a **leveled set of DFDs**.

While a set of balanced DFDs is being created, the secondary documentation is also being created. The secondary documentation includes creation of a data dictionary and optional graphics for real-time applications called state-transition diagrams. The **data dictionary**[2] compiles detailed definitions for each element in a DFD (see Figure 7-5 for contents for each entry type). The dictionary entries for processes contain details of how to accomplish the

2 See DeMarco [1979] and Yourdon [1989].

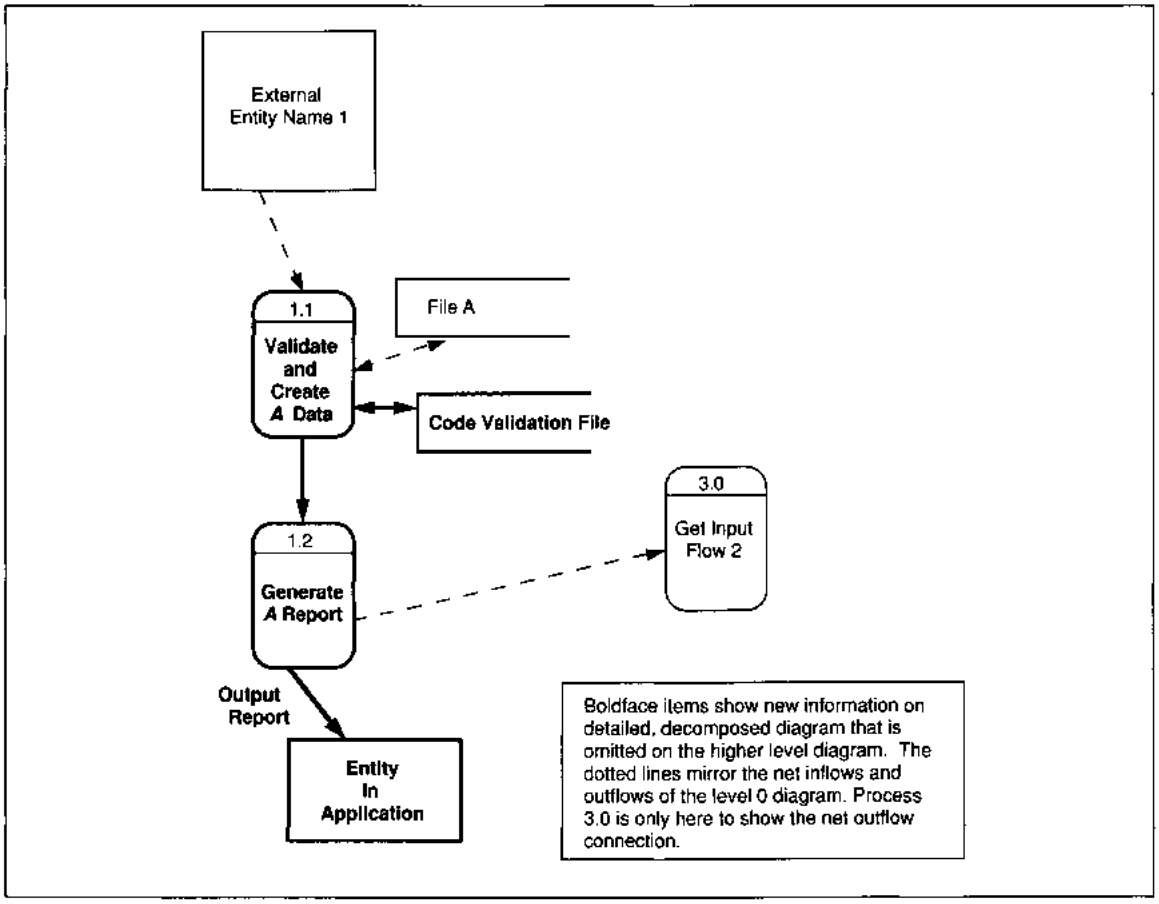


FIGURE 7-4 Example of Decomposed DFD

process. For instance, a process description for order creation might contain requirements for data entry, customer validation, item validation, order printing, and order filing. Since you get information on data piecemeal throughout the analysis (and design), it is easiest to document what you know as you go along. Surfacing assumptions, misconceptions, and data conflicts can be easier with this approach because the dictionary is always up to date with information and its source. If you collect pieces of paper and create the dictionary late in the analysis phase, identifying the source of conflicting information can be difficult.

Although not originally part of structured analysis, state-transition diagrams are frequently used to supplement DFDs in structured analysis for on-line (and real-time) applications. A **state-transition diagram** shows the time ordering of processes and identifies relationships between processes. State-transition diagrams are an integral part of object-oriented analysis and are deferred until that discussion in Chapter 11.

STRUCTURED SYSTEMS ANALYSIS ACTIVITIES

The specific activities in structured systems analysis are:

1. Develop a context diagram
2. Develop a set of balanced data flow diagrams
3. Develop a data dictionary
4. Optionally, develop a state-transition diagram if building an on-line or real-time application.

Data File or Data Base	File/Database Name Aliases Primary Key Alternate Keys Size of Relations/Records Growth Percentage per Year Security Data Structure Organization
Data Field or Attribute	User Name System Name Aliases Definition, if needed Creating Process(es) Length Data Type Allowable Values and Meanings Validation Method (e.g., cross-reference file, code check, etc.)
Data Flow	Name Aliases Timing (e.g., daily, weekly, as occurs, etc.) Contents Constraints (e.g., requires 5 second response; only occurs for sales orders, etc.)
Process	Process Name Process Number Description Constraints (e.g., must be complete within 20 seconds or Process *x* times out.)
External Entity	Entity Name Aliases Definition Relationship to Application Contact (if entity is an organization)

FIGURE 7-5 Data Dictionary Contents by Type of Entry

Structured analysis can be likened to a video camera with a zoom lens. At a distance, with no zoom, the item being examined is abstract and fuzzy. It has shape but no details. We can tell the photo is a building but little else (see Figure 7-6). When we draw a context diagram, we are examining the abstract shape of the item, in our case an application. Next, we zoom in with the camera to identify a greater level of detail about the object. In the photo, colors are distinct and some features of objects stand out. Pieces of the structure, for instance, columns, can be discussed in isolation of other pieces. Internal photos might show position, size, and type decor of rooms. There are still details which remain indistinct. When we develop the Level 0 diagram, we zoom in a level to expose more details of the problem. At this level, we describe the major normal processes, data flows, and files, and how they interrelate with external entities from the context.

In the third photo, we see all of the details: loose tiles on a roof, a crack in a foundation. Internal photos at the same level might detail construction materials (e.g., hardwood or concrete floors), and windows and doors to the outside. We can describe the context and surroundings, as well as the photo item, in as much detail as needed to suit our purpose. Similarly, at each additional level of application problem decomposition, we are zooming in to examine ever more detailed layers of the item, until we arrive at the essential processes in the application. At the lowest level of decomposition, we analyze not just the normal processing but all exceptions, errors, and details of reporting that accompany the normal processes. From systems theory, we know we are finished decomposing when we can no longer identify *minisystems* as the components of subprocesses.

The problem with the photographic zoom analogy is that the activities in structured analysis are not strictly top-down. First, we do not think in a strictly top-down manner. We jump back and forth between levels of detail to 'test' how a higher level decision might look at a lower level, to get details of a new process so we are sure how it 'fits' with the other processes, and so on. When we are developing an application similar to something we have already done, we have a good understanding of familiar parts and little understanding of new parts. We spend time

FIGURE 7-6 Zoom Analogy to Structured Analysis

analyzing the new parts of the application to see how they fit with what we already know. We have to change our preconceptions based on the new information, and alter our 'mental model' at all levels of detail to accommodate the new information. We may go into great detail on a new aspect of the application, ignoring the known aspects temporarily. Then, when we understand the new parts, we can go back up to a high level of abstraction to document how the parts fit together.

Second, application analysis is iterative. We have already discussed planned iterations to move to lower levels of detail in documentation. We also reiterate through analysis when we find some unexpected, unknown, or changed requirement to ensure that it fits what we already know. To decide that fit, we must *walk-through* the entire process top to bottom. Recall that a **walk-through** is a formal review of analysis, design, program code, test design, or some other component of application development work. A walk-through can be used to determine

where the new requirement fits
what other processes, flows, stores, or entities are involved in the change
what are the ripple effects of the change through the set of DFDs.

Another analogy for structured analysis, as equally applicable as the photo zoom, is from geology (see Figure 7-7[3]). If we are trying to drill for oil, we might find a variety of different formations and even have different drilling results, depending on the depth and angle. So, too, in structured analysis our results depend on our approach and the information we obtain from interviews and information gathering. The information differs for each user because their perspective of the problem, their job goals, and their personal aspirations all distort their view. We require multiple approaches, multiple interviews with both the same and different people, and multiple perspectives of analyzing the information. Figure 7-6 shows unfocused probing. The pieces and views do not fit together. We know we are at the end of analysis when all users agree *and* all the disparate

3 This analogy is from Gary Moore, University of Calgary, who originally used it to describe research in information systems. It fits the application development context as well.

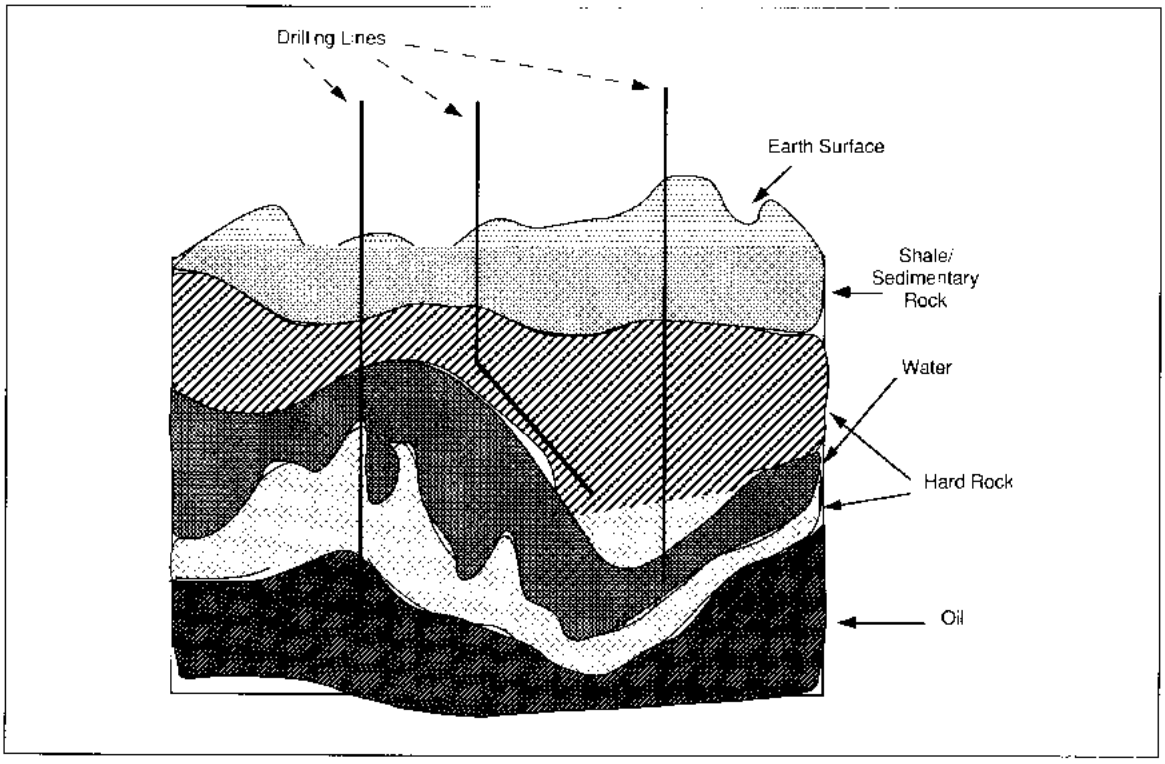


FIGURE 7-7 Geologic Analogy to Structured Analysis

views fit together coherently. Recall that triangulation is a data gathering technique comparing multiple verifying sources of all information. The purpose of triangulation is to ensure that *our* resulting view of an application accurately depicts the requirements of the work process it supports. So, we analyze top-down, sideways-out, bottom-up, *and* do them all more than once in the analysis process.

Now, we turn to the discussion of *how* to actually develop the documentation in structured analysis.

Develop Context Diagram

Rules for Developing Context Diagram

The context diagram summarizes the scope of the project. The rules for developing the context diagram are listed below for easy reference.

1. Define the boundaries (i.e., scope) of the application. Specifically, define what the application will do *and* what it will not do. Draw the circle identifying the application and write the application name in the center.
2. Using the application boundary as a starting point, identify all external entities with which the application must interact. For each entity, draw one square on the diagram and label the square.
3. For each entity, create a definition in the data dictionary.
4. For each external entity, identify the specific data flows that define the interface.
5. For each data flow, create a definition and list of tentative contents in the data dictionary.

Scoping may take place before analysis actually occurs and is usually part of the feasibility study as

discussed in Chapter 6. Some organizations which might not perform feasibility analysis still require a bounding of the application. Review that portion of Chapter 6 if you do not remember the political and organizational issues involved. Here, we assume that boundaries are defined and that the application and its interfaces to external entities are reasonably well defined.

Definition of external entities is next. External entities are people, places, or things which interact with the application. Usually, we identify titles/roles (e.g., Customer), departments (e.g., Accounts Receivable), organizations (e.g., Medicare Administration), or applications (e.g., Accounts Receivable Application) as entities. The phrase 'interact with the application' has a very specific meaning. The entity is *outside* the control and/or processing being modeled for the current application. That is, external entity processing, procedures, and data are *not* subject to analysis or change. Relationships between external entities are *not* shown on the diagram(s) (i.e., external entities cannot connect to each other). For example, if you are modeling an order processing application that does not do inventory control, the warehouse would be on the context diagram. If inventory control and warehouse processing are within the scope of the application, the warehouse would not be on the context diagram.

After entities are identified and drawn on the diagram, they should be defined in the data dictionary. The entries for an entity include a name and definition (see sample Figure 7-8). This step is important for two reasons: to develop a common vocabulary, and to develop documentation as analysis proceeds. Frequently, individuals might believe they have a common vocabulary because they use the same words in their discussions. Only when they develop a common definition of the terms can they be sure that their shared terminology also means they share the meaning of the terms (see Example 7-1). Finally, in organizations having a data administration function, a dictionary (or repository) of 'corporate' data is an integral part of the organization's data architecture (see Chapters 9 and 10 for more on this topic). The name and definition of each entity (and, eventually, each attribute) should be matched against the organizational definitions to ensure consistency with other uses of the same name, or uniqueness of the name if a new definition is developed.

Entity Name	Customer
Aliases	None
Definition	A company, government agency, nonprofit organization, or individual who orders goods and services from *X* Company
Relationship to Application	Order goods, return goods, receive invoice
Contact, if entity is an organization	None
Entity Name	Medicaid Administration
Aliases	Medicaid
Relationship to Application	Receives claims, sends claim reconciliation, payment
Contact, if entity is an organization	Mary Jones 202-445-0011, NY State Claims Adjustor Medicaid Administration 1401 Avenue C, NE Washington, D.C. 01010

FIGURE 7-8 Example of External Entity Description

There are several reasons for documenting definitions in the dictionary as work proceeds. First, the dictionary provides the basis for intraproject communication. Whenever a definition is developed and added to the dictionary, the more the team builds a shared view of the application reflecting the dictionary contents. Second, documentation is best done as the project progresses to ensure that it gets done. If documentation is delayed until after implementation, it rarely includes the wealth of detail and history of decisions that can be incorporated if done instream.

The next action in developing the context diagram is to define data flows between the application and each external entity. The questions you ask yourself to identify data flows are, "What *information* do I (as the application) need from this entity?" and "What *information* do I feedback or provide to this entity?" Frequently, but not always, input flows (to

EXAMPLE 7-1

A Case of No Shared Meaning

The XYZ Annuity Company was developing a new application to define the institutions which defined its customer base. The exercise was prompted partially by a lament from the head of marketing who claimed, "There are 6,400, 7,500, or 9,650 institutions, depending on who I ask and which application they are getting the numbers from. Can't I have *one* number of institutions?"

A newly founded Data Administration team decided that the first "corporate" definition they would tackle was institution. The data analyst assigned first asked *application developer* colleagues, "What is an institution?"

The replies were varied and generally, unsatisfactory:

Anyone we do business with.
An organization we do business with.
Any legal entity we do business with.
A school, research and development institution, not-for-profit foundation, or other organization which is approved by the IRS to contract for annuity business with XYZ Annuity.
An organization that has a plan defining a group of annuity contracts.

Then the analyst asked the *users*, "What is an institution?"

Some organization that remits annuity payments (a remittance clerk's definition)
An organization with a plan defining a group of contracts (a accounting manager's attempt at a generic definition)
An approved organization which may or may not have a contract plan (a marketing definition)
An organization to whom annuity and pension product counseling is provided (a counselor's definition)
A target audience for marketing and selling annuity products (a marketing definition)

The analyst then asked the *senior manager* in charge of institutional relations to please define an institution. His response was a three-page, single-spaced memo that defined six major variants and over 30 different situational definitions for an institution.

Two important ideas here are, first, *all of these definitions are correct*, and second, *each definition has some generally accepted component*. Definitions relate to perspective. A systems person defines an institution in relation to the application's use of the term. A user defines the term in relation to their job's use of the term. The manager tried to synthesize all perspectives and highlighted the variation and divergence that had evolved throughout the organization. Third, *all of these definitions have some element that appears important to defining "institution."*

When asked about the differences in the definitions, one user said, "Oh, yes, we know we don't all mean the same thing when we use the term institution. I even mean different things depending on the topic."

Resolution of the differences took over six months of part-time work, resulted in the definition of 20 new attributes of an institution, and required the approval of 72 managers in the process. Several applications under development that were using an institutional billing code as the primary key identifier underwent substantial redefinition as a result of the development of a shared term, 'institution.'

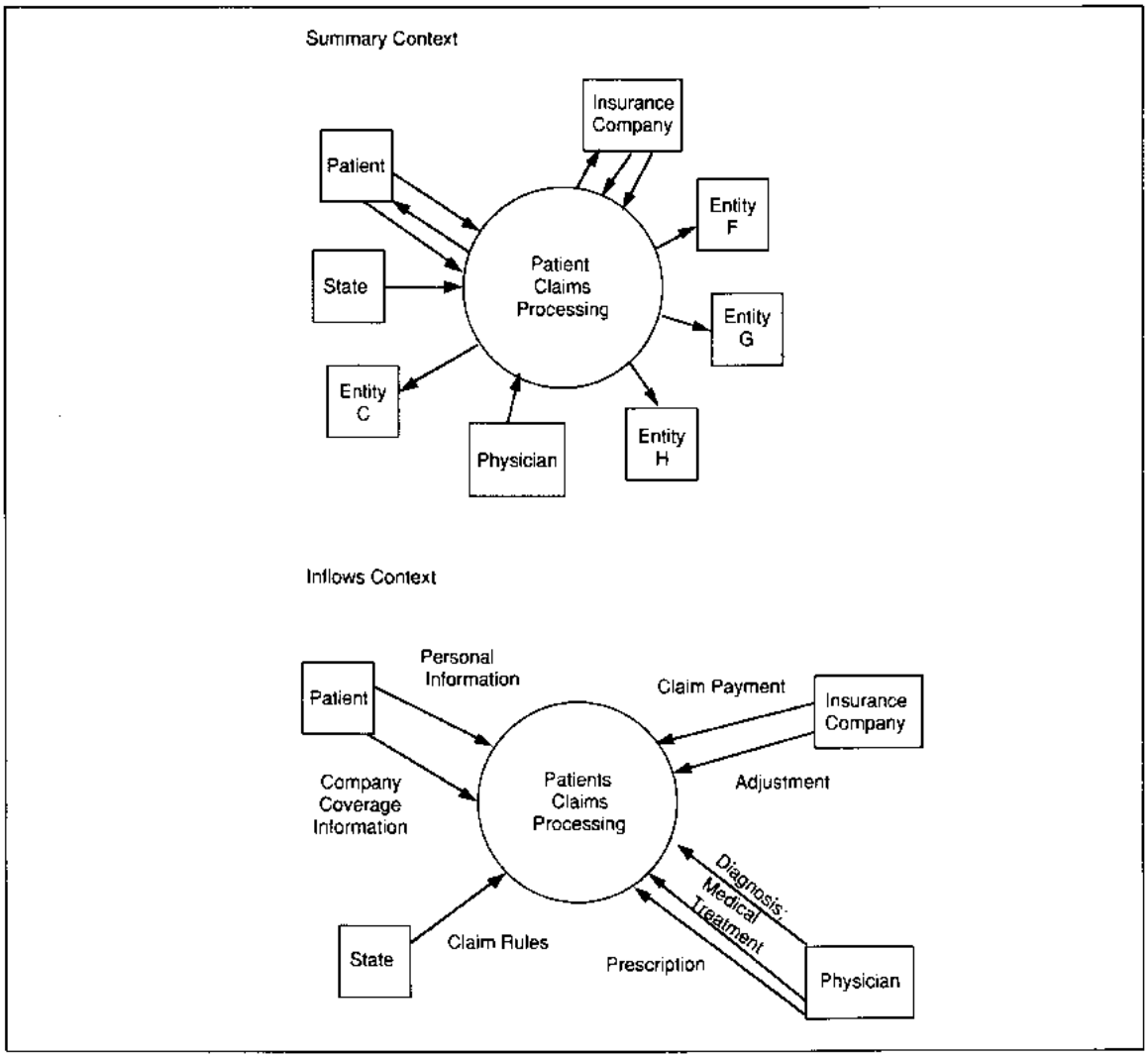


FIGURE 7-9 Example of Complex Context Diagram

the application) are matched with output flows to the same entity. For instance, customers place orders; the application sends an invoice (and goods) back to the customer. Check for reciprocating input-output flows such as these. When you identify single flows to/from an entity, you want to double check by asking, "How do I know they got this output?" or "Do I have to tell them I got this input?" As you define each data flow, draw the directed arrow on the context diagram, and label the flow. For a complex application, you might need two levels of context diagrams (see Figure 7-9). One level summarizes *all entities* with directed arrows that are unlabeled. The other level shows *input flows* on one diagram, and *output flows* on the other diagram with labeled data flows on both diagrams.

Data flows are *information* about some business event being tracked by the application. They do not identify physical items. For example, an invoice is information about an order that would also have

Name	Order
Aliases	None
Timing	As Occurs
Contents	Customer Name + [Address | Customer ID] + Shipping Instructions + 1{Item name + (Item number) + (Color) + (Size) + Quantity ordered}m
Constraints	80% must be billed and shipped within 24 hours 100% orders in by noon must be billed and shipped the same day

FIGURE 7-10 Example of Data Flow Dictionary Description

actual goods. A data flow to a customer shows the invoice but not the physical goods.

Last, for each data flow, create a definition in the data dictionary. The dictionary information provided for a data flow is its name, contents, and contents' source when it is not obvious (see Figure 7-10 for sample data flow description).

ABC Video Example Context Diagram

The scope of the project for ABC Rental Processing system is to provide rental/return processing for videos, including customer maintenance, video inventory maintenance, historical information maintenance, and reports to management. At the end of the day, accounting totals of sales information are generated, but there is no automated accounting interface. There is no purchase order processing in this application. The application's main function is rental processing, so we will call it 'ABC Rental Processing.' We draw the circle for the application in the context diagram and label it '*ABC Rental Processing.*'[4]

Then we define the entities. Possible entities are customer, video vendor, ABC management, ABC accountants, and the Internal Revenue Service (IRS). The IRS is omitted because there is no tax-related processing performed in the application. *ABC accountants* are included because they receive an end-of-day report of receipts. How management and/or accountants use that information is beyond the scope of the application. ABC management could conceivably be on the diagram. Now, we ask ourselves, "Do we have control over what ABC management does with respect to the Order Processing application?" The answer, *in this case*, is yes, because ABC is so small. In other circumstances, the answer could be no. For instance, with a large application generating reports for many levels of management or for other departments' management, the answer might be no. Here, ABC management is not on the context diagram; in other companies or contexts it might be.

The entities left are Customers, Video Vendors, and video. *Customers* should be obviously correct. All rental and return processing relate to interactions of the application with customers. ABC has no control over customers' rental choices.

Vendors as an entity might be less obvious. Even though there is no automated purchase order process, the videos entered into the application come from somewhere, so video *vendors* should be identified as the source of video information.

Last, we deal with video. Is video an entity that the application interacts with? The answer is yes. Is video an entity that the application can control? The answer is again yes. Video is *not* on the context diagram because it is *in* the application. In effect, the video is *within the circle* that describes the ABC Rental Processing.

As a result of this analysis, we add three external entity squares to the context diagram labeled *Customer*, *Video*, *Vendor*, and *Accountant* (see Figure 7-11), and define the entities in the dictionary (see Figure 7-12).

Next, we define the data flows and document them in the dictionary. What happens in this application? When a customer selects a video, they first tell the clerk their phone number. The clerk uses the phone number to 'look up' the customer and validate their rentals. If the customer is new (i.e., not on file),

4 The names of items from a diagram are in italics to set them off from the rest of the discussion and, hopefully, minimize your confusion.

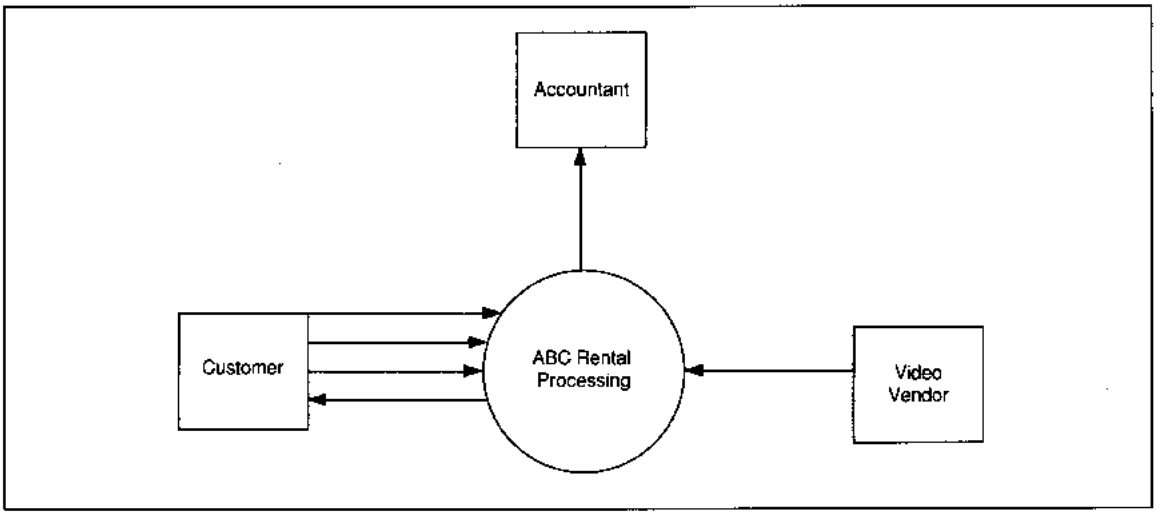


FIGURE 7-11 Skeleton ABC Rental Processing Context Diagram

the customer information is entered and stored. After phone number processing, the customer either gives the clerk the cardboard shell, or tells the clerk the video name (see Chapter 2). This sentence identifies a data flow: *rental request*. After entering the information into the computer system, the clerk needs to provide some record with customer signature that the rental took place. This record accounts for the transaction and establishes customer liability for the rental property. This information identifies a reciprocating outward flow to the customer: *rental receipt*. When the tape is returned, the charges are computed based on the due date of the rental(s). This identifies another incoming data flow for a *video return*. So we have identified four data flows between the ABC Order processing application and customers:

- *New Customer* to store customer information
- *Rental request* (analogous to placing an order) from the customer to create a video rental and payments
- *Rental Receipt* from the application to confirm the rental
- *Video Return* to determine late charges, if any, and payment due.

For these four flows, there are four arrows between *customer* and ABC Rental Processing. Three arrows are *from customer* for *new customers*, *rental requests*, and *returns*. One arrow is *to customer* for the *rental receipt*.

Entity Name	Customer
Aliases	None
Relationship to Application	Rents and Pays for Videos, Provides New Customer Information, Returns Videos
Contact	None
Entity Name	Video Vendor
Aliases	Vendor
Relationship to Application	Provides New Videos
Contact	None
Entity Name	Accountant
Aliases	None
Relationship to Application	Part-time employee receives end-of-day reports
Contact	None

FIGURE 7-12 ABC Rental Processing Data Definitions for External Entities

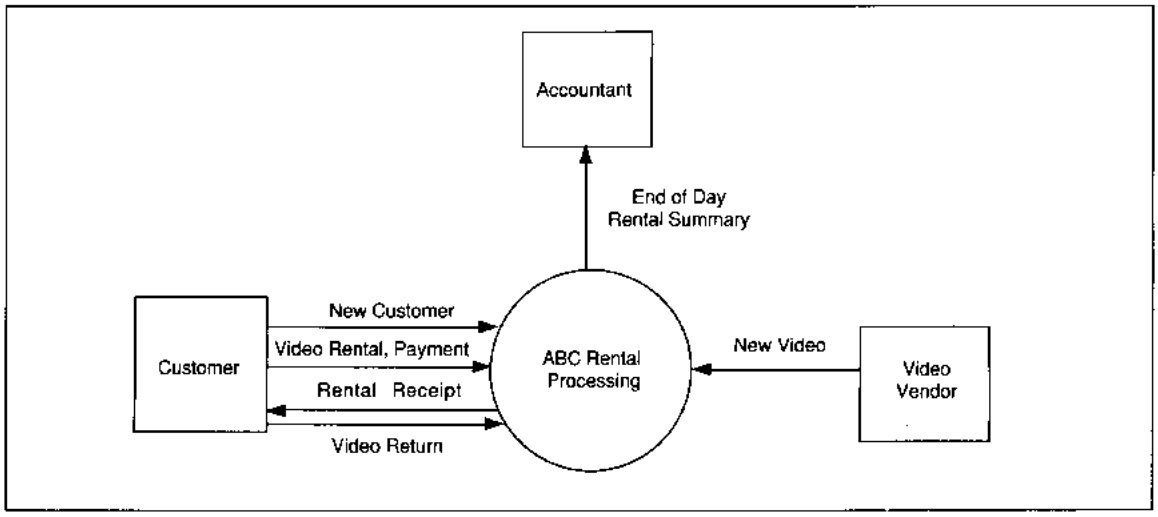


FIGURE 7-13 ABC Order Processing Context Diagram

The data flow relating to *vendors* is somewhat obscure, but is identified by the need to enter new video information. Since new video information comes from somewhere, its source must be identified as the entity. There is one data flow *from vendor* to ABC Rental Processing for *video information*. There are no data flows back *to vendor* because the scope does not include ordering videos from the vendor.

Last, we define the data flows to and from the accountant. The accountant does not feed any information into the application, and receives only an end-of-day rental summary. So, there is one data flow *to* accountant for the '*end-of-day rental summary*.' Next, we draw the data flows on the context diagram and label them (see Figure 7-13).

While we label the flows, we evaluate the names of the data flows to ensure their meaningfulness. *Rental request* implies a request for assistance in rental processing and is a weak name. Stronger, more meaningful names are '*Video rental*' or '*Video rental information*.' Either of these might be used. Here, we use *Video rental* since the word 'information' is not particularly meaningful. Also, rentals are always accompanied by payments which are added to the name to be more explicit.

Novice analysts frequently have trouble differentiating between the *thing*, and information *about the thing*. Keep in mind that what we document on DFDs is always information *about* the thing. So, when we name a data flow '*Video Rental*' we really mean *information about* '*Video Rental*.' That is why the word 'information' is weak in the data flow name. The other names: *Rental Receipt*, *Video Return*, *New Customer*, *New Video* (*not* New Video Information), and *End-of-day Rental Summary* are all acceptable. Again, there are no 'right' or 'wrong' names for data flows. Some names are more descriptive than others, and, therefore, stronger. Many companies define their own conventions, or local rules, for naming data flows, entities, and processes.

Last, we define data flows in the dictionary (see Figure 7-14). Keep in mind that just because the information is in the dictionary does not mean it is cast in concrete. It is subject to review and change throughout the life of the project. The goal is to define the application at a level of detail so that changes can be made *before* they become costly, that is, during analysis.

Upon completion of the context diagram, you are ready to do the next level of analysis, opening up the circle, to define a data flow diagram.

Name	New Customer
Aliases	None
Timing	As Occurs
Contents	Name + Address + Phone Number + Credit Card Type + Credit Card Number + Credit Card Expiration Date
Constraints	None

Name	Rental, Payment
Aliases	None
Timing	As Occurs
Contents	Phone Number + 1{Video ID}m + Total Amount of Order
Constraints	None

Name	New Video
Aliases	None
Timing	As Occurs
Contents	Video ID + Video Name + Date + Rental Price
Constraints	None

Name	Copy of Order
Aliases	Printed Order
Timing	One per rental transaction
Contents	Phone Number + Customer Name + Customer Address + 1{Video ID + Video Name + Rental Charge + Due Date}m + Total Amount + Total Amount Paid + Total Amount Due (should be zero)
Constraints	Must be signed by customer. Optional that customer takes a copy.

Name	End of Day Summary
Aliases	EOD Rental Summary
Timing	Close of Business
Contents	Videos Rented + Total Fees Collected + Videos Returned + On-Time Returns + Late Returns + Total Late Days + Late Fees Collected
Constraints	None

FIGURE 7-14 ABC Video Data Flow Definitions—Tentative

Develop Data Flow Diagram

Rules for Developing a Data Flow Diagram

To develop a data flow diagram, iterate through the following steps until a primitive level is reached:

1. Define the processes.
2. Define the files and other data flows required to support the processes.
3. Draw a Level 0 DFD. At level 0, ignore trivial error paths and data stores. If you define a validation process, you must eventually identify an error path. Define the error path at the primitive level. Similarly for data stores, define files when they are shared between processes. Introduce files that are only used within a given process at the level at which the file is shared between two or more subprocesses.
4. Balance the DFD with the context diagram. Compare the net inputs and outputs to external entities on the DFD to the net inputs and outputs on the context diagram. There should be a one-to-one correspondence between the diagrams.
5. Iterate through this procedure until the primitive level of DFD is reached for all processes.

> Always balance the current level DFD's net inputs and outputs with those of the previous level.

First, we will discuss how to identify the Level 0 processes that are within the circle of the context diagram, without defining any data stores. The difficulty of this activity varies with your understanding of the problem domain and the scope of the project. One of the hardest parts of this activity is to decide the 'right' level of abstraction. What is right in one instance may not be right in another. For instance, if you have a multidepartmental, multiapplication environment you are trying to describe, the Level 0 diagram might link departments and the net data flows of the context diagram (see Figure 7-15). If you have a multidepartmental, single application environment, you might identify major functions and their relationships (see Figure 7-16). Or, if you have a single department, single function application, such as ABC Rental Processing, you try to define the general functions to be performed. The approach in this text is to define the simple environment, discussing the common features for all levels of abstraction.

During the information gathering stage of the application, you discussed with users what they did and how they did it (see Chapter 4). The individual steps that each user performs in the tasks relating to the application are components of the applications' processes. There are a variety of ways to identify processes; some examples are:

1. **Direct identification:** If you have similar experience and either know the processes, or have articulate users who know the processes, identify them directly.
2. **Top-down:** Decompose the problem into its constituent parts. The functions at each level should completely define the problem and

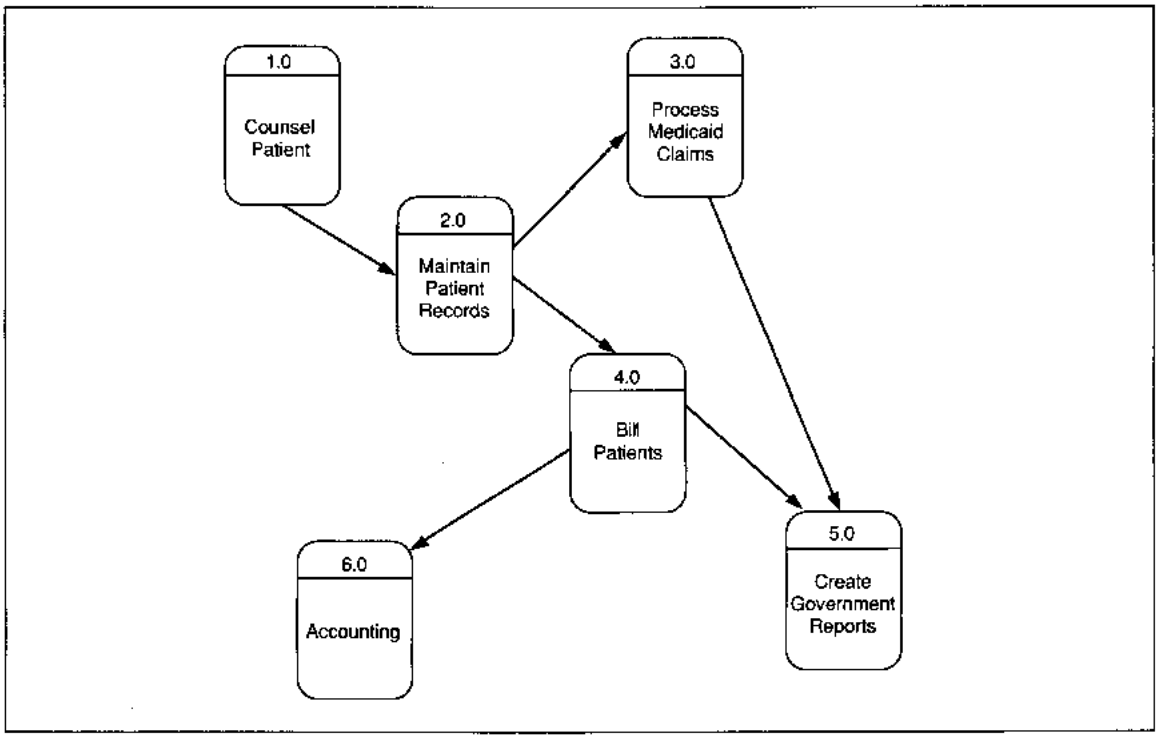


FIGURE 7-15 Multidepartment, Multiapplication Level 0 DFD

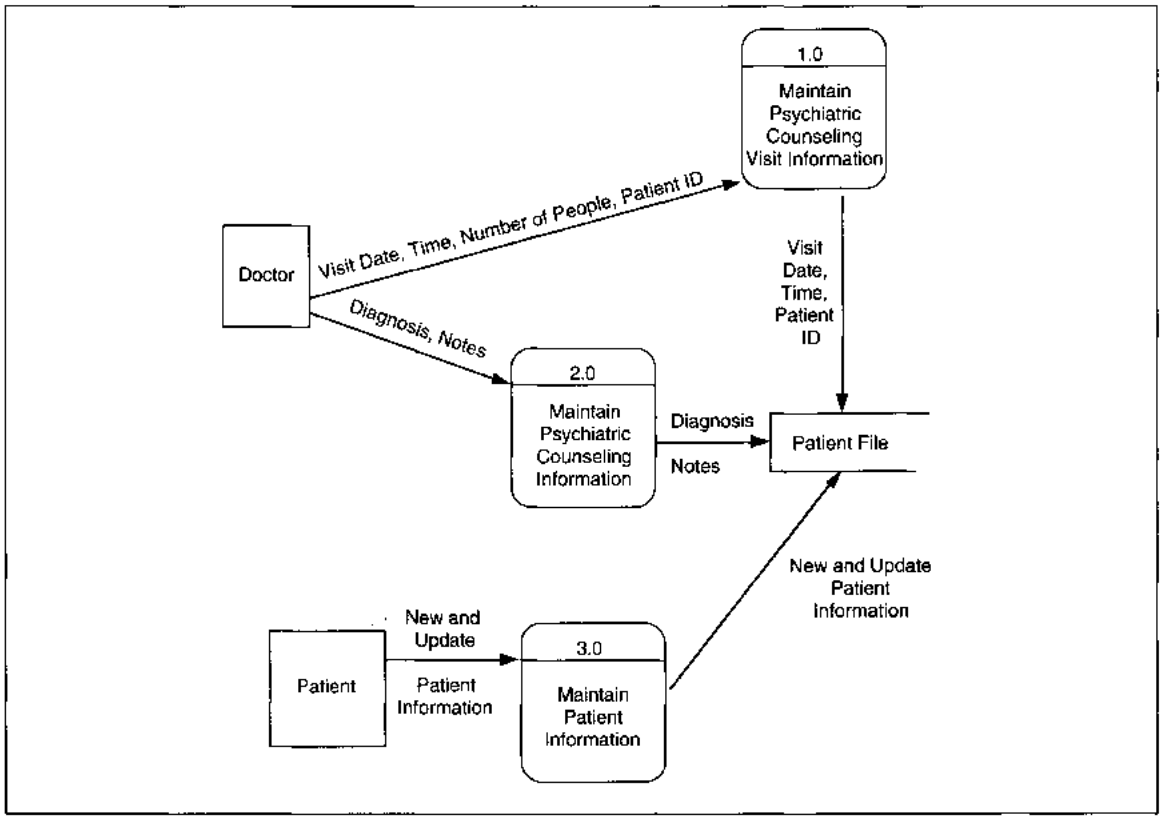


FIGURE 7-16 Multidepartment, Single-Application Level 0 DFD to Maintain Patient Records

should be as independent of each other as possible. The resulting independent functions can be analyzed in isolation of the other parts to develop each part's subprocesses. Decomposition continues until atomic levels of processing are identified.

3. **Bottom-up:** Do bottom-up analysis starting with the details of task steps and procedures described by users, synthesizing and combining the steps to define processes.
4. **Outward-in:** Use context diagram entities and data flows to identify 'boundary' processes with which they directly interact. Work outward-in to define what other transformations are required to link the input and output boundary processes.
5. **Functional sequence:** Examine the input data flows from external entities to identify the 'first process' in a sequence of processes. From that first process, define the other transformations that are required to go through each function from beginning to end.

All of these approaches can work. None is more right than another. We all use one or more of these in performing analysis without thinking about how we actually do it. A good approach is to use two or three of the methods as a way of double-checking that all processes are defined and connected properly. For ABC, we will combine the last two approaches, using the information from the context diagram.

Once processes are identified, you draw them and connect them to the external entities via the named data flows. Other data flows and processes are identified to connect the initial ones defined until you feel

the diagram completely describes the overall processing. Keep in mind while you are performing this activity that you do not pay attention to timing or sequencing of processes. You do not show start-up or shutdown activities on a data flow. If you have end of day, end of month, or other periodic processing, the DFD shows the processes without necessarily identifying the timing of the processing. As the processes are drawn, name each with a verb and the data they create, and number them. Numbering of processes is not meant to sequence them, even though we unconsciously tend to do this.

Also, at Level 0, ignore exception processing. You might have a data flow named '*Valid X*' without a matching '*Invalid X*.' The exception process is added at the next lower level. This avoids unnecessary clutter at the highest level.

After the processes are identified, next define file locations on the Level 0 data flow diagram. You could leave files for a lower level of analysis as many texts and companies do by convention. In that case, you are ready to draw the diagram. Here, we will develop the thoughts that are used to identify data stores.

To identify data stores, first consider each process. Can the process be completed without reading or writing to a data store? If your answer is yes, then you do not need a file at this level. If the answer is no, you need one data store for every required read action and every required write action. Many times, the reads and writes are to the same data store. Then, you have one data flow per input/output action. As these required reads and writes are identified, you add to the DFD to include the data store name and data flow(s). When you do this part of the drawing, make sure that each flow and store has a name.

Finally, when you have reviewed each process for determining whether to include data stores, review the diagram to make sure that its DFD syntax conforms to the rules. The first seven rules relate only to processes and their connectivity. Processes with connection errors are called 'pathological' processes because they do not follow the philosophy of DFDs that processes are connected via flows, files (data stores), or entities.

The next four rules check that all connections in the diagram are legal. The rule about no dangling arrows[5] is our own. Work and teaching experience have proven that novices use dangling arrows to hide their lack of understanding of what they are doing. The final two rules deal with balancing, error handling, and the introduction of files.

The DFD syntax rules are:

1. *All* processes are *connected* to something else.
2. *All* process have *both inputs and outputs*.
3. *No* processes have only outputs or only inputs.
4. Processes may connect to anything: other processes, data stores, or entities.
5. All processes have a unique name and number.
6. Each process number is used once in the diagram set.
7. Only subprocesses of a process shall follow the numbering scheme of the parent process.
8. Entities and data stores may connect *only* to processes. Another way to state this is that each data flow must have at least one end connected to a process.
9. Data flows are the only legal type of connection between entities, processes, and data stores.
10. Make sure there are no dangling arrows.
11. The net data flows to and from context diagram external entities *must* balance, that is, be present, in each level of DFDs.
12. Trivial errors and exceptions are not handled until L1 or lower in the DFD set.
13. Trivial data stores show up in the diagram set the first time they are referenced by a process.

When the Level 0 DFD is complete, walk through the DFD with your peers, then review it with your user. Keep in mind that you are teaching the users

5 I realize that this is contrary to DeMarco, Yourdon, and many undergraduate texts. For novices, dangling arrows frequently mean you have no clue about what attaches at the other end. In addition, most companies want all terminators identified to ensure accuracy and to simplify quality assurance. Until you are proficient, draw the *entire* diagram!

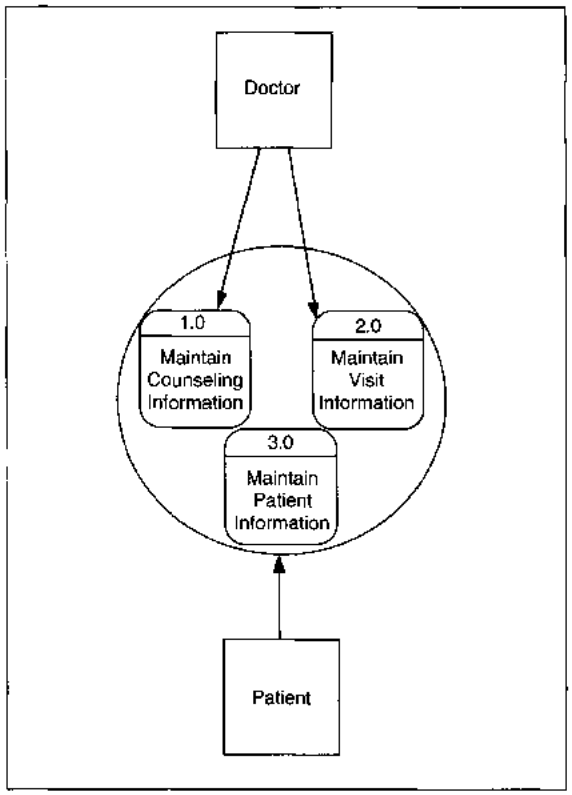


FIGURE 7-17 Context Expansion of Level 0 Processes: Maintain Patient Records

as well as having them review your work. If they do not understand what you are showing them, they cannot adequately comment on it. So, use a top-down approach to the presentation, too. First, show the users the context diagram. Define all of the items in the diagram. Once they agree on the external entities, show them a blowup of the context diagram that includes the inside of the circle: the major processes and the data flows connecting them to external entities (see Figure 7-17). Then, replace that diagram with a Level 0 DFD showing the entities and processes. Use overlays, adding the data stores and remaining data flows. Finally, review the detailed definitions from the data dictionary for each process, data flow, data store, and entity. If you take a step-by-step approach, users can more easily accept and assimilate the information.

Do not expect to have agreement on the first, or even second, review. One benefit of data flow diagrams is focusing thoughts on the problem. Users will frequently 'see' what is missing when they look at a diagram that they could not 'see' when they discussed the topic verbally. When they begin a sentence, "Well, what about . . ." pay close attention; the subject is usually some variation, exception, or forgotten information that they did not discuss previously.

As you understand and users agree on the context and Level 0 processes (see Figure 7-18), begin work on the lower level DFDs. For each Level 0 process,

1. Draw the input and output flows and the icons to which they connect from the higher level diagram. This forms the skeleton of the diagram (see Figure 7-19). These are called the **net[6] inflows and outflows.**
2. Define the subprocesses by asking, "What are the steps required to *do* this process?" Then for each step, "Can I separate this from the other steps and do it in isolation?" For each *subprocess* you *isolate*, draw a process rectangle on the lower level diagram.
3. Identify whether data stores are required or not. Add them and, if they are new, name them.
4. Identify data flows to complete the diagram (see Figure 7-19). Make sure you provide *all and only* the information required to perform the process.
5. Review the diagram for unnecessary connections and, if found, remove them.
6. Update the data dictionary with all new information.

The goal of subprocess identification is to decompose the upper level processes into what will eventually be programmable modules. A good, that is, correct, design has certain characteristics that are

6 Net, from accounting, means remaining after all necessary deductions. Here, net means remaining data flow and data store connections after the higher level process is removed. The net data flows in and out of a higher level process may connect to different subprocesses at the lower level.

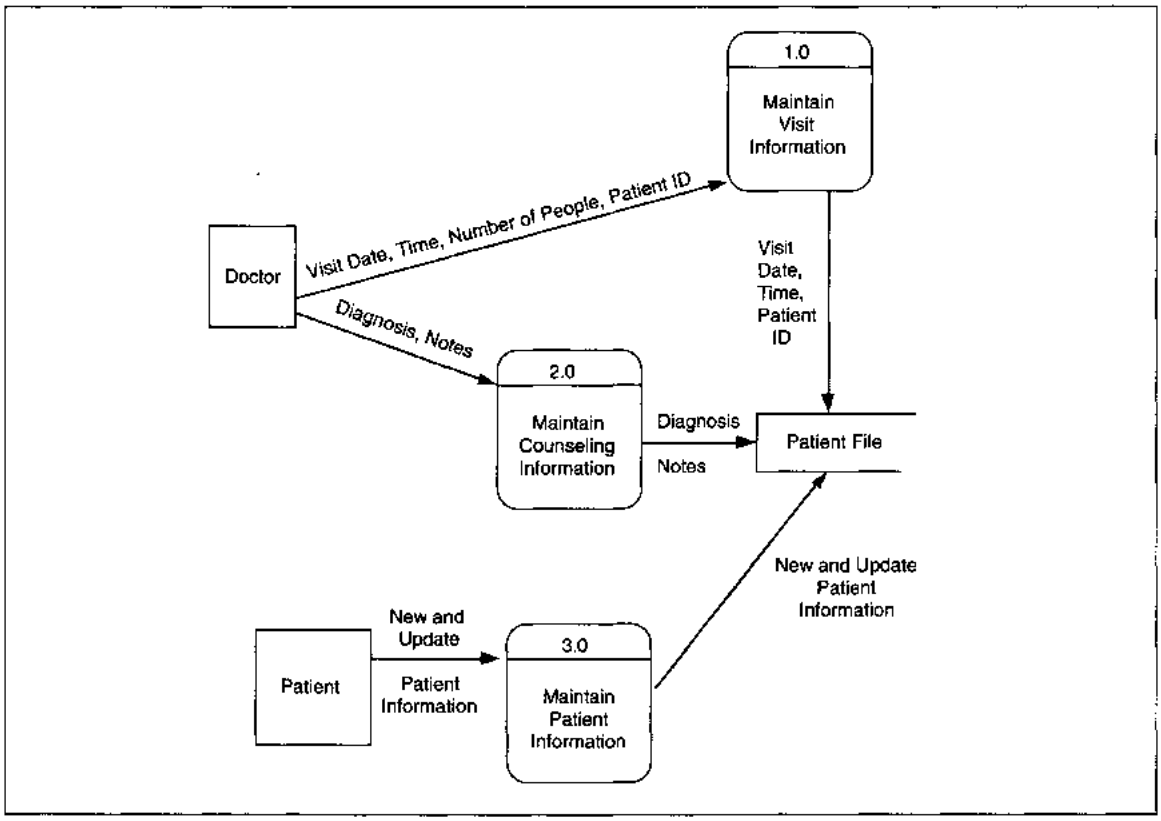


FIGURE 7-18 Completed Level 0 DFD to Maintain Patient Records

traceable back to a properly decomposed DFD. The two most important characteristics are maximal cohesion and minimal coupling. **Cohesion** measures the *internal* strength of a process (this is also called intraprocess strength). We want modules that result from process descriptions to have exactly the logic required to perform the task, and nothing more. Minimal **coupling** measures the *interprocess* connec-

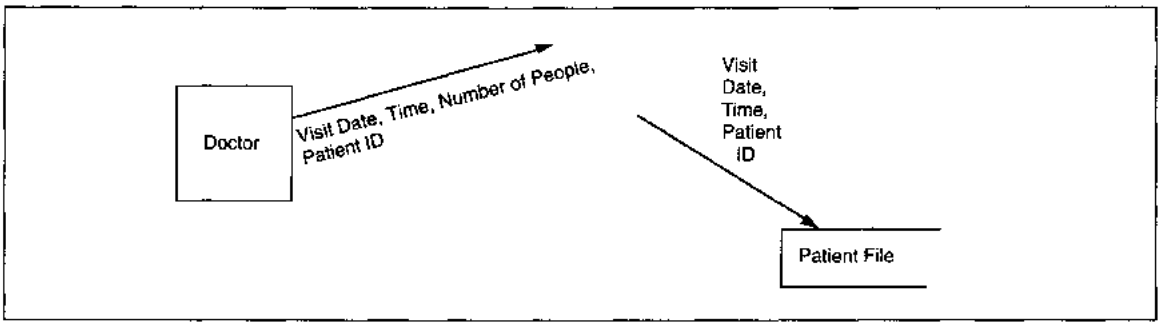


FIGURE 7-19 Skeleton Level 1 DFD with Net Inflows and Outflows for Process 1.0: Maintain Visit Information

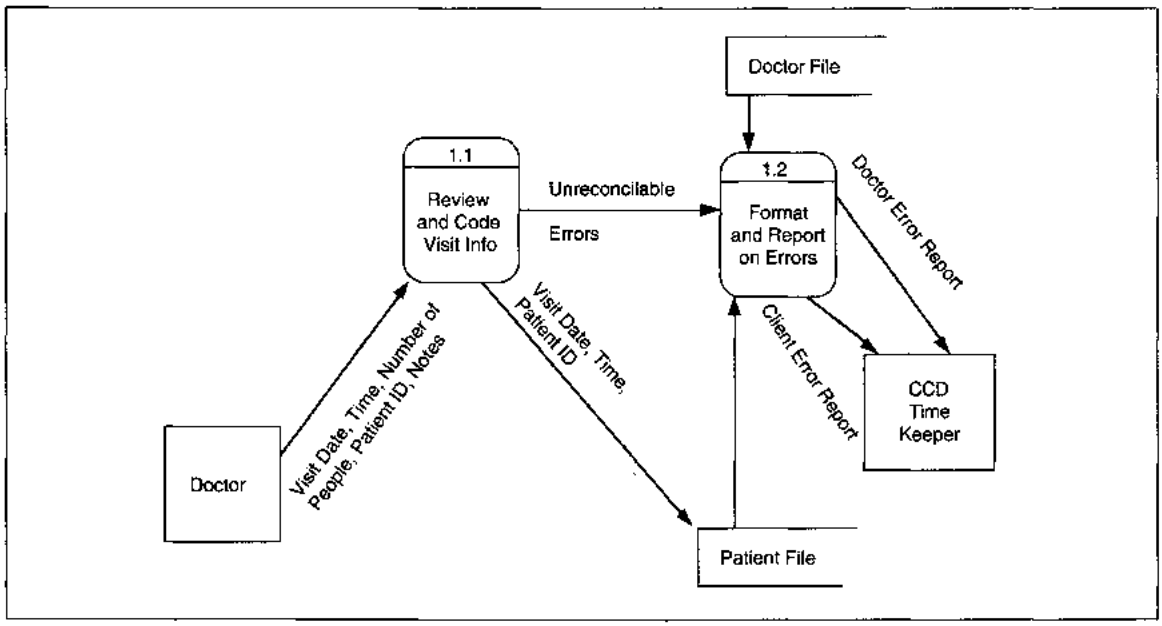


FIGURE 7-20 Completed Level 1 DFD: Maintain Visit Information

tions. Ideally, we want data flows and stores to contain exactly the information needed to trigger or perform each task, and nothing more. The questions and evaluation of processes in the decomposition process, if done properly, result in cohesive, minimally coupled processes.

Three types of quality checking are performed on the analysis results. First, **correctness checking** determines that the syntax and connections used in diagrams, charts, and so forth are accurately used. Next, **completeness checking** is performed with the users to validate the meanings of all terms and to verify the semantics used in all documentation. Last, **consistency checking** ensures consistency and correctness of all entries that span multiple diagrams, text, charts, and so on. Consistency checks evaluate the interitem syntax and semantics. These checks are first performed by the project team during walk-throughs or other quality assurance evaluations. Then, they may be reviewed by independent quality assurance analysts as an added check.

If you find data flows that are identical, with no transformations, going to many processes, reassess the processes definitions (see Figure 7-21). On the other hand, if you have a transaction processing application in which each transaction has its own version of some process, this type of diagram is correct (see Chapter 8). If the processes all do different transformations *and* have either unique inputs or unique outputs, leave them separate. If the transformations have an if-then-else logic, they are at too low a level and should be combined (see Figure 7-22). If they all do different transformations to the incoming data, are the processes' outputs going to the same place? If so, you may have overdecomposed and should combine the processes. Figure 7-23 shows two possible corrections to the overdecomposition. Either correction may be acceptable depending on the 'Y.y' data complexity and their processing complexity. Semantic (i.e., interpreting problem *meaning*) DFD problems are discussed again in the next section.

At Level 0, we did not concern ourselves with exception processing. At the lower levels, when a data flow is named '*Valid X*,' you must balance that flow with another one called '*Invalid X*.' In other words, you *do* define errors and exceptions at the

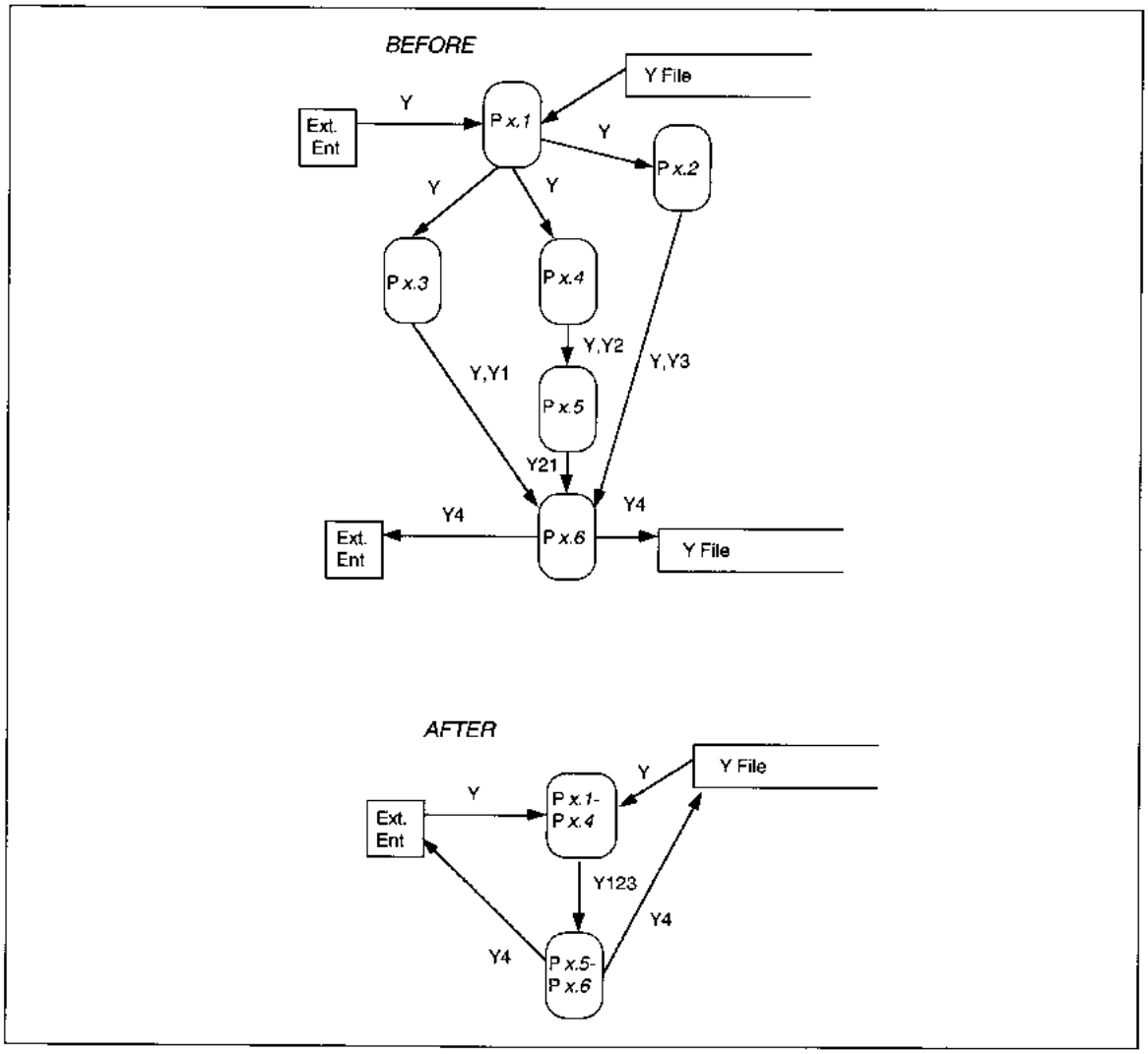


FIGURE 7-21 Example 1 of Excessively Detailed Processes

same level at which you define the split of valid and error/exception processing.

Let's examine how to apply these thoughts to develop a set of DFDs for ABC Rental Processing.

ABC Example Data Flow Diagram

We said above that in ABC Rental processing we are combining the analysis of context with analysis of the sequence of actions for each data flow. So, we start with a customer placing a video rental request. Customer and video information trigger a '*Create rental*' process. The first check in '*create rental*' is to validate the *customer*; if the *customer* does not currently exist, we want to '*add new customer*' to the company's files before rental processing. Here, we have a decision to make. We just described two input data flows to the *create rental* process. We need to decide if they are related or not. In this case, the issue is whether we can add new customers as a sub-

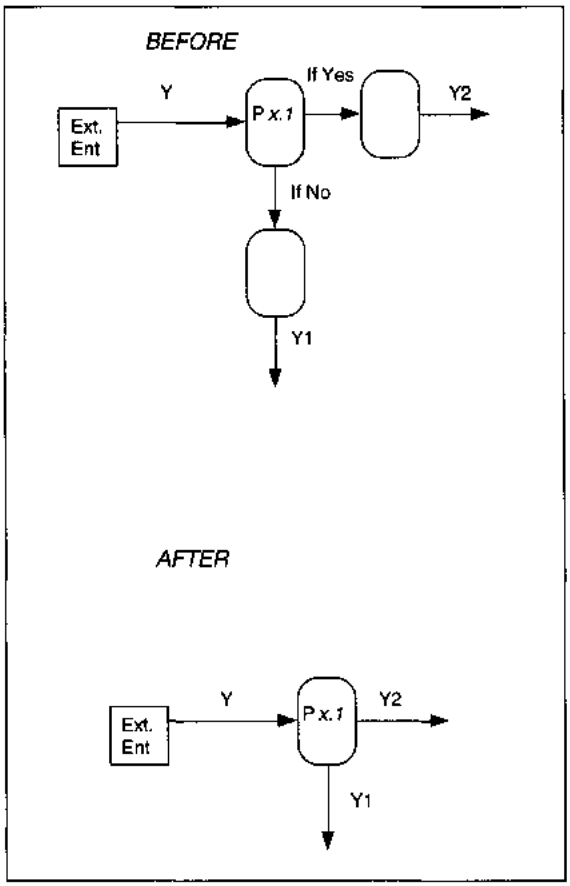


FIGURE 7-22 Example of If-then-else Logic in DFD

process of rental processing, or whether they are separate. If we separate the two, we have the two data flows we defined. If we combine them, we only have one data flow that optionally contains new customer information with rental information. If you do not know how the user wants the processing performed, you go back and ask. So, we will set this issue aside for the moment and finish defining what it means to '*create rental.*'[7]

7 Postponing decisions that are noncritical to the main logic is an important problem-solving behavior. Notice that we first identify alternatives and implications of the postponed item before setting it aside. If there are more side effects we have not identified, we are more likely to notice them with alternatives and implications than without.

After customer validation, we next have to validate the video and get a rental price. This requires reading some sort of video inventory file. Again, we ignore invalid video information for the moment. Once we have found the information on all the videos to be rented, we compute the total amount due. Again, we have a decision. At this point, how do we know whether late fees have been paid or not? Do we assume that people always return videos as they come into the store, and rent videos on their way out of the store? The rule is, *never assume anything*. If we know how to deal with this issue from the data gathering, we continue; otherwise, we add it to the list of questions for the user and continue.

After the rental amount is created (whatever it is and however it is computed), payment information is entered and customer change is computed. Then, the rental 'order' is written to a file and a paper copy is created for customer signing.

So, we have a process, '*Create rental,*' and we have several subprocesses, '*Validate customer,*' '*Validate video,*' '*Compute rental total,*' '*Process payment,*' '*Write rental,*' and '*Print rental.*' We also have several questions and decisions that we deferred. We can create the Create rental process on the Level 0 diagram whether we deal with the deferred issues or not. But we cannot identify the other processes, with certainty, until the issues on new customers and late fees are decided. So, we review the interview information and go see Vic for the detailed answers.

Mary goes back to Vic and says: "We are talking about the options for entering rentals and we have several questions. The first question is about new customers. One option is to separate the functions, that is, add new customers in a separate process from rental processing. A second option is to allow adding a new customer as part of video rental processing. A third option is to allow both. Do you have a preference?"

Vic: "I don't know. What will the cost differences be?"

Mary: "No matter what, you want to be able to add, change, and delete customers. It seems desirable to do that without being tied to the rental process. However, rental processing is

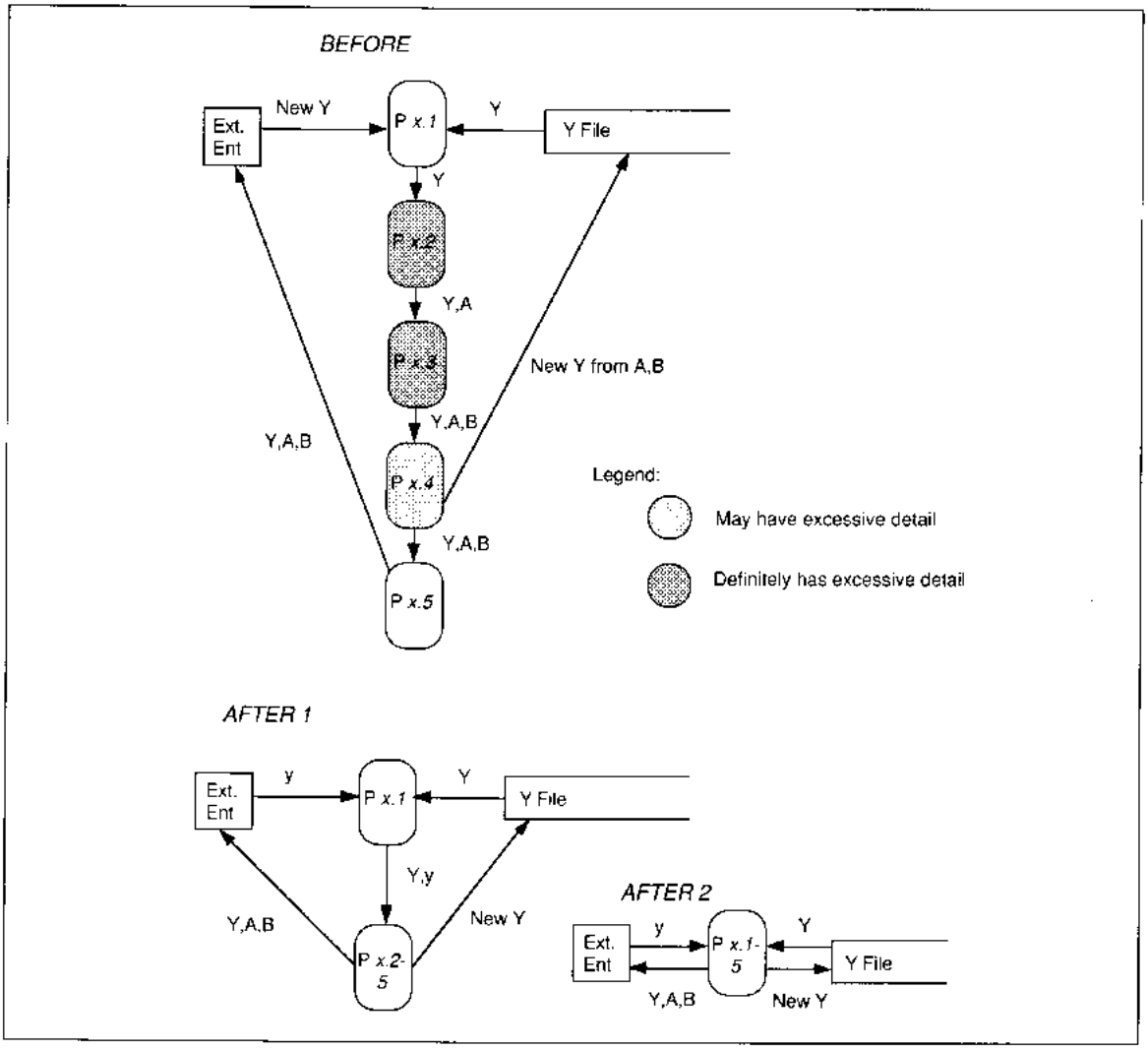


FIGURE 7-23 Example 2 of Excessively Detailed Processes

90% of your activity and you don't want to slow it down by having to leave that process to add a new customer. The slow-down for going from rental processing to add customer and back will range from 4 to 30 seconds depending on the PC's speed and the software we use. Unless you have a business reason for separating the two processes, I would suggest that you allow both. If we decide this direction now, there is no added cost. If we change direction in a few weeks, there will be a cost, as high as several thousand dollars."

Vic: "OK, let's do both, then. It sounds more convenient this way anyway."

Mary: "OK, we will allow entry of new customers as a process to be run by itself, *or* as part of rental processing.[8] My second question

8 Notice that Mary reconfirms the decision by repeating the agreed upon solution.

relates to video returns. When we collected our information, we observed people returning videos in several ways. First, they can put them into a slot and pay the fee the next time they rent a video. Second, they can return them and pay when they come in to get a new rental. Third, they can return them and rent a new video both at the same time. Do you want all of these options in the new system?"

Vic: "Yes, why wouldn't I?"

Mary: "It is easier for us if we have a somewhat fixed method of returns. But, if you want no changes, then we allow for all return methods. This may have a cost implication, but I can't tell right now. Should we talk about this again when I know what the cost of the options are?"

Vic was a little upset: "I told you at the beginning, NO bureaucracy and changes only if it improves convenience to my customers. If we don't allow them to return in all three of these ways, someone will get mad. Besides, don't customers pay when they rent? So, my only risk is on the 10% of customers who have late fees.

"Also, if I limit the ways they can return tapes, I lose my edge over Ajax Video's chain up the street. If there is a cost to allowing all of these things, why can't you tell now, and, if you can't tell now, when will you know?"

Mary tried to placate Vic somewhat but is still completely honest: "Usually, there is little incremental cost when all variations are known at this stage of the analysis. But I can't tell until we've proceeded a little further and have a sense of how many different programs will result from the most flexible design. I will know when we get to about two more levels of detail which will be in a few days. If there is no added cost, we will go for the flexibility. If there is an added cost, I will let you decide and give you an estimate for the different choices.

"Let me summarize: We will analyze for returns through the drop box, returns as a person coming in, or returns as part of rentals, and get back to you with cost implications, if any."

From the application perspective, maximum flexibility for both customer and return processing means, at least, that the rent and return screens and processing must be closely linked to each other. Now we need to guard against having the processes too closely coupled. Ideally, we want to accommodate Vic's wishes and still have processes separated as much as possible. To obtain this goal, we need to decide the minimum information needed to link customer and rental processing, and rental to return processing. Then, visualizing an implementation, we might be able to use, for example, windows for each process. We might open a new window to add a customer during rentals and maybe open another window to process returns during rentals. Also, with minimal coupling, we maintain separation even though the processes are interleaved.[9] This decision process is another example of how not top-down a top-down process is. We are going to an implementation level of detail to jump back up and define the data at the higher, more abstract level. Don't think this is the final answer. It is one way to reason through the problem and figure out how it might work at the computer level. Then, we back off to the logical level to describe that possible model.

We said before that the first step in *create rental* is to validate customer. If either the *phone number* or *customer name* is not retrieved, we know we have a new *customer* and can switch to that process. Once the *new customer information* is entered and saved, we can pass it back to rental processing as if it were in answer to an original request. Once we have the *customer information* in the *create rental process*, we can automatically check outstanding rentals. If there are any, we can ask if they want to return them or add the new *rentals* to the list. Our problem is solved unless Vic wants *late fees* processed whether or not the outstanding *rentals* have been physically returned. This decision, however, does not affect us until we try to define the details of processing. At the moment, we will assume *late fees* are only processed when the physical tape is returned.

9 Interleaving means weaving pieces of multiple processes together to give the *appearance* of parallel processing. Each process progresses a little. First, we switch to a process and do some of its function. Then we switch to another, then back to the first process, and so on.

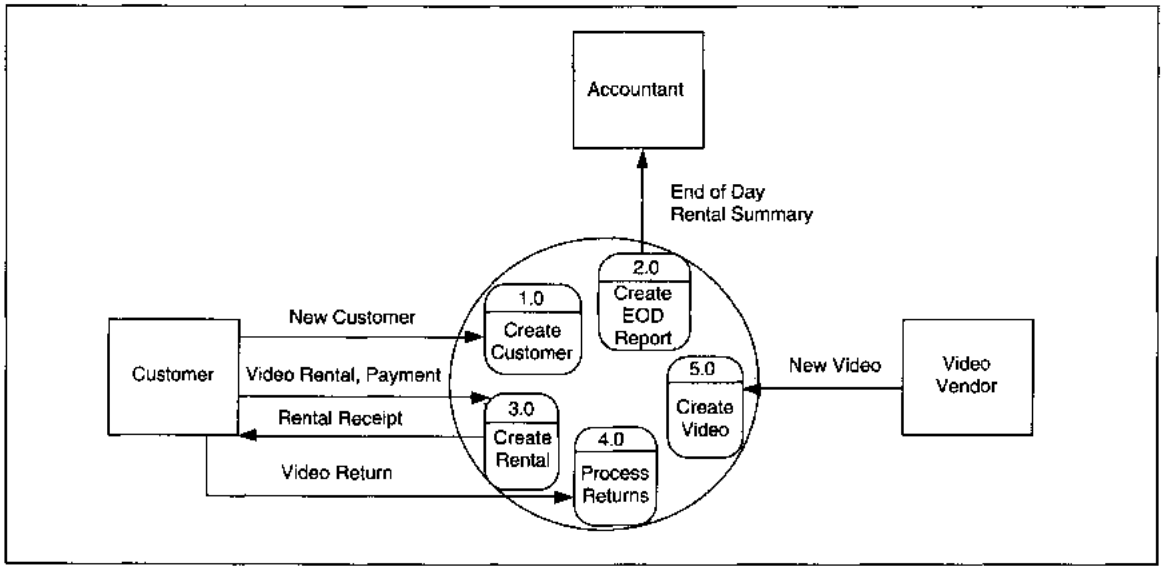


FIGURE 7-24 ABC Video Expanded Context Diagram

The result of this discussion so far is that we have three processes identified: *create rental*, *create customer*, and *process returns*. Each process could be initiated by the *create rental* process, or could be initiated by a customer action. We draw these processes (see Figure 7-24) and attach them to the correct data flows. Within the context circle expansion, do not show connections between processes. Processes still unaccounted for are '*create video*' and '*Create end of day report*' for summary totals. We know we have to get video information into the system, so we add that process and connect it to the data flow from video vendor. Since we must print an end-of-day summary for the accountant, we add the process to the diagram.

Figure 7-24 shows our high level processes of ABC Video Rental Processing, expanding the context diagram within the circle. The processes are shown in small circles or in rounded vertical rectangles, depending on local customs. This text uses rounded vertical rectangles. Notice that the data flows to/from each external entity are attached to a process, and all data flows are labeled and have a directional arrow showing which way the data is flowing. Also notice that the processes each have an action name beginning with a verb, and each process has a numeric identifier.

The next step is to expand to a Level 0 DFD, defining the data stores[10] in the application and linking processes, as required (see Figure 7-25). Data store identification usually occurs naturally during the identification of processes and subprocesses. For instance, what actions are done to enter a rental? First, you would check to verify that the customer is, in fact, a customer. This means checking some permanent 'list' or file for presence of the customer. Then, you would ask for each video they want to rent and verify the description and its price. To retrieve the description and price, we need a permanent file of the video inventory. When the rental is complete, it is stored somewhere (in a rental file), completing the process. Following this logic, we need at least three files *at this level of analysis*: *customer file*, *video inventory file*, and *rental file*. At this stage, we don't concern ourselves too much with the file con-

10 Other names for data stores are files, relations, or databases. The term data store means data relating to this name and does *not* imply normalized form. Data stores can contain more than one data structure [Gane, 1990].

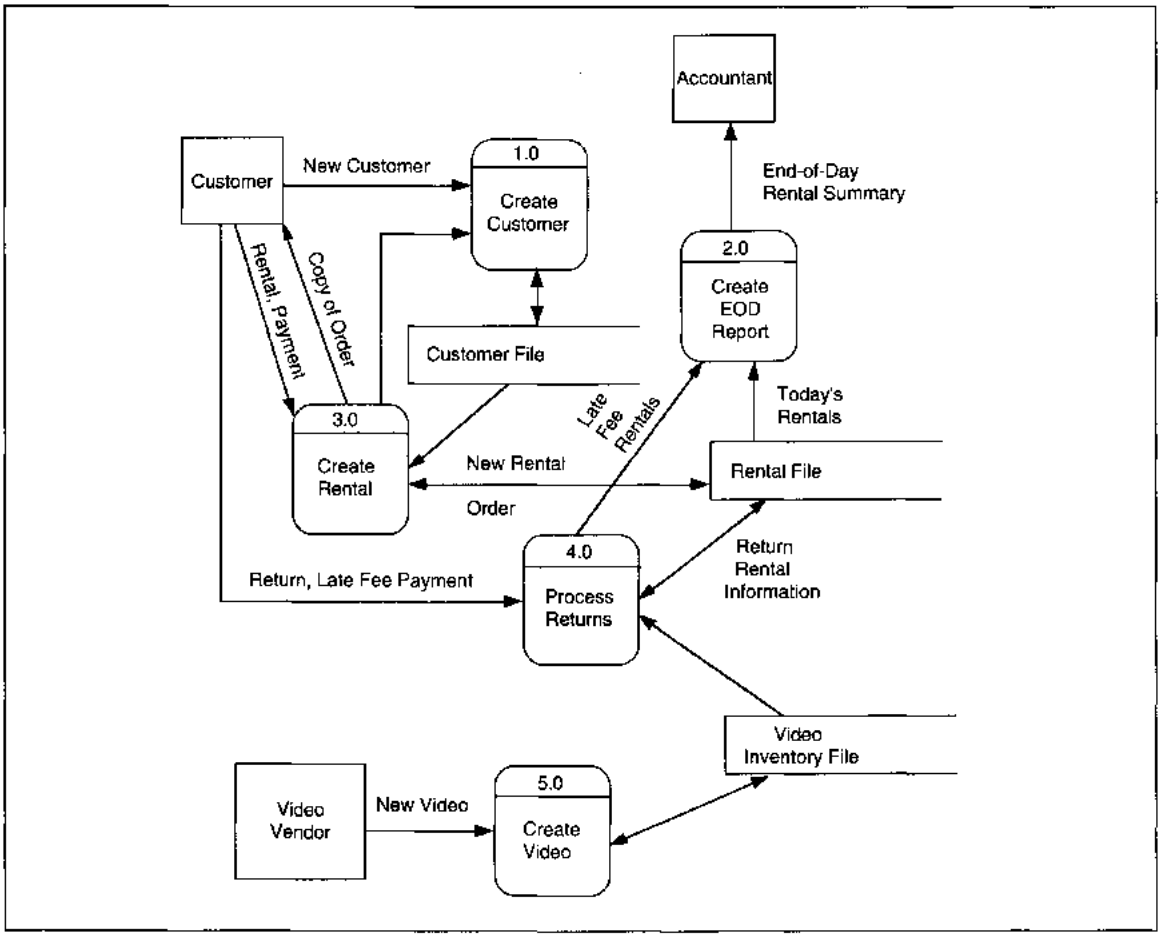


FIGURE 7-25 ABC Video First Cut Level 0 DFD

tents, although we identify the contents throughout analysis as they become known. As attributes, or fields, are discussed, it is a good practice to add to an attribute list for each file. The linkage between *create rental* and *create customer* is shown on the DFD as a data flow. The details of initiating *create customer* processing when a *customer* is not found are deferred to the next level of detail.

Before showing the DFD to Vic for his comments, we evaluate its level of abstraction and correctness (see Figure 7-25). Are *create customer, create rental, create video*, and *Process Returns* all on the same level of abstraction? The first clue that they are is that the first three processes all have the same verb. *Process returns* is the removal of rentals just as *create rental* is the creation of rentals; they are reciprocal processes. The reciprocal processes also appear to be at the same level. The name *process returns* is not the best we could choose to show reciprocity; *return rental* is a stronger name that does and we change the process name.

Next we evaluate correctness of the diagram. Are all the connections legal? Yes. Are there any pathological connections? No. Is there a *flow* through the application? Yes, the main flow is for rental and return processing.

Now, we could return to Vic and ask his opinion, giving him a verbal presentation of the details

TABLE 7-1 Decision Table for Decomposing Another Level of Detail

Conditions											
Domain Knowledge	H	H	-	-	H	H	H	L	L	L	L
Language	4GL	3GL	3GL	3GL	2GL	2GL	2GL	4GL	4GL	3GL	2GL
Similar Experience	-	Y	N	N	Y	N	N	-	-	-	-
Simple Process/ Few Files or Complex Process or Many Files	-		S	C	-	S	C	S	C	-	-
Recommended Decomposition Levels											
Level 0	X	X	X	X	X	X	X	X	X	X	X
Level 1	Opt.	X	X	X	X	X	X	Opt.	X	X	X
Level 2		Opt.	Opt.	X	Opt.	X	X	Opt.	X	X	X
Level 3 . . . n				X		Opt.	X		Opt.	X	X

Legend:

H	Extensive experience
L	Little experience
4GL	Fourth Generation Language, e.g., SQL
3GL	Third Generation Language, e.g., COBOL
2GL	Second Generation Language, e.g., Assembler
Y	Yes
N	No
S	Simple
C	Complex

underlying each of the processes, and in the details, getting verbal agreement to the next lower level of subprocesses.

At Level 1, we first decide which, if any, processes need decomposition. What happens when you create customer? A quick definition of fields and the type of validations required is necessary. According to the information (see Chapter 2), we need customer phone, customer name, customer address, and credit card ID, number, and expiration date. Validation for these fields is that the data are present and legal for the data type. For complex validation, you frequently use extra **cross-reference files** to contain the legal codes and their meanings.

Do we also need to provide modify and delete processing for customers? Always is the answer, . . . and query processing as well. Now, we need to know the implementation language to decide whether or not to decompose further. The decision table shown in Table 7-1 summarizes the decision criteria and the most likely outcomes. Keep in mind that you can *always* go to another level of detail and can always get *some* benefit from the exercise. But, why do the work if you don't have to?

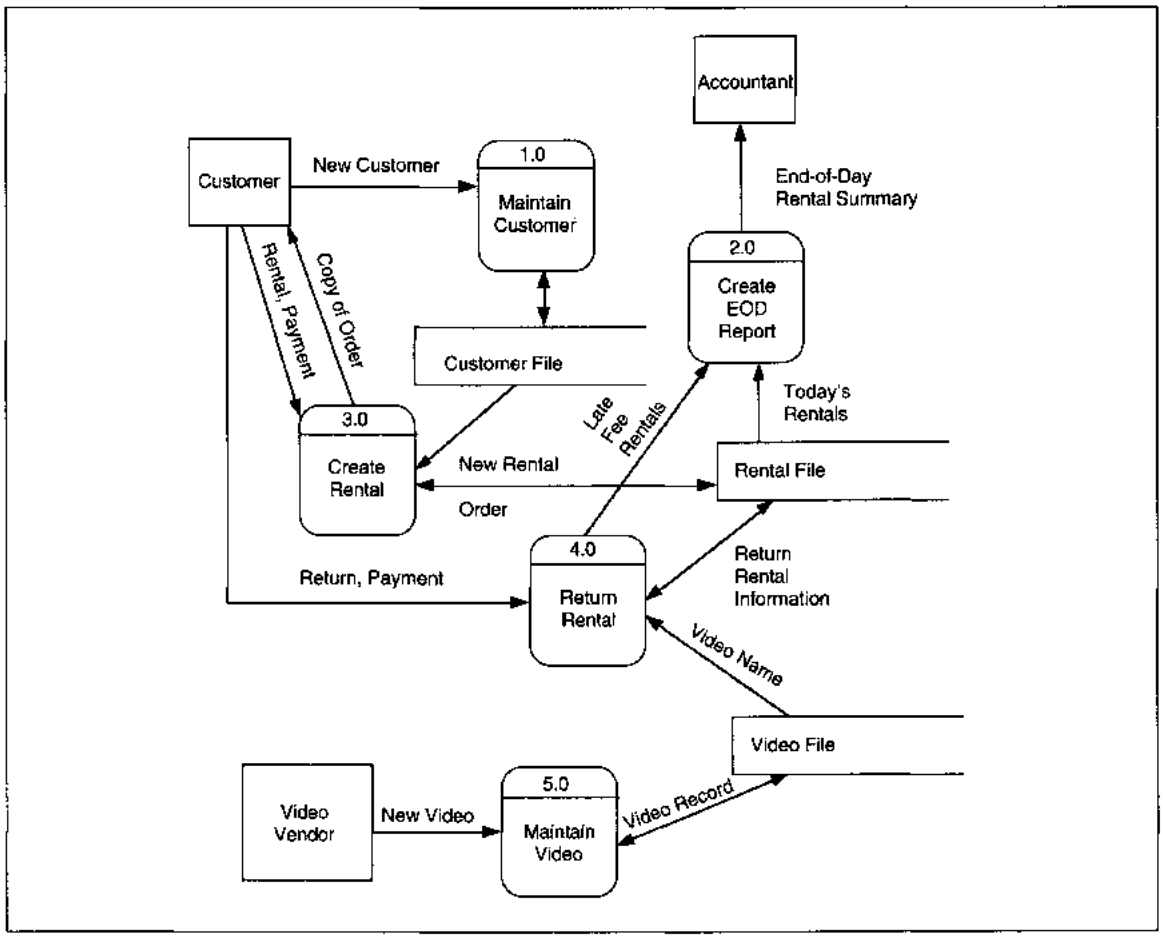


FIGURE 7-26 ABC Video Final Level 0 DFD

We are planning to build this application for a LAN environment, using a 4GL-nonprocedural language. For *create customer* there are no other data stores needed for validation. There will be add, change, delete, and query processing. The corresponding decision cell—4GL, simple process, one file—shows Level 1 to be optional. The decision depends on who is doing the programming. Is the person experienced with similar applications? Is the person involved in analysis fully knowledgeable about the requirements for this application? If the answer to either of these questions is 'no,' the next level of DFD should be developed with the details entered in the dictionary.

For ABC Rental Processing, we will opt not to discuss development of the Level 1 DFD for *create customer*. We will change the process name to '*maintain customer*' to denote the more general and expanded processing. The final Level 0 DFD is Figure 7-26; the Level 1 DFD is shown as Figure 7-27 for reference.

A similar set of arguments for Process 4.0, '*create video*,' is possible. We also rename that process '*maintain video*' to denote the expanded processing, and omit the level 1 DFD.

Both rental processing and return processing should be expanded regardless of the implementation language because they are fairly complex and

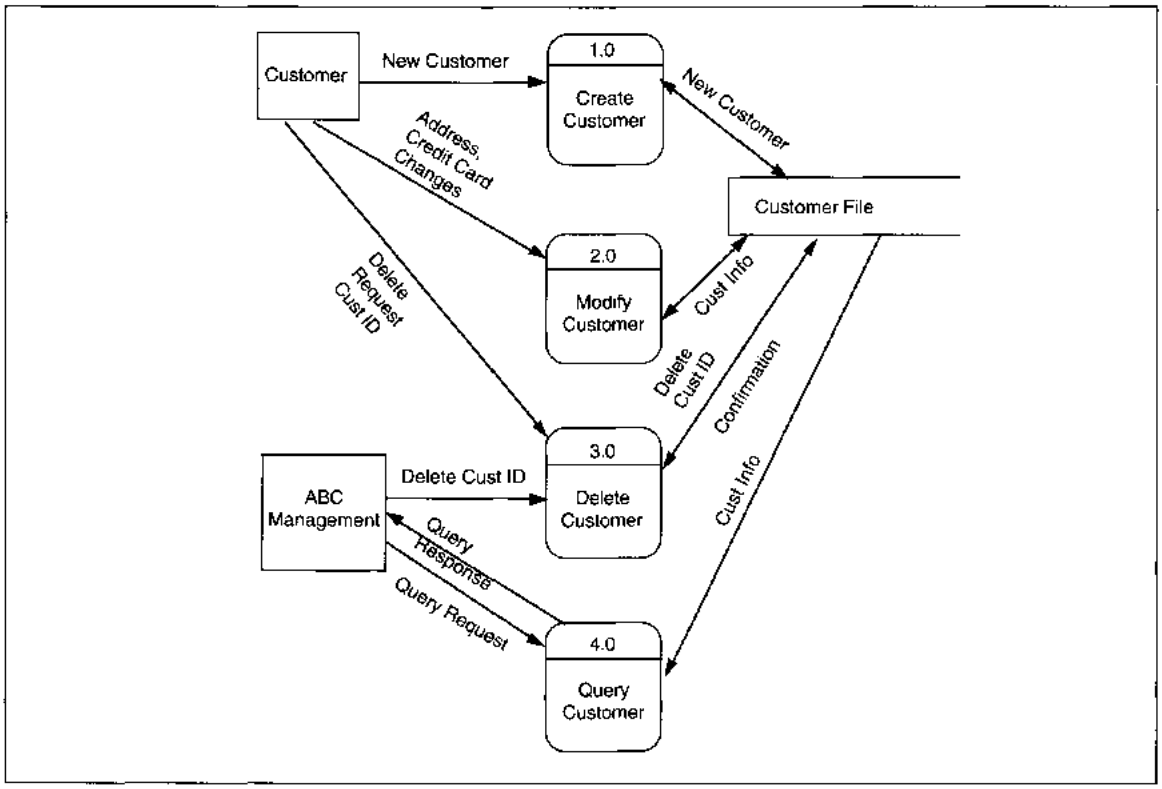


FIGURE 7-27 ABC Rental Level 1 DFD for Maintain Customer

we have not discovered how they work yet although we have described rental processing in some detail. First we examine the DFD from our knowledge so far, then expand it as required (see Figure 7-26). In the level 0 DFD, the create rental process interacts with customers twice and with all three data stores. To untangle and clarify the processing of these five interactions, we decompose the process further.

The first interaction is to get rental information from the customer. The '*rental information*' includes *customer ID* (or name) and *video IDs* (or names). The *customer ID* is used to validate the customer and get the rest of the customer information for the rental. Similarly, the *video ID* is used to validate the video and get the rest of the video information for the rental. *Customer ID* is also used to check for late fees and to retrieve outstanding rentals. We also know that if the customer is not on file, we want to initiate process 3.0, *maintain customer*. When combined, this processing is fairly complex and somewhat extensive. It is complete when the clerk does something to show that entry of rentals is complete. We can group these processes together and call them '*get valid rental*' (process 1.1) because once these actions are complete, the rental is ready for the next step of processing. The detailed steps we identified are either used to create another level of DFD or are documented in the dictionary for process 1.1.

A valid rental is totaled by adding all of the rental fees for the current set of entries and any late fees outstanding from past rentals. Once the total is displayed, the amount of money paid by the customer is entered into the system by the clerk. The total paid is subtracted from the total due to get the change due to the customer. When the change and total due amounts are both zero, the rental is complete and ready for the last part of the process. Because this stage is discrete, beginning with the successful vali-

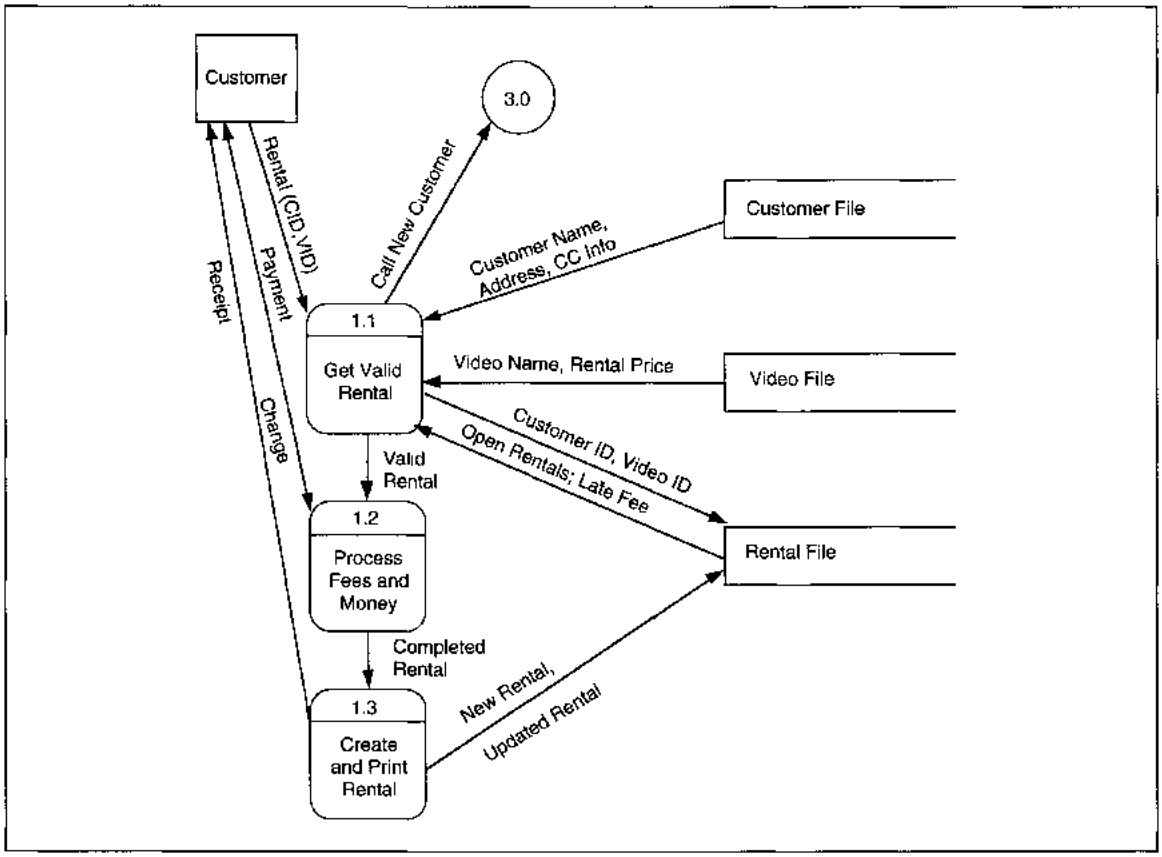


FIGURE 7-28 ABC Rental Processing Level 1 DFD

dation and ending when the change and total due are zero, we group these actions together and call them '*process fees and money*' (see Figure 7-28).

Finally, a rental is completed by saving all the information in the *rental file* and printing the *receipt* for customer signature. When these actions are complete, the *create and print rental* process is complete (see Figure 7-28).

Notice that we have decomposed the data flows as well as the processes. Where we group rental and payment on the level 0 diagram, we separate them on the level 1 diagram. We add *change* to the process because now we are dealing with the details. Similarly, the data flows connecting to the data stores are decomposed to show details of data passing back and forth. On a DFD, we assume all data can be passed when the data flows are not labeled, and it is okay to summarize on level 0. At level 1, we become specific and show the interface accurately and in detail.

When you are drawing the DFD, you have to guard against being too detailed. This is difficult, especially for novice analysts. If your drawing has these symptoms, you are too detailed and must combine processes to a higher level of abstraction. The semantic process problems to look for are listed with examples below. These problems violate one or more of the DFD Semantic Rules and Heuristics:

1. Processes that have only one data flow from the previous process as its input are probably overspecified. The solution is to

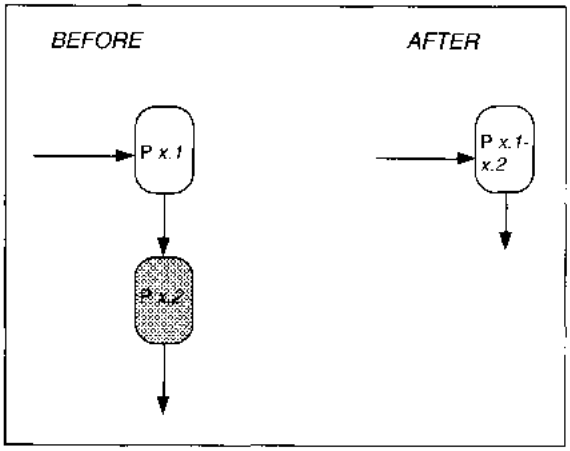


FIGURE 7-29 Example of Pathological Data Flow

combine the data flows (see Figure 7-29). Another solution may be the addition of a missing external entity (see Figure 7-30).

2. When several processes have interactions with the same external entity and at least one process has no other interactions, check that the data flows and transformations are different. If any two processes have the same outflow or are closely related, that is, passing one's input data to the next, they are probably overspecified. It may be possible to localize all external entity interactions in one process, and to perform all processing on the information obtained in the other process (see Figure 7-31).
3. When several processes have interactions with the same file and at least one process has no other interactions, check that the file contents read/written and transformations are different. One goal of all application is efficiency. If you read the same data more than once, it is inefficient. It is somewhat better to pass the data between processes. If you are identifying only logical processing and have the reading to show where data is used, make a note that during design you will need to redevelop the DFD to show physical reads of the file. It may save time to redevelop the DFD at this stage rather than wait. Several solutions are possible (see Figure 7-32). In the first solution, all file interactions are localized in one process; in the other, inputting from the external entity and file are in one process and outputting is in the other. Both of these solutions require rethinking of the functional decomposition.
4. If several processes have more than one write to the same file, check that the processes are distinct and that the data must be written disjointly. Again, to have efficient file processing, minimal reading and writing is desired. The alternatives are to localize reading and writing as in the first solution (see Figure 7-33), or to combine

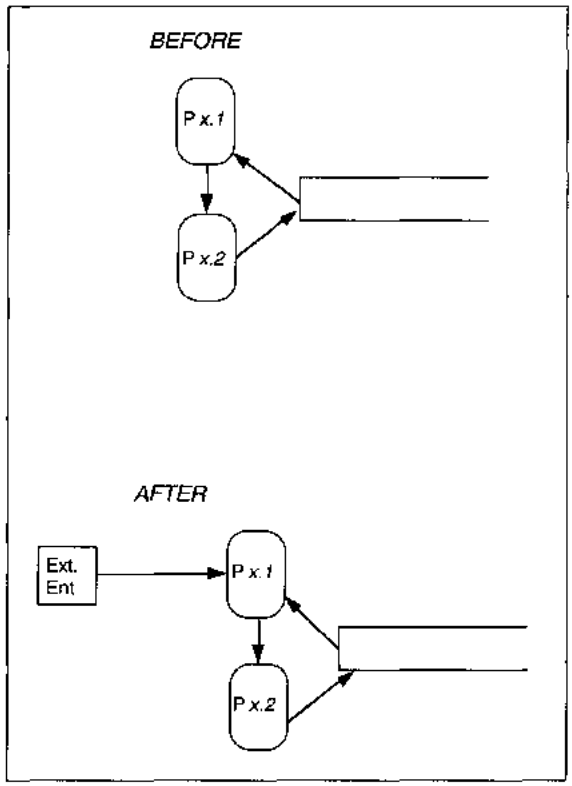


FIGURE 7-30 Example of Spontaneous Process

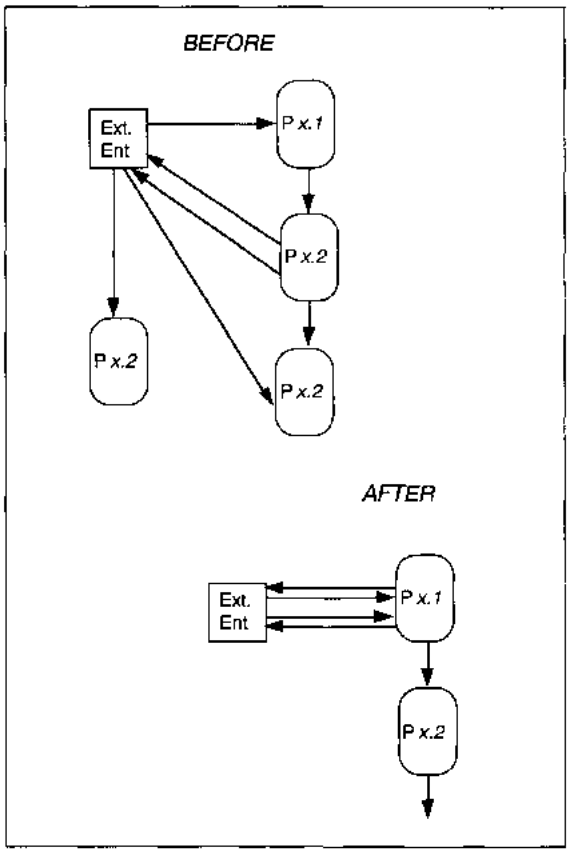


FIGURE 7-31 Example of Overspecified Entity Processes

some of the processing but include writing in more than one process as in the second solution.

5. Any imbedded if-then-else logic that describes process interaction is wrong. Remove the logic by consolidating the processes. The logic belongs *inside* the process box, not outside; one solution is shown as Figure 7-34. If this problem occurs, make a note to include the control on the structure chart for the if-then-else logic, as required.
6. Processes that do only one very minor process, for instance, check customer number for validity, may be overspecified. A better process would check the customer information, do a credit status check, and identify outstanding late fees (see Figure 7-35). This example is an improvement because it is reading and validating all customer data only once.
7. Make sure that no physical entities, such as cash register or bar code reader, have sneaked into the DFD. Also make sure that no immediate users of the application are identified on the DFD. The solution to this problem is to remove all physical entities on any diagram in which they occur (see Figure 7-36).
8. Make sure that data flow names are field contents being passed or some group name for field contents that clearly identifies the information (see Figure 7-37). Unnamed data flows are frequently masking overspecified processes. If you cannot develop a unique, meaningful name, reevaluate the process they attach.
9. Data stores may show up on diagrams multiple times with the same name. To show that you know it is repeated, place a vertical bar down the left side of the file symbol.
10. Similarly, data flow names may show up multiple times with the same name. This condition is okay *if, and only if,* the contents are identical. This condition is rare, so when multiple data flows with the same name are present, there is frequently an error. Double check any data flows with the same name and give any unique data flows their own descriptive name (see Figure 7-38).
11. To simplify the design phase activities, make sure that process names include the transformation name and identify the data being transformed.
12. If data stores have only one input or one output, check that it is correct. This condition may be okay on the input side as long as maintenance is performed in some other application, or for files that are cross-reference tables only. The condition for output-only connections may be correct, for instance, for temporal databases in which

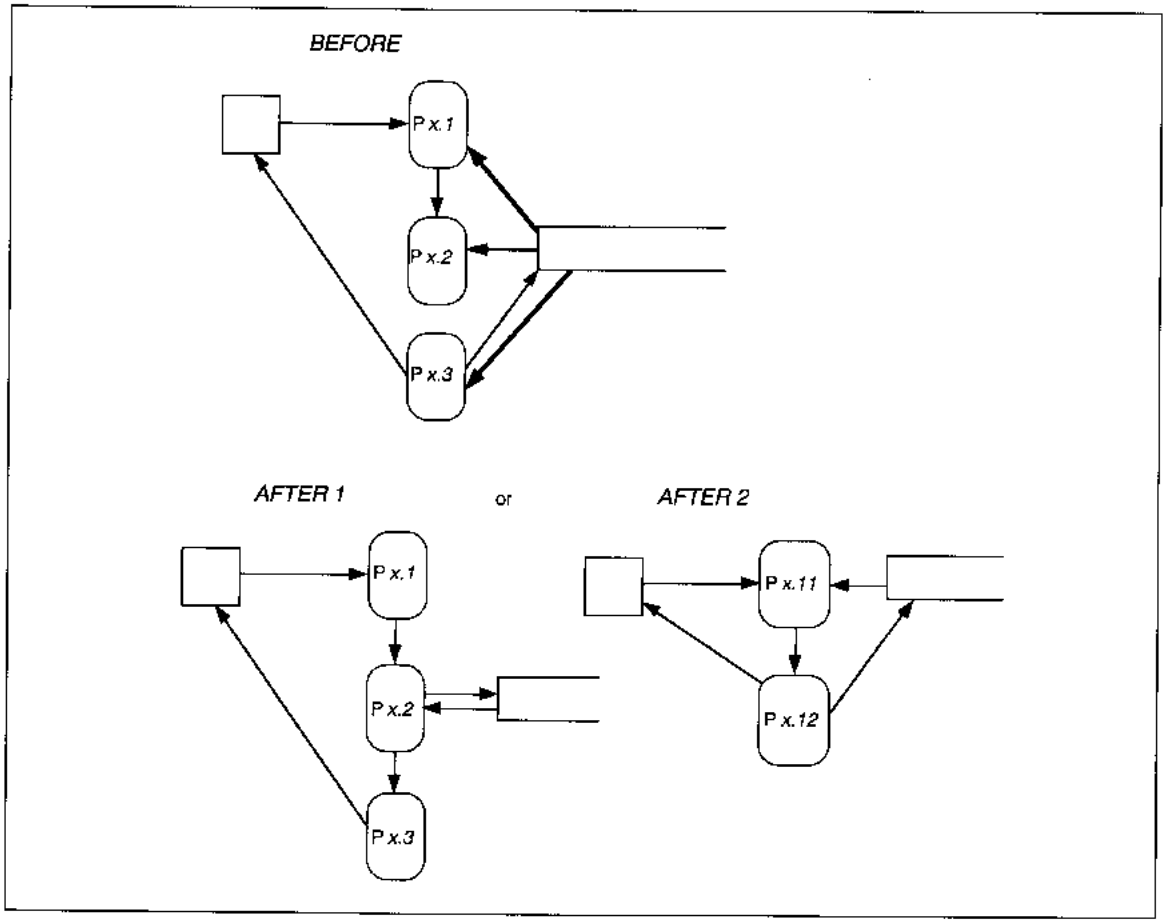


FIGURE 7-32 Example of Overspecified Read File Processing

nothing is thrown away. Check the business rules relating to the data and verify the processing.

For return processing, we need to walk-through the process to define if we need subprocesses. A *video ID* is entered and used to retrieve the rental. The system assigns today's date as the return date. *Late fees*, if any, are computed. The *total amount due* is computed. The *total amount due* is displayed, an amount of money received from the customer is entered, and *change* is computed. When both *total amount due* and *change* are zero, payment processing is complete. If no *late fees* are owing or payment is complete, the *open rental record* is removed from the *open rental file* and *history information* is updated. If *late fees* are owed but not paid, the *open rental record* is rewritten with *return date* and *late fee* information. Return processing has several steps, but each is simple, requiring at most one file per step. There is little need for a Level 1 DFD for this process at this time.

Notice that the *process fees and money* is identical to the same process for rental processing. We can develop a common, reusable module for both rental and return processes. Also, notice that we introduce history here. If we decide to have a history file, it would show at this level of DFD.

At this point, we are ready to reevaluate the new DFDs and proceed to development of dictionary entries for all DFD information. Check the final DFDs for legal connections, similar levels of abstraction, and balanced net inflows and outflows between levels. Then, continue to the data dictionary.

Develop Data Dictionary

In this section we briefly discuss the contents and rules, if any, for each type of dictionary entry. Then, we will document the information from the ABC rental application. Since you have seen examples of each type of entry, this section is short.

Data Dictionary Contents and Rules—Entities

The contents of the dictionary for external entities are listed in Table 7-2. The most important are the name and the definition of the entity. In organizations with data administration functions, this information must conform to the 'corporate' dictionary definitions or must be reconciled with it to define new terms. The SEs work with users and data administrators to name and define the entities for the organization. IS personnel do *not* name and define the terms by themselves. Most external entities are people, job titles, organizations, or applications with

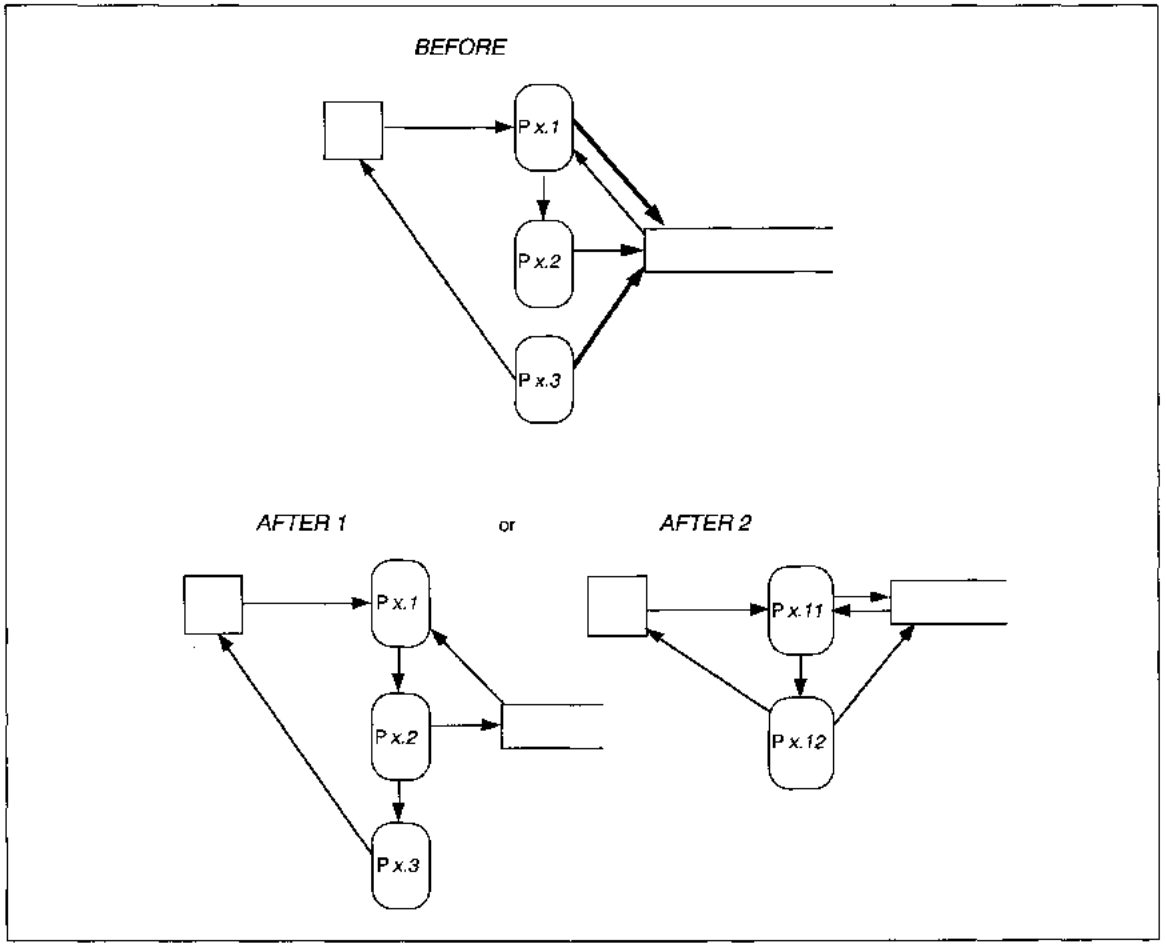


FIGURE 7-33 Example of Overspecified Write File Processing

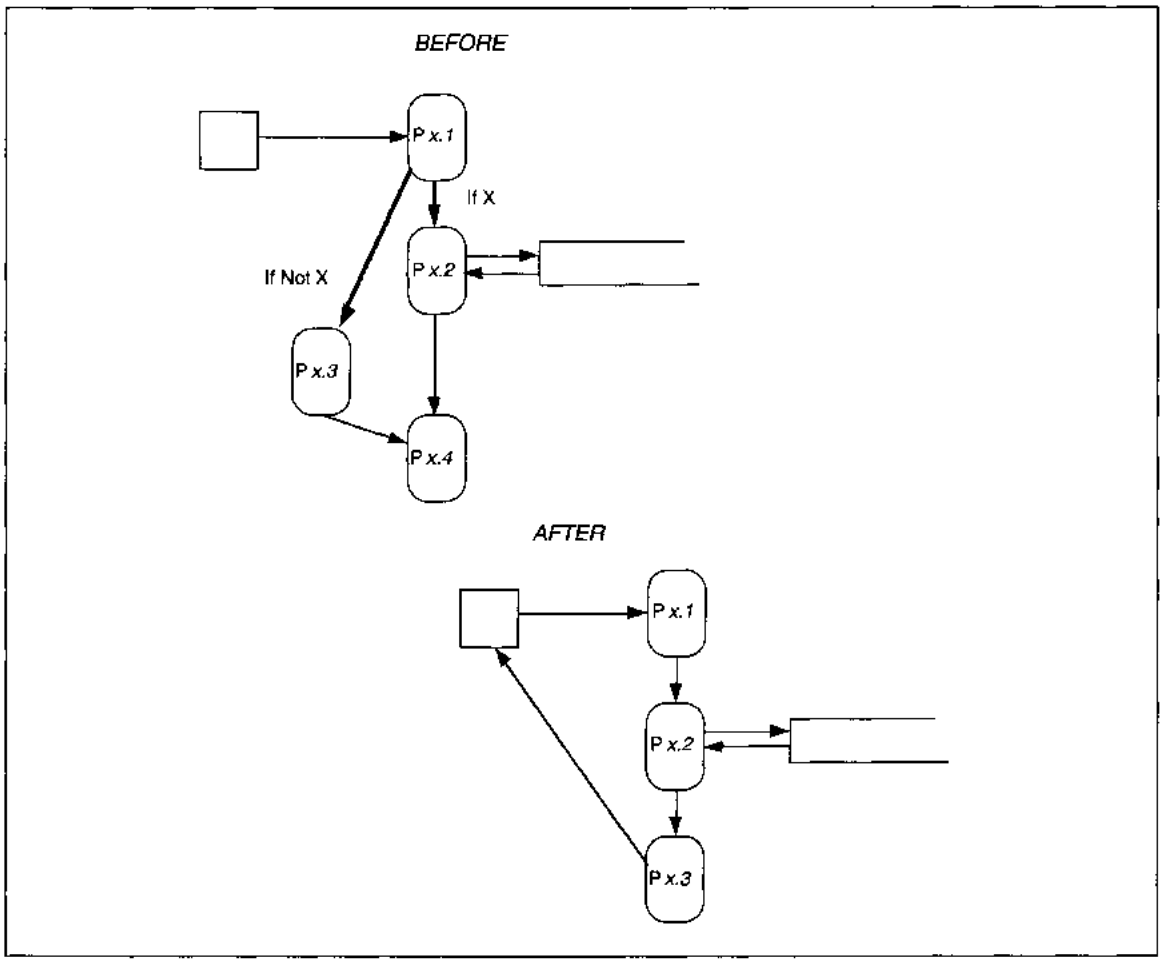


FIGURE 7-34 Example of If-then-else Logic in DFD

TABLE 7-2 Data Dictionary Entity Contents

Entity name
Aliases
Definition
Relationship to application
Contact, if entity is an organization

which the application under development interacts. Choose a meaningful business name that describes the entity accurately and completely. If you have a data administration function, use their name. The definition should be a business definition and should be completely independent of *any* technology.

Make sure you include in the definition any aliases or names used in your application that do not conform to the corporate standard. Describe the entity's relationship to the application in terms of the nature and timing of the interaction. If the entity

is an organization, include the name, address, and phone number of the person most frequently contacted.

Figure 7-39 shows the notation to be used in describing the contents of an entity to a dictionary. Keep in mind that this convention works well if you are using a manual method. Automated tools have their own format and notation for repository contents. There is one notational structure for each type of entry: optional information, multiple repeating information, required information, selection between attributes, and primary keys.

ABC Example Data Dictionary—Entities

The external entities in ABC Rental are *customer*, *vendor*, and *accountant*. The entries for each of these are shown in Table 7-3. If the accountant is an

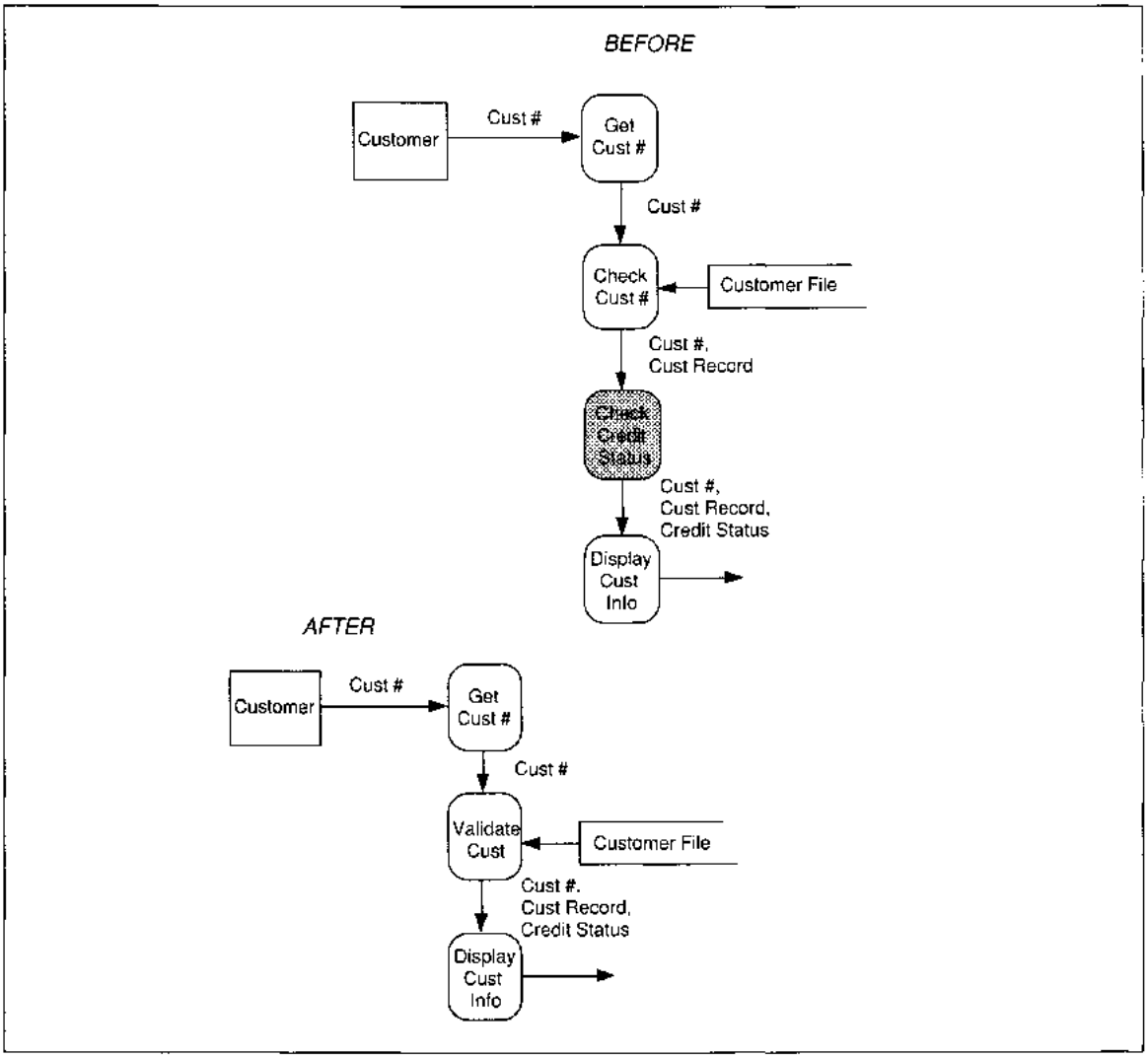


FIGURE 7-35 Example of Excessive DFD Detail

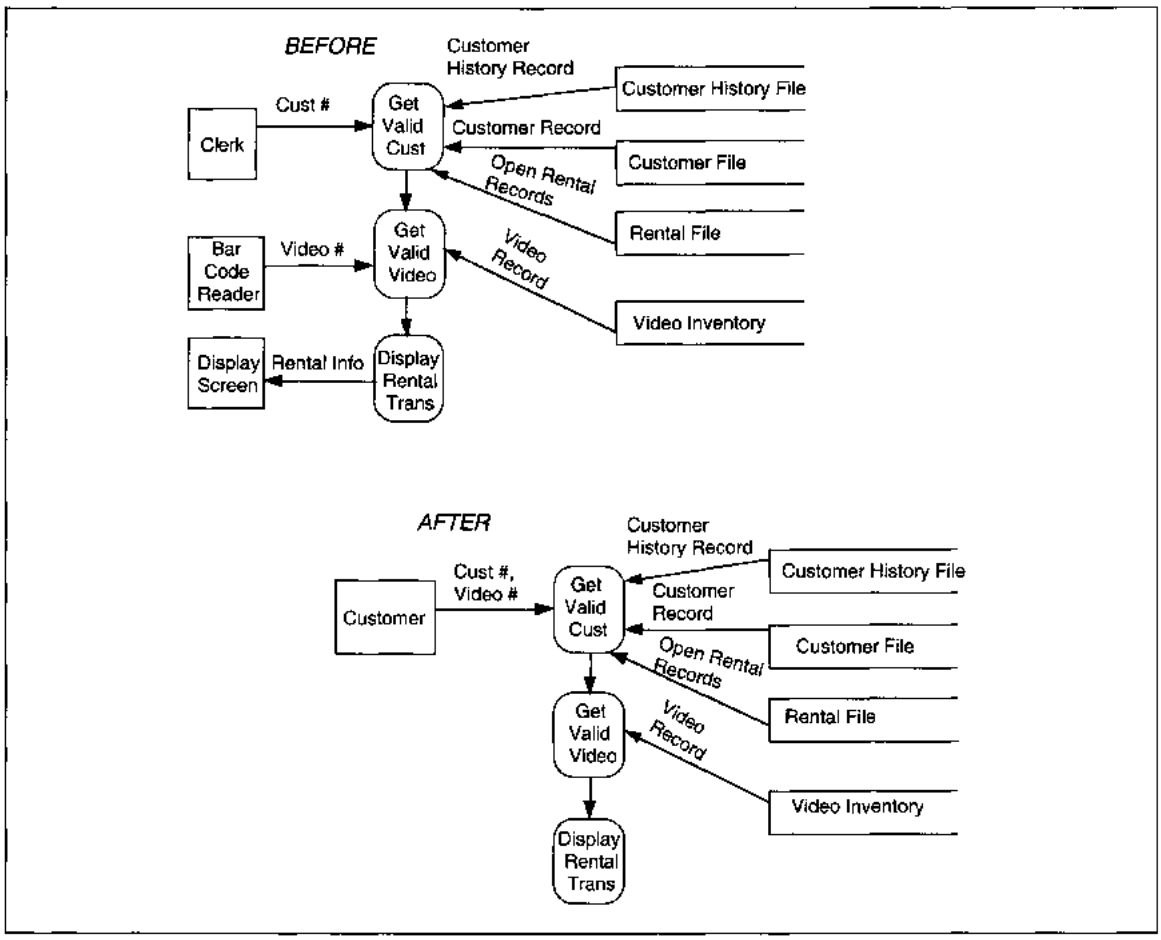


FIGURE 7-36 Example of Physical Entities

employee, you would not include his or her name in the dictionary. If the accountant is an outside firm, you *would* include the information.

Data Dictionary Contents and Rules—Processes

The contents of the dictionary for processes are listed in Table 7-4. For processes, we include the process number from the DFD to allow quality assurance, and to easily link back to the process model. In a computer-aided software engineering tool (CASE), if you used one, you usually have automatic linkage between the diagram and the dictionary entries. The name of the process should be *exactly the same* as the process name used in the DFD.

The **process description** details the steps to complete the process and can take several forms. The most common are pseudo-code and structured English, supplemented by decision trees or decision tables as needed. **Pseudo-code** uses the syntax from a language in abbreviated form for easy translation into the target language. **Structured English** is a computer-language independent description of a process using only simple verbs and terms from the dictionary; no adjectives or adverbs are used. Structured English is used here.

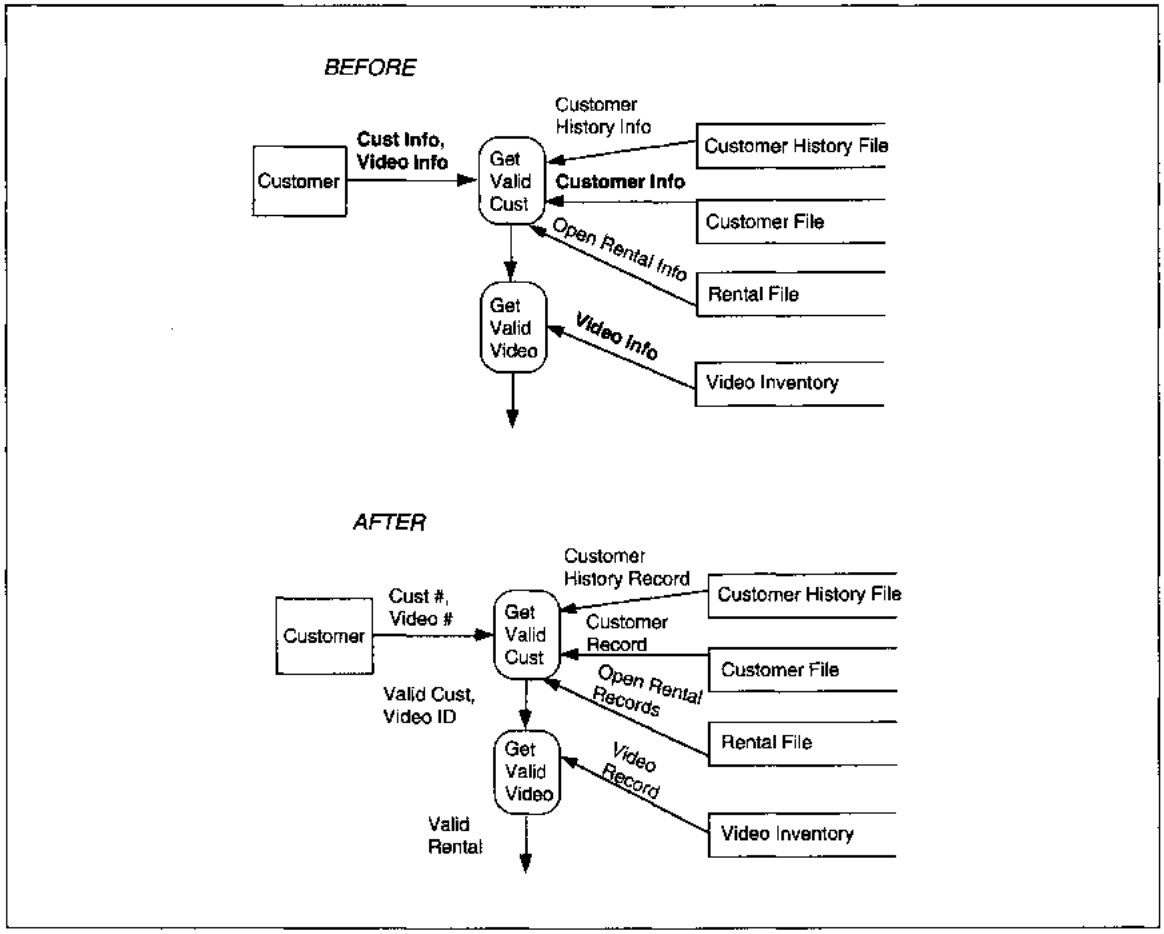


FIGURE 7-37 Example of Weak Data Flow Names

ABC Example Data Dictionary—Processes

The process entries for ABC are all included at the level 0 detail level (see Table 7-5). To document the entire application, you would create a data dictionary entry for each lower level process, then refer to that process in the higher level dictionary entries. In this way, the hierarchy of processing and linkages are documented.

Notice that there are some uneven levels of detail in the process entries. For instance, the *process fees and money* routine is fairly detailed, while the reference to *create history* in *return rental* is not detailed at all. You document the information you have, replacing the high level abstract thoughts with the details as you come to know them. The dictionary is constantly evolving and changing as more information becomes known.

Data Dictionary Contents and Rules—Data Stores

The data store defines persistant data; contents of a data store dictionary entry are listed in Table 7-6. There is a significant amount of detail that is eventually documented. You begin completing the information as it becomes known and complete the rest

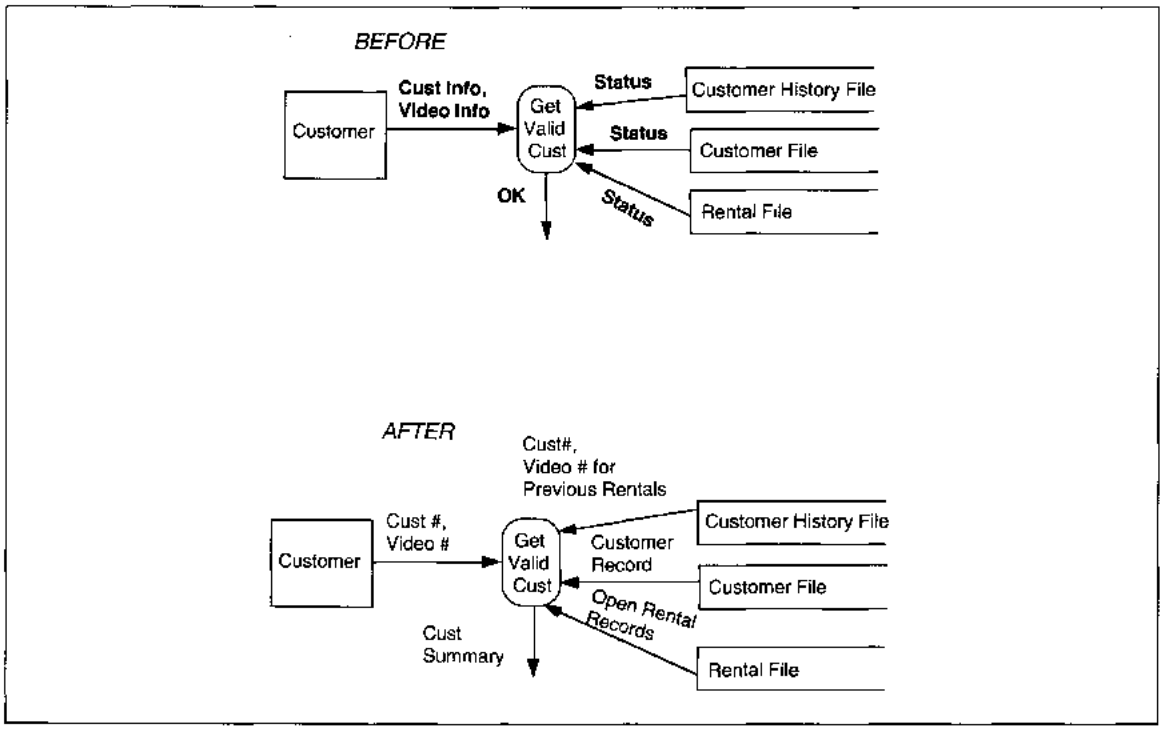


FIGURE 7-38 Example of Nonunique Data Flow Names

when it is available. Also, some of the information may not be relevant in your organization (for instance, if all projects always use DB2 relational files, you may not need detailed documents because the information already exists). The goal of the documentation is to present necessary information without much verbiage. Keeping that in mind, trim the dictionary entries to fit your situation.

ABC Example Data Dictionary—Data Stores

The dictionary entries for data stores are in Table 7-7. For now, we know very few of the details about, for instance, volume, growth, and security. Those entries are left blank.

Above, we said that you trim the contents of the dictionary entries to fit the project. In a consulting situation, such as Mary and Sam are in at ABC, the likelihood of them also maintaining the application is unknown. So, the more detailed the documentation, the more you simplify future maintenance.

Data Dictionary Contents—Data Flows

Data flow contents are important pieces of documentation because they cause the creation and change of files and determine the data each process actually accesses. The data flow contents are shown in Table 7-8. Contents have a primary key to uniquely identify the data. The difference between primary key for a data flow and for a data store is one of time. What period of time is the flow 'alive'? Data flows usually have a short life which means that less data is required for a unique ID. For instance, the flow payment is a money amount which is acceptable here. At the implementation level, that field might also require a terminal ID or a transaction ID

Symbol	Definition
=	is composed of
+	and
( )	Parentheses show an optional entry which may or may not be present
n{ }*m*	Braces show iteration *n* is minimum entries *m* is maximum entries If no limit to entries, the maximum is shown as *m*.
[ ]	Square brackets identify selection from among alternatives
|	Vertical bar is a separator of alternative choices within square brackets
*	Comment
------	Underline identifies a component of a primary key

*Adapted from Yourdon, Edward, *Modern Software Engineering*. Englewood Cliffs, NJ: Prentice-Hall, Yourdon Press, 1989, p. 191.

FIGURE 7-39 Data Dictionary Notation*

to be unique; implementation requirements are not dealt with in analysis. Data flow constraints are most often present in real-time applications or in applications with contingent processing of data. The source of the data flow is a cross-reference back to the entity, process, or file from which it flows.

ABC Example Data Dictionary—Data Flows

The data flows for ABC rental processing are shown in Table 7-9. There is nothing difficult about any of them. Keep in mind that these definitions are not cast in concrete; they can change whenever the need arises. It is important to keep this information up to date, because programmers use the dictionary to check that their modules are receiving the correct information.

TABLE 7-3 ABC Entity Dictionary Entries

Entity Name:	Customer
Aliases:	None
Definition:	A Customer is any individual, organization, or other entity authorized by ABC management to rent videos.
Relationship:	Rents and pays for videos Signs rental order Provides new customer information Returns videos
Contact:	N/A
Entity Name:	Video Vendor
Aliases:	Vendor
Definition:	A Video Vendor is any organization or individual from which ABC purchases or otherwise acquires videos.
Relationship:	Provides new video information
Contact:	N/A
Entity Name:	Accountant
Aliases:	None
Definition:	The employee providing accounting services for ABC video.
Relationship:	Gets end-of-day summary accounting reports
Contact:	N/A

TABLE 7-4 Data Dictionary Process Contents

Process ID Number
Process Name
Process Description
Constraints (e.g., concurrence, sequential with another process, time-out, etc.)

TABLE 7-5 ABC Process Dictionary Entries

Process Number:	1.0
Process Name:	Create Order
Description:	For each customer, Enter customer ID (or name) Read customer file using customer ID (or name) as key If NOT present display 'Customer not currently on file, switching to create customer' Call New-customer routine. Display all customer infor- mation. Read Rental file using customer ID If rentals exists, display rentals If returns Call Return routine else continue else If late fees outstanding add late fees to total. For each video, Read inventory file using video ID (or description) as key If NOT present display 'Video not on file, switching to create video' Display video description (or number), price. Add all extended price to total. Perform process-money routine. Write order to order-file. Print order confirmation. Return. Process money routine Display total. Get amount. Subtract total from amount giving change. Display change. If change and total = zero, return, else go to process money.
Constraints:	None
Process Number:	2.0
Process Name:	Return Rental
Description:	For each video, Enter video ID Retrieve rental If NO rental, display error message and return. Use Customer ID to retrieve other rentals. Display entire rental. Move to today's date to return date. If return-date-rental-date > 2 compute late charges display late charges add late charges to total. Create history. If new rentals, return else call process money routine.
Constraints:	None

Data Dictionary Contents—Attributes

Attributes, or **fields**, are facts about an entity. Attribute definitions are tedious and tend to be overdocumented unless you are using a CASE tool. As you can see from Table 7-10, there is a large amount of information about attributes that is needed to fully document them. In organizations with a data administration function, much of the information for the type of attributes used here would already be documented, and you would just copy that documentation.

TABLE 7-5 ABC Process Dictionary Entries (*Continued*)

Process Number:	3.0
Process Name:	Maintain Customer
Description:	If new create new customer else If modify prompt customer ID retrieve customer record get changes and verify rewrite customer else if delete prompt customer ID retrieve customer record prompt "Are you sure you want to delete?" If yes, delete customer else else if query call query routine. Return.
Constraints:	None
Process Number:	4.0
Process Name:	Maintain Video
Description:	If new create new video else If modify prompt video ID retrieve video record get changes and verify rewrite video else if delete prompt video ID retrieve video record prompt "Are you sure you want to delete?" If yes, delete video else else if query call query routine. Return.
Constraints:	None
Process Number:	5.0
Process Name:	Create EOD Report
Description:	Read rental file count today's rentals total today's rental receipts Read cash register count today's returns count today's late returns total today's late fees count today's rentals total today's rental receipts Format and print end-of-day summary report.
Constraints:	None

ABC Example Data Dictionary—Attributes

As the two examples provided in Table 7-11 show, the contents get quite long and take quite a bit of paper. In the interest of saving a few trees, and keeping the dictionary useable, when using a paper dictionary, capture only the essential information about attributes and put it in a short-form attribute table as shown in Table 7-12. Essential information is usually the user name, system name, data type, data length, and edit rules. If there is other

TABLE 7-6 Data Dictionary Data Store Contents

Data Store Name	Volume
Aliases	Percent change per cycle
Definition	Frequency of cycle (e.g., as occurs, daily, weekly, etc.)
Data Attributes (Contents in normalized form)	Growth percentage per year
Data Structure (e.g., relation, hierarchy)	Allowable actions (read, write, or read/write) by process
Organization (e.g., Vsam entry sequenced)	Security access restrictions
Sequence and sequence attributes	Backup/recovery requirements
Size of Relations/Records	Special processing considerations
Primary Key	If in a distributed environment, form of partitioning, schematic showing number/location of replications for each partition.
Alternate Keys	
Index Attributes	

information required, such as security restrictions or cross-reference file names, you would add it for that attribute but not all of the others. The short form is used in this text to document ABC's attributes.

AUTOMATED SUPPORT TOOLS

Structured analysis and process methods, in general, are the oldest and most widely used methods. Because they are most widely used, a large number of CASE tools to support structured analysis are available on the market. All of the tools support DFDs; all have a dictionary (although they are not all 'active'). A table of representative CASE tools supporting structured analysis is listed below in Table 7-13.

If you did not get the impression that CASE tools represent a 'buyer beware' situation, perhaps some comments from a recent survey will prove that it is. Data flow diagrams in 12 CASE environments were compared on DFD correctness checking.[11] The authors developed 19 rules by which automated DFDs might be evaluated. The most checked by any of the CASE tools evaluated was 13 (by two CASE tools); the least rules checked was three; the average was eight. The extent of intelligence in CASE obviously varies and is inconsistent with the collective wisdom about how DFDs should be developed and drawn.

Thus, there are many CASE tools available which 'support' structured analysis. The tools vary widely in the diagrams supported and in the extent to which rules about developing DFDs and other diagrams are enforced.

SUMMARY

Process-oriented structured analysis originated in the work of DeMarco, Gane and Sarson, and Yourdon. In structured analysis, we first define the application context then follow a top-down approach to progressively more detailed levels of process analysis. The application is documented

11 See Vessey, Jarvenpaa, & Tractinsky [1992].

(*Text continues on page 274*)

TABLE 7-7 ABC Data Store Contents

Data Store Name:	Customer File	Data Store Name:	Rental File
Aliases:	None	Aliases:	None
Definition:	A computer file of information about customers who are allowed to rent from ABC.	Definition:	A computer file of rental orders outstanding. When a rental is made, it is added to the file. When it is returned, if there are no late fees, it is removed. If there are late fees, the rental stays on file until the late fees are paid.
Data Attributes:	Customer Phone = [Area code + exchange + number] + Customer Last Name + Customer First Name + Customer Address line 1 + Customer Address line 2 + Customer City + Customer State + Customer Zip+4 + Credit Card Type + Credit Card Number + Credit Card Expiration Date + Date of entry	Data attributes:	Customer Phone + Customer Last Name + Customer First Name + Rental Date + Video ID + Video Title + Date Due + Date Returned + Rental Price + Late Fees
Data Structure:	Relational	Data Structure:	Relational
Organization:	Random	Organization:	Random
Sequence:	Entry	Sequence:	Entry
Sequence Attributes:	N/A	Sequence Attributes:	
Record Size:	198 Bytes decompressed	Size:	134
File Size:		Primary Key:	Customer Phone + Video ID
Primary Key:	Customer Phone	Alternate Keys:	
Alternate Keys:	Address line 1	Index Attributes:	Customer Last Name, Customer Phone, Video ID, Customer Phone+Video ID, Video Title
Index Attributes:	Customer last name, Customer zip, Credit Card Number, Address line 1	Volume:	
Volume:		Percent Change:	
Percent Change:		Cycle Frequency:	
Cycle Frequency:		Growth:	
Growth:		Allowable actions by process:	Rental = Add, Change, Read Return = Change, Delete, Read
Allowable actions by process:		Security Access:	
Security Access:		Backup/Recovery:	
Backup/Recovery:		Special processing:	
Special processing:			

TABLE 7-8 Data Dictionary Data Flow Contents

Data Flow Name
Aliases
Timing (e.g., as occurs, daily, weekly, etc.)
Contents
Constraints (e.g., requires 5-second response; only occurs for sales orders, etc.)
Source

TABLE 7-9 ABC Data Flow Dictionary Entries

Data Flow Name:	New Customer
Aliases:	None
Timing:	As occurs
Contents:	*Customer Phone* = [Area code + exchange + number] + Customer Last Name + Customer First Name + Customer Address line 1 + Customer Address line 2 + Customer City + Customer State + Customer Zip+4 + Credit Card Type + Credit Card Number + Credit Card Expiration Date + Date of entry
Constraints:	None
Source:	Customer

Data Flow Name:	Rental
Aliases:	Rental Information
Timing:	As Occurs
Contents:	[Customer Phone | Customer Name] + 1{[Video IS | Video Name]}m
Constraints:	None
Source:	Customer

Data Flow Name:	Payment
Aliases:	Money
Timing:	One per complete rental transaction
Contents:	Total Paid
Constraints:	None
Source:	Customer

Data Flow Name:	Copy of Order
Aliases:	Printed Rental Order
Timing:	One per complete rental transaction
Contents:	= Rental
Constraints:	None
Source:	System

Data Flow Name:	Return
Aliases:	Video Return
Timing:	As Occurs
Contents:	Video ID + (Customer Phone)
Constraints:	None
Source:	Customer

Data Flow Name:	Late Fee Payment
Aliases:	None
Timing:	As Occurs
Contents:	Total Late Fee
Constraints:	May be included within rental payment
Source:	Customer

TABLE 7-10 Data Dictionary Attribute Contents

Attribute User Name	Primary Data Store
System Name	Other files where stored
Aliases	Flows where used
Attribute Definition	Edit/Validation Rules
Data Type	Validation Method (e.g., cross-reference file, code check, etc.)
Data Length	Security access restrictions
Allowable values and meanings	Special processing considerations
Creating Process(es)	

TABLE 7-11 Sample ABC Attribute Dictionary Entries

User Name:	Customer Phone	User Name:	Video ID
System Name:	CPhone	System Name:	Video ID
Aliases:	None	Aliases:	None
Attribute Definition:	The customer's phone number	Attribute Definition:	The numeric identifier for a specific videotape. Uniquely identifies a copy of a group of tapes with the same title.
Data Type:	Numeric	Data Type:	Numeric
Data Length:	10, Area code (3), exchange (3), and number (4)	Data Length:	15
Allowable values and meanings:	Numeric	Allowable values and meanings:	Numeric
Creating Process(es):	Add custor	Creating Process(es):	4.1 Create video
Primary Data Store:	Customer	Primary Data Store:	Video File
Other Files:	Rental File	Other Files:	Rental File
Flows:	New rental order Customer record Rental Return rental information	Flows:	Video Information, Rental Information, Return rental information, New Rental Order
Edit/Validation:	Numeric	Edit/Validation:	Numeric
Validation Method:	Software check	Validation Method:	Software check
Security Access:	None	Security Access:	None
Special processing:	None	Special processing:	None

TABLE 7-12 ABC Attributes—Short Form Dictionary

User Name	System Name	Data Type	Length	Edit/Validation Rules
Customer Phone	CPhone	N	10	Must be present, Check for numeric
Customer Last Name	CLast	A	50	Must be present, Check for alpha
Customer First Name	CFirst	A	25	Must be present, Check for alpha
Customer Address Line 1	CLine1	A/N	50	Must be present
Customer Address Line 2	CLine2	A/N	50	None
Customer City	City	A	30	Must be present, Check for alpha
Customer State	State	A	2	Post Office Abbreviation
Customer Zip	Zip	N	10	Must be present, numeric
Credit Card Type	CCType	A	1	A=AmExpress V=Visa M=Mastercard
Credit Card Number	CCNo	N	17	Must be present, numeric
Credit Card Expiration Date	CCExp	N	8	Valid Date, Format YYYYMMDD
Date of Entry	EntryDate	N	8	Valid Date, Format YYYYMMDD
Credit Rating	CCredit	A	1	0 = OK, 1 = not OK

via graphical forms including a context diagram, leveled set of data flow diagrams, a data dictionary, and, optionally, a state-transition diagram. Diagram symbols and their meanings include (1) circle, entire application; (2) square, external entity; (3) rounded vertical rectangle, process; (4) open ended rectangle, data store, and (5) directed arrow, data flow. Each diagram symbol has a formal definition that is documented in a data dictionary. DFDs identify processes and the flow of data through those processes to achieve some business function. DFDs start at a high level of abstraction to summarize the processing taking place. At successively more detailed levels, procedural and data are added to describe the processing in more detail. Graphical representation replaces much of the text, but does not completely replace text descriptions of individual processes. The data dictionary (or repository) is used to maintain definitions of all DFDs and other analysis information, including files, fields, flows, and external entities, in addition to processes.

The reasoning process for defining the application context and the detailed levels of data flow diagrams was presented. The definitions and contents of data dictionary entries were described. All diagrams and dictionary entries were developed using the ABC rental processing application to show variations and nuances in the thought processes.

TABLE 7-13 CASE Support for Structured Analysis

Product	Company	Technique
Analyst/Designer Toolkit	Yourdon, Inc. New York, NY	Context Diagram Data Flow Diagram (DFD) State-Transition Diagram
Anatool	Advanced Logical SW Beverly Hills, CA	DFD Structured English
Deft	Deft Ontario, Canada	DFD
Design/1	Arthur Anderson, Inc. Chicago, IL	DFD Warnier-Orr Diagram
The Developer	ASYST Technology, Inc. Napierville, IL	DFD Matrix Diagram (for decision tables and real-time systems)
Excelerator, Telon	Intersolv Cambridge, MA	DFD State-Transition Diagram Matrix graph (for real-time systems)
IEW	Knowledgeware Atlanta, GA	DFD Database diagram
MacAnalyst, MacDesigner	Excel Software Marshalltown, IA	DFD Decision Table State Transition Diagram Structured English
Maestro	SoftLab San Francisco, CA	DFD
MetaSystem Tool Set	Meta Systems Ann Arbor, MI	DFD

(*Continued on next page*)

TABLE 7-13 CASE Support for Structured Analysis, *Continued*

Product	Company	Technique
Multi-Cam	AGS Management Systems King of Prussia, PA	DFD State-Transition Diagram Matrix graph (for real-time systems)
PacBase	CGI Systems, Inc. Pearl River, NY	Context Diagram DFD
ProKit Workbench	McDonnell Douglas St. Louis, MO	DFD
ProMod	Promod, Inc. Lake Forest, CA	DFD State-Transition Diagram
Silverrun	Computer Systems Advisers, Inc. Woodcliff Lake, NJ	User-Controlled Modeling
SW Thru Pictures	Interactive Dev. Env. San Francisco, CA	Data Structure DFD State Transition Diagram
System Engineer	LBMS Houston, TX	DFD
Teamwork	CADRE Tech. Inc. Providence, RI	Decision Table DFD State Transition Diagram
Transform	Transform Logic Corp. Scottsdale, AZ	Uses ProKit, Excelerator
Visible Analyst	Visible Systems Corp. Newton, MA	DFD
vs Designer	Visual Software Inc Santa Clara, CA	DFD Ward-Mellor Diagram for real-time systems

REFERENCES

Curtis, B., M. I. Kellner, and J. Over, "Process modeling," *Communications of the ACM*, Vol. 35, #9, September 1992, pp. 75–90.

DeMarco, Tom, *Structured Analysis*. New York: Yourdon Press, 1979.

Frances, B., "A window into CASE," *Datamation*, March 1, 1992, pp. 43–44.

Gane, C., and T. Sarson, *Structured Systems Analysis: Tools and Techniques*. Englewood Cliffs, NJ: Prentice-Hall, 1979.

Gane, Chris, *Computer-Aided Software Engineering: The Methodology, The Products and the Future*. Englewood Cliffs, NJ: Prentice-Hall, 1990.

Krasner, J., J. Terrel, A. Lindhan, P. Arnold, and W. H. Ett, "Lessons learned from a software process modeling system," *Communications of the ACM*, Vol. 35, #9, September 1992, pp. 91–100.

Lee, T., "Bridging the CASE/OOP gap," *Datamation*, March 1, 1992, pp. 63–64.

Lindholm, E. "A world of CASE tools," *Datamation*, March 1, 1992, pp. 75–81.

Martin, James, *Systems Design from Provably Correct Constructs*. Englewood Cliffs, NJ: Prentice-Hall, 1985.

McClure, C., *The Three R's of Software Automation: Re-Engineering, Repository and Reusability*. Englewood Cliffs, NJ: Prentice-Hall, 1992.

McMenamin, Stephan M., and John F. Palmer, *Essential Systems Analysis*. NY: Yourdon Press, 1984.

Slater, D., "PacBase, IEF lead rising CASE satisfaction," *Computerworld*, August 3, 1992, p. 81.

Sullivan, Louis, "The tall building artistically considered," *Lippincott's Magazine*, March 1896.

Vessey, I., S. Jarvenpaa, and N. Tractinsky, "Evaluation of vendor products: CASE tools as methodology companions," *Communications of the ACM*, Vol. 35, #4, April 1992, pp. 90–105.

Yourdon, Edward, *Modern Structured Analysis*. Englewood Cliffs, NJ: Prentice-Hall, Yourdon Press, 1989.

KEY TERMS

attribute
balancing
bottom-up
cohesion
completeness checking
consistency checking
context
context diagram
correctness checking
coupling
cross reference file
data attribute
data dictionary
data flow
data flow diagram (DFD)
data store
direct identification
elementary components
external entity
field
file
function
functional sequence
level 0 DFD
level 1 . . . n DFD
leveled set of DFDs
net inflows and outflows
outward-in
primitive level
process
process description
pseudo-code
quality assurance
structured decomposition
structured English
structured systems analysis
systems model
systems theory
top-down

EXERCISES

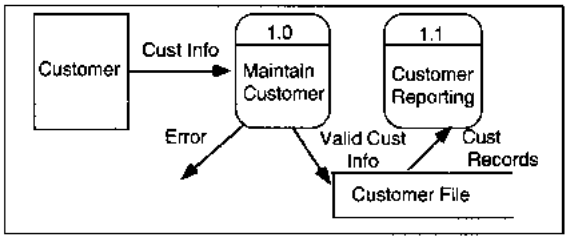
1. Complete the level 1 DFD for 2.0 Rental return process and discuss it in class. Compare several of the answers. Are they the same? Why, or why not?
2. Make a list of outstanding and deferred issues to discuss with Vic.

The next three questions have bothered my students for several years. For each question, identify and discuss the issues and ramifications of each decision, technical issues, user issues, legal or other issues.

3. How should customers be identified to the application? What are the security issues? What are the bureaucracy issues? Is there a way to 'minimize bureaucracy' and still have good security?
4. Should late fees relate to a person or a tape or a rental? What are the issues? How do you decide? Can Vic be helpful in deciding this issue?
5. Where should history get created—at tape rental time? or at tape return time? Can Vic be helpful in deciding this issue? How do you decide?

STUDY QUESTIONS

1. Define the following terms:
   balancing — external entity
   context — function
   data flow — net inflow
   data store — top-down
   direct identification
2. How do you define the scope of a project? Who should define the scope?
3. What is a leveled set of DFDs? How do you know you have that?
4. Why is the strategy of using net inflows and outflows from the previous level of DFD as a starting point for a new level of detail a good idea?
5. Is structured process analysis more like analyzing with a zoom feature on a set of photos or more like analyzing a geologic formation?
6. Define structured decomposition. Why do you use this technique?
7. What is the purpose of the data dictionary?
8. Discuss the reasoning process used in structured analysis. Does it guarantee that everyone will get the same analytical result? If not, why not?
9. How might the process of structured analysis be improved to be more rigorous, i.e., guarantee the same results regardless of who performs the analysis?
10. Evaluate the following diagram. What type of diagram is it? What is its purpose? Label errors and list all reasons why they are wrong. Redraw the diagram correctly.



11. What are the major diagrams in the analysis phase? How are they derived?
12. List and briefly describe the five approaches to identifying processes.
13. Describe all data dictionary entries and give an example of each.
14. Why might CASE tools be useful in structured analysis?
15. Draw and identify five common DFD errors and their corrections.
16. Discuss the three types of quality checks done on DFDs.

★ EXTRA-CREDIT QUESTIONS

1. The example used in Figures 7-15 through 7-20 refers to a psychiatric clinic and processing performed for Medicaid claim processing. Perform a structured analysis of this problem as described in the Appendix Case: The Child Development Clinic. Refer to the figures in the text to help you if you get stuck.
2. Perform a structured analysis of any of the problems in the Appendix. Decide what information in the problem description is relevant to an automated application. Then, build a context diagram, a levels set of DFDs and a data dictionary.

CHAPTER 8

PROCESS-ORIENTED DESIGN

INTRODUCTION

Structured design is the art of designing system components and the interrelationships between those components in the best possible way to solve some well specified problem. The main goal of design is to map the functional requirements of the application to a hardware and software environment. The results of structured design are programming specifications and plans for testing, conversion, training, and installation. In addition, the design may result in prototyping part or all of the application. This section discusses the mapping process and the development of program specifications. The other topics are discussed in Chapter 14.

The goals of structured design, as first documented by Yourdon and Constantine [1979], have not changed much over the years. They are to minimize cost of development and maintenance. We can minimize the cost of development by keeping parts manageably *small and separately solvable*. We can minimize the cost of maintenance by keeping parts manageably small and separately *correctable*. In design we determine the smallest solvable parts as a way of managing application complexity.

Conceptual Foundations

The concept 'form follows function' that informed analysis is again the basis for structured design. The application processes determine the form of the application. The divide and conquer principle guides the definition of the smallest solvable parts while keeping the general goals of maintainability and low cost in mind. Partitioning and functional decomposition are the basic activities used in dividing processes into modules. The basic input-process-output (IPO) model from the DFD results in a structure chart that adds a control component to the IPO model (see Figure 8-1).

Principles of good structured design are information hiding, modularity, coupling, and cohesion. **Information hiding** means that only data needed to perform a function is made available to that function. The idea is a sound one: You cannot mess up what you don't have access to. **Modularity** is the design principle that calls for design of small, self-contained units that should lead to maintainability. Following systems theory, each module should be a small, self-contained system itself. **Coupling** is a measure of intermodule connection with minimal

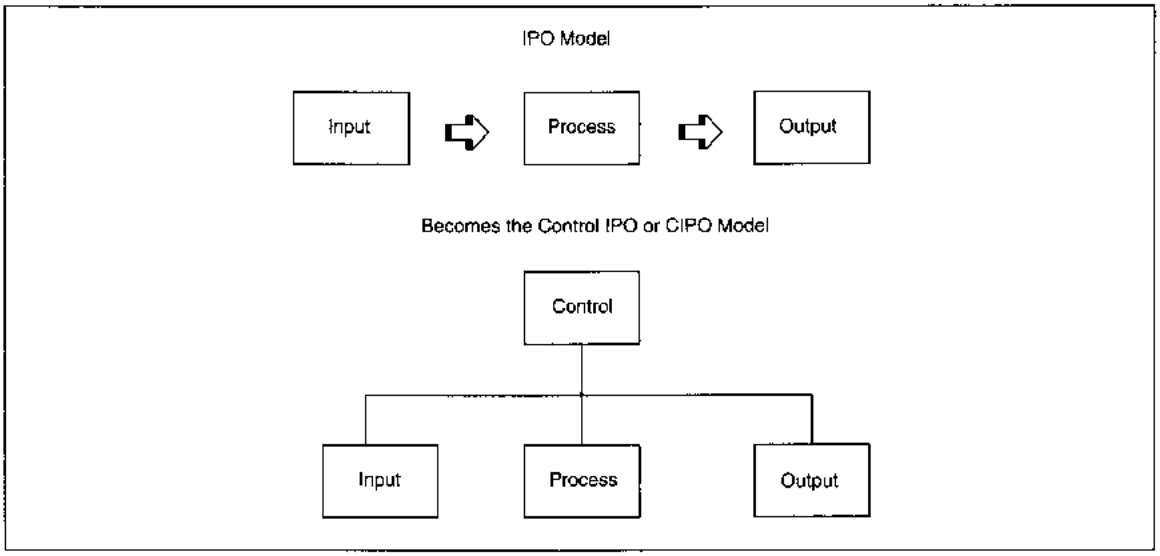


FIGURE 8-1 Input-Process-Output Model and Structure Chart

coupling the goal (i.e., less is best). **Cohesion** is a measure of internal strength of a module with the notion that maximal, or functional, cohesion is the goal. These principles are related to the process of design in the next section.

DEFINITION OF STRUCTURED DESIGN TERMS

The major activities of structured design are:

1. Transform or transaction analysis of DFD
2. Refine and complete structure chart
3. Identify load units and program packages
4. Define the physical database
5. Develop program specifications

The terms associated with each of these activities are defined in this section and summarized in Table 8-1.

In design we partition the application to divide subprocesses into codifiable program modules. **Partitioning** is the divide and conquer strategy by which we divide existing subprocesses from the DFD into groups for implementation. The two methods of partitioning used are transform analysis and transaction analysis.

DFD processes transform data from one form to another; these transformations will eventually be automated by programs each containing several modules. **Transform analysis** is the process of identifying the clusterings of subprocesses based on their major functions. The functions are either input, output, or transform-oriented. The input-oriented processes are called **afferent flows**. **Afferent** means bringing inward to a central part. Afferent processes read data and prepare it for processing. The output-oriented processes are called **efferent flows**, where **efferent** means moving away from the central part. Efferent processes write, display, and print data. The remaining processes are collectively called the **central transform**. The central transform processes have as their major function the change of information from its incoming state to some other state.

An example of a data flow diagram with its afferent and efferent flows and its central transform identified is shown in Figure 8-2. Notice that multiple afferent or efferent flow streams may be found.

TABLE 8-1 Structured Design Concept Definitions

Term	Definition
Stepwise refinement	The process of defining functions that will accomplish a process; includes definition of modules, programs, and data
Program morphology	The shape of a program, including the extent of fan-out, fan-in, scope of control, and scope of effect
Data structure	The definition of data in an application includes logical data definition and physical data structure
Modularity	A property of programs meaning they are divided into several separate addressable elements
Abstraction	Attention to some level of generalization without regard to irrelevant low-level details
Information hiding	Design decisions in one module are hidden from other modules
Cohesion	A measure of the internal strength of a module
Coupling	A measure of the intermodule strength of a module

The streams are partitioned off from the rest of the diagram by drawing arcs showing where they end.

Examples of transform-centered applications include accounting, personnel, payroll, or order entry-inventory control. For these applications, getting data into and out of the system is secondary to the file handling and manipulation of numbers that keep track of the information. In accounting, for instance, balancing of debits and credits takes place at end-of-day, end-of-month, and end-of-year processing. These periodic process transfor-mations summarize and move data, erase some information, archive other information, and write data to the general ledger to summarize the details in the receivables and payables subledgers. All of these *transforms* process data that is already in the files. These processes are the *heart* of accounting processing. Without these processes, the application would be doing something else.

Not all applications are transform-centered. Some applications do simple processing but have many different transaction types on which the simple processes are performed. These systems are called **transaction-centered. Transaction analysis** replaces transform analysis for transaction-centered applications with partitioning by transaction type, which may not be obvious from DFDs. Figure 8-3 shows an example of a partitioned DFD for a transaction-centered application. This detailed DFD looks like it contains redundancy because many of the same processes appear more than once. Look closely and you see that each set of processes relates to a different type of transaction.

When the high-level partitioning is done, the information is transferred to a first-cut structure chart. We will develop the structure chart from Figure 8-2. A **structure chart** is a hierarchic, input-process-output view of the application that reflects the DFD partitioning. The structure chart contains one rectangle for each lowest level process on the DFD. The rectangles are arranged in a hierarchy to show superior control and coordination modules. Individual process modules are the lowest in their hierarchy. The rectangles in the hierarchy are connected via undirected lines that are always read top-down and left to right. The lines imply transfer of processing from the top to the bottom of the hierarchy. Diamonds overlay the connection when a conditional execution of a module is possible using if-then-else logic. Reused modules are shown in one of two ways. Either they are repeated several times on the diagram and have a slash in the lower left corner to signify reuse, or they are connected to more than one superior module via the linking lines.

The identification of afferent flows, efferent flows, and transforms results in chains of processes, each its own 'net output.' If we look at Figure 8-2 again, we see the net afferent output is data flow *Good Input*. For the central transform, the net output is *Solution*. For the efferent flows, the net output is *Printed Solution*. These net outputs are used to determine the initial structure of the structure chart, using a process called factoring.

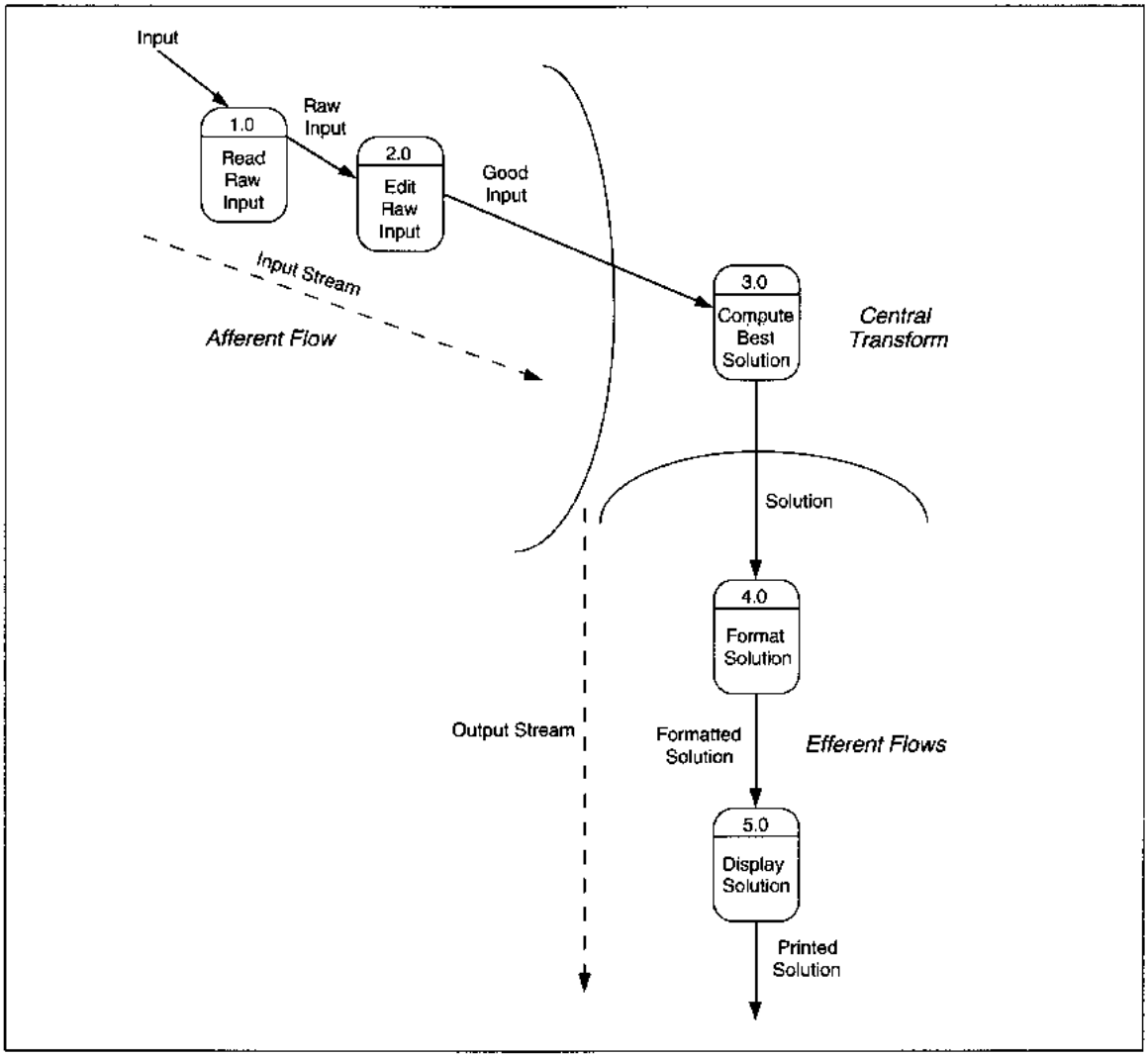


FIGURE 8-2 Transform-Centered DFD Partitioned

Factoring is the process of decomposing a DFD into a hierarchy of program components that will eventually be programmed modules, functions, or control structures. Each stream of processing is analyzed to determine its IPO structure. When the structure is identified, the processes are placed on the structure chart and named until all low-level DFD processes are on the structure chart (see Figure 8-4).

Next, data and control information are added to the structure chart. **Data couples** identify the flow of data into and out of modules and match the data flows on the DFD. Data couples are identified by a directed arrow with an open circle at the source end (see Figure 8-5). The arrowhead points in the direction the data moves.

Control couples identify the flow of control in the structure. Control couples are placed to show where the control data originates and which module(s) each couple affects. A control couple is usually a program switch whose value identifies how a module is activated. Control couples are drawn as

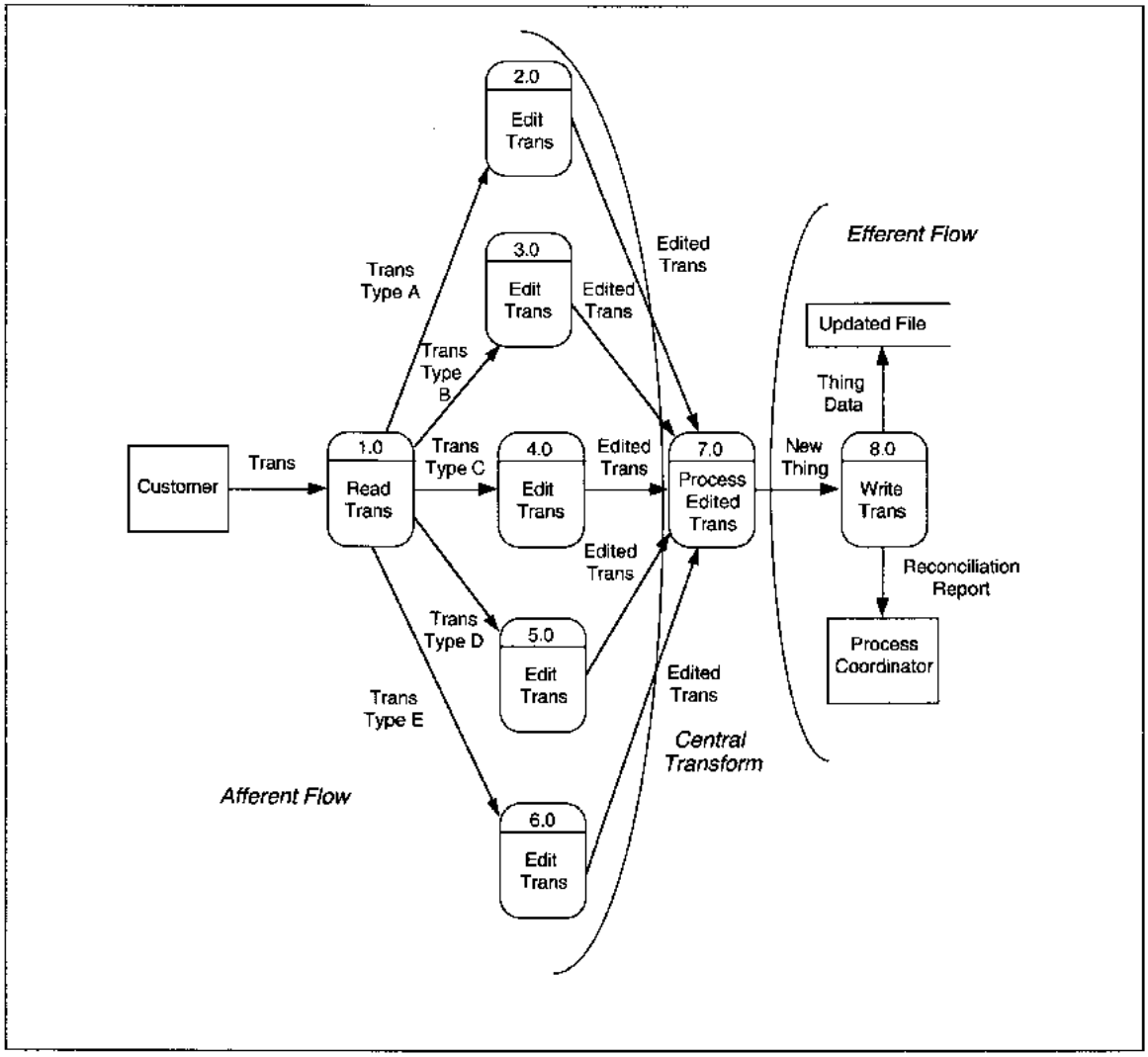


FIGURE 8-3 Transaction-Centered DFD Partitioned

directed arrows with a closed circle at the source end (see Figure 8-6). The arrowhead points in the direction the control travels. If a control couple is in, set and reset in the same module, it *is not* shown on the diagram. A control couple that is set and reset in one place, but used in another, is shown. If a control couple is set in one module and reset in another, it *is* shown as both input and output. Control is 'designed into' the application by you, the SE, based on the need for one module to control the processing of another module. The goal is to keep control to a minimum. Figure 8-4 shows the completed structure chart for the DFD in Figure 8-2.

Next, we evaluate and revise the structure chart to balance its morphology. **Morphology** means form or shape. The shape of the structure chart should be balanced to avoid processing bottlenecks. Balance is determined by analyzing the depth and width of the hierarchy, the skew of modules, the span of control, the scope of effect, and the levels of coupling and cohesion. When one portion of the structure chart is unbalanced in relation to the rest of the diagram, you

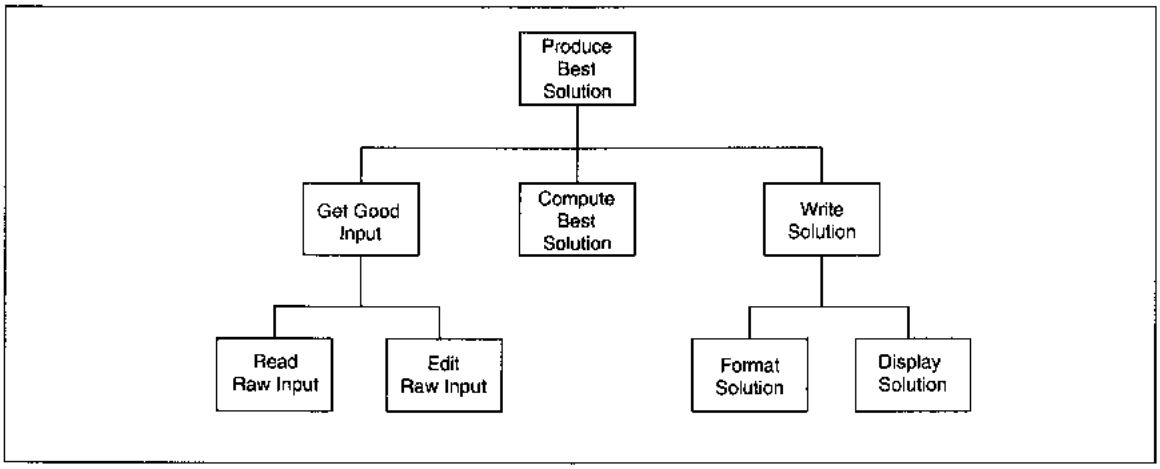


FIGURE 8-4 First-Cut Structure Chart

modify the structure to restore the balance, or pay closer attention to the unbalanced portion to ensure an efficient production environment.

The **depth of a hierarchy** is the number of levels in the diagram. Depth by itself is not a measure of good design nor is it a goal in itself. Rather, it can indicate the problem of too much communication overhead and not enough real work taking place (see Figure 8-7). Conversely, adding a level of depth can be a cure for too wide a hierarchy.

The **width of the hierarchy** is a count of the modules directly reporting to each superior, higher-level module (see Figure 8-8). **Span of control** is another term for the number of immediate subordinates and is a synonym for the width of the hierarchy. Width relates to two other terms: fan-out and fan-in. **Fan-out** is the number of immediate subordinate modules. Too much fan-out can identify a processing bottleneck because a superior module is controlling too much processing.

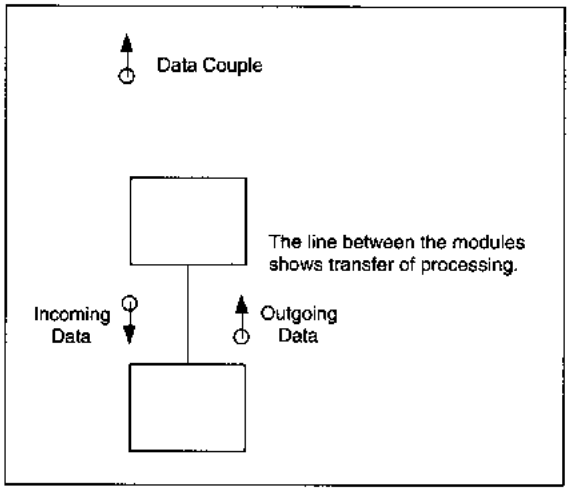


FIGURE 8-5 Data Couple Notation

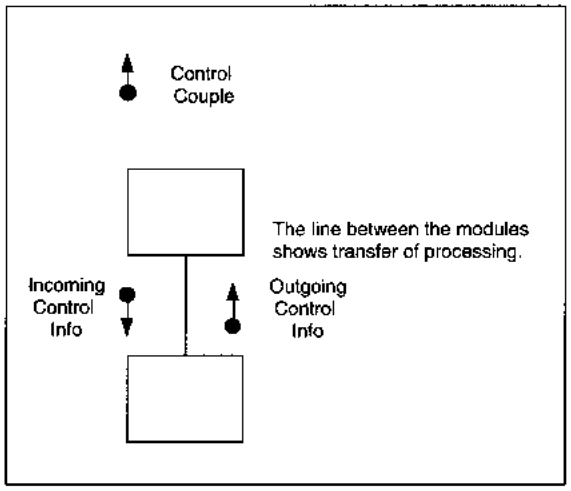


FIGURE 8-6 Control Couple Notation

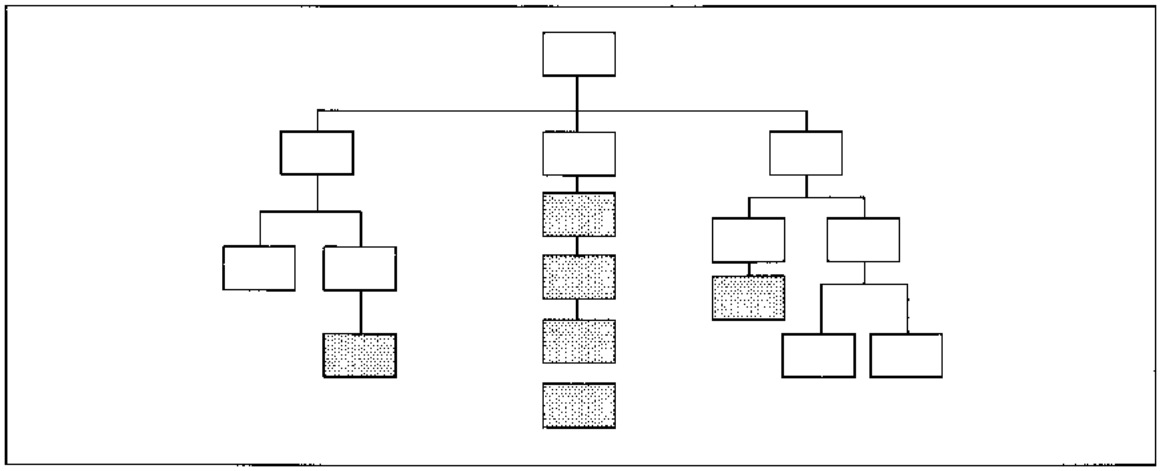


FIGURE 8-7 Excessive Depth of Hierarchy

While there is no one number that says 'how wide is too wide,' seven ±2 is the generally accepted guideline for number of fan-out modules. One solution to fan-out processes that are functionally related is to factor another level of processing that provides middle-level management of the low-level modules. Another solution to fan-out problems that are factored properly, but not functionally related, is to introduce a new control module at the IPO level.

Fan-in, on the other hand, is the number of superior modules (i.e., immediate bosses) which refer to some subordinate module (see Figure 8-9). Fan-in can be desirable when it identifies reusable components and reduces the total amount of code produced. The major tasks with fan-in modules are to ensure that they perform a whole task, are highly cohesive, and are minimally coupled.

Skew is a measure of balance or lopsidedness of the structure chart (see Figure 8-10). Skew occurs

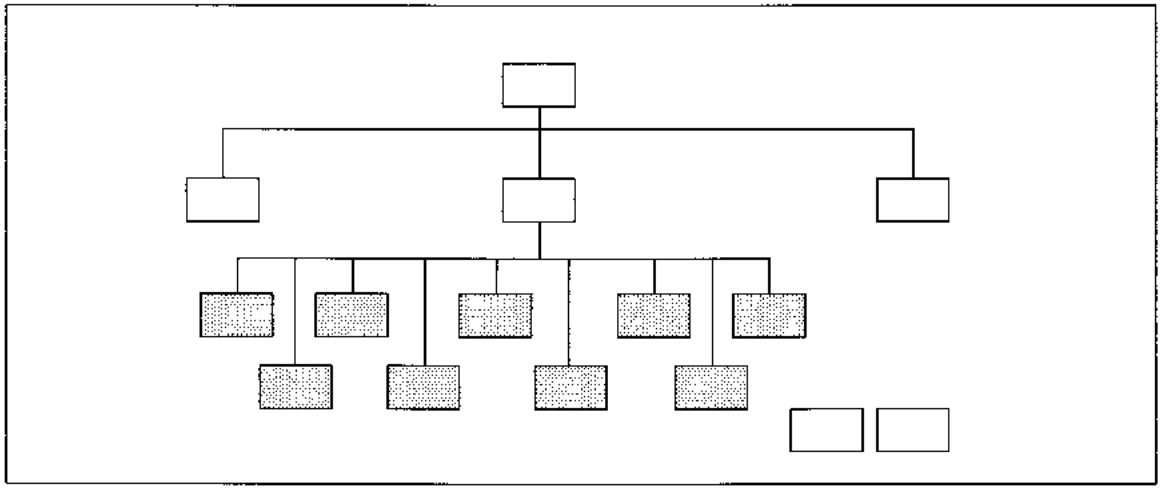


FIGURE 8-8 Excessive Width of Hierarchy

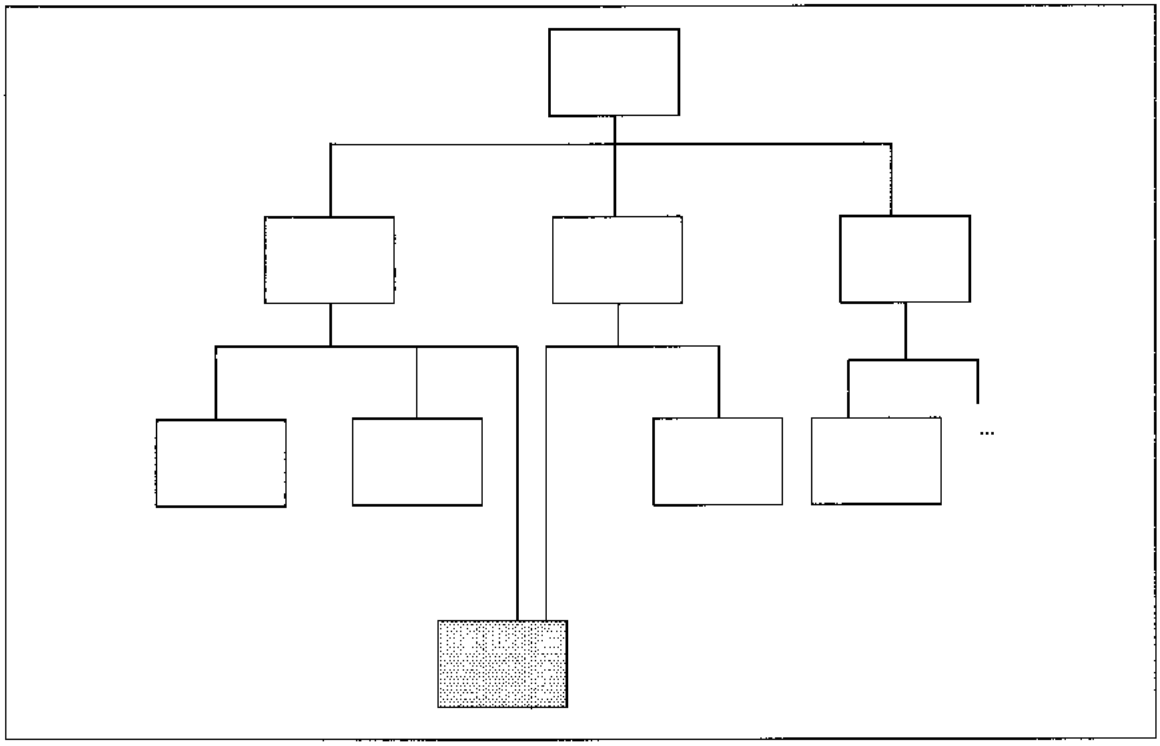


FIGURE 8-9 Example of Fan-In

when one high-level module has many subordinate levels and some or most of the other high-level modules have few subordinate levels. Skew can indicate incorrect factoring. If factoring is correct, then skew identifies a *driver* for the application that might require special consideration. If the skew is on the input side, we say the application is input driven or **input-bound**. Similarly, if the skew is on the output side, the application is **output-bound**. If the input and output are skewed with little transform processing, the application is **I/O-bound** (for input/output). Finally, if the application has little input or output, but lots of processing, the application is **process-bound**. The special considerations of each of these occurrences deal with ensuring correct language selection and meeting I/O and process time constraints.

The **scope of effect** of a module identifies the collection of modules that are conditionally processed based on decisions by that module (see Figure 8-11). The scope of effect can be identified by counting the number of modules that are *directly* affected by the process results of another module. High scope of effect relates to fan-out, fan-in, and coupling in that it may identify potential problems with debugging and change management. Ideally, the scope of effect of any one module should be zero or one. That is, no more than one other module should be affected by any processing that takes place in any other module.

The last measures of structure morphology which are analyzed throughout the remainder of structure design are coupling and cohesion. Cohesion is a measure of the *intramodule* strength. Coupling is a measure of the *intermodule* linkage. Maximal, functional cohesion and minimal coupling are the ideal relationships. Coupling and cohesion are related inversely (see Figure 8-12). If cohesion is high, coupling is low, and vice versa; but, the relationship is not perfect. That means that if you have strong cohesion, you may still have strong coupling due to

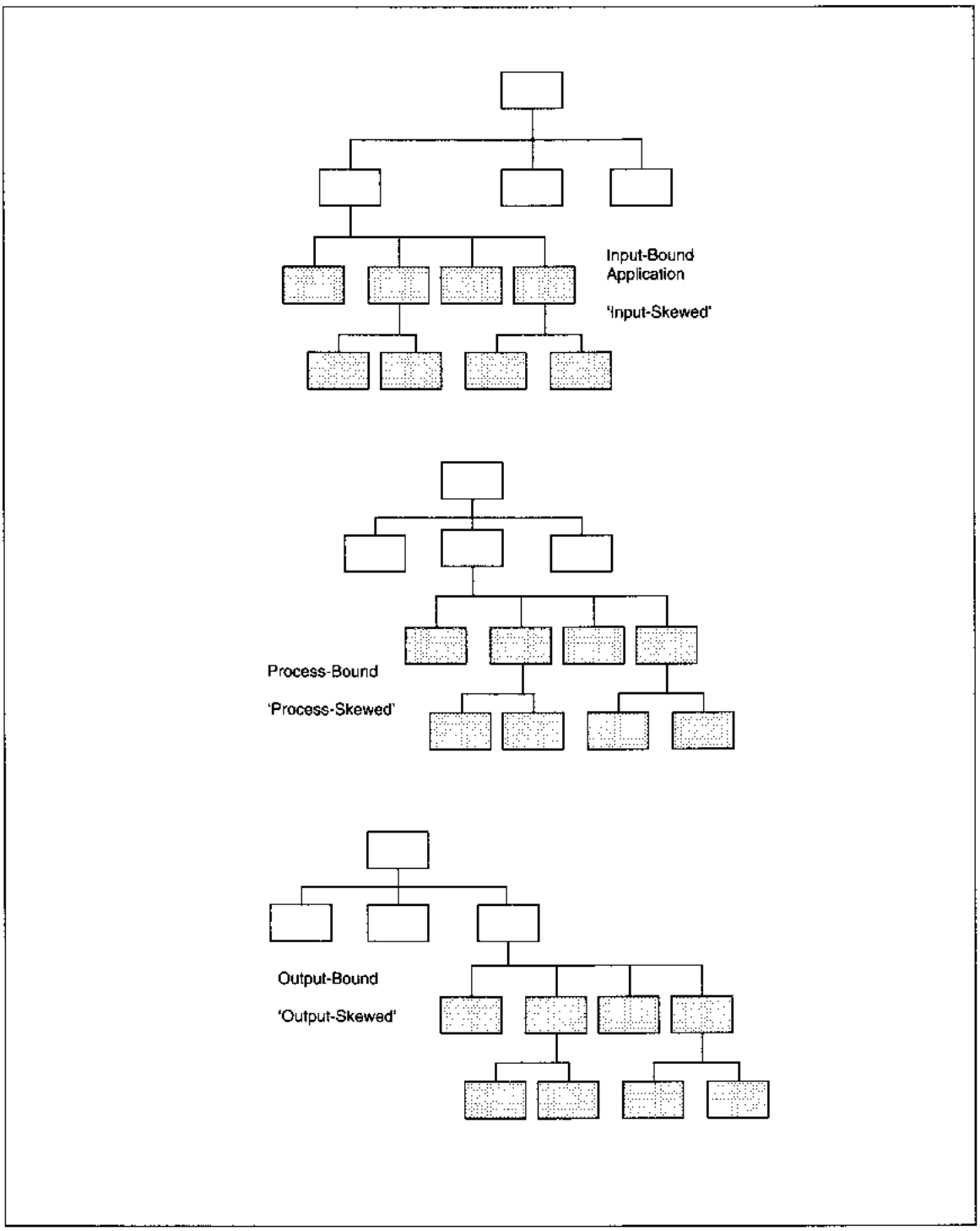


FIGURE 8-10 Examples of Skewed Structure Charts

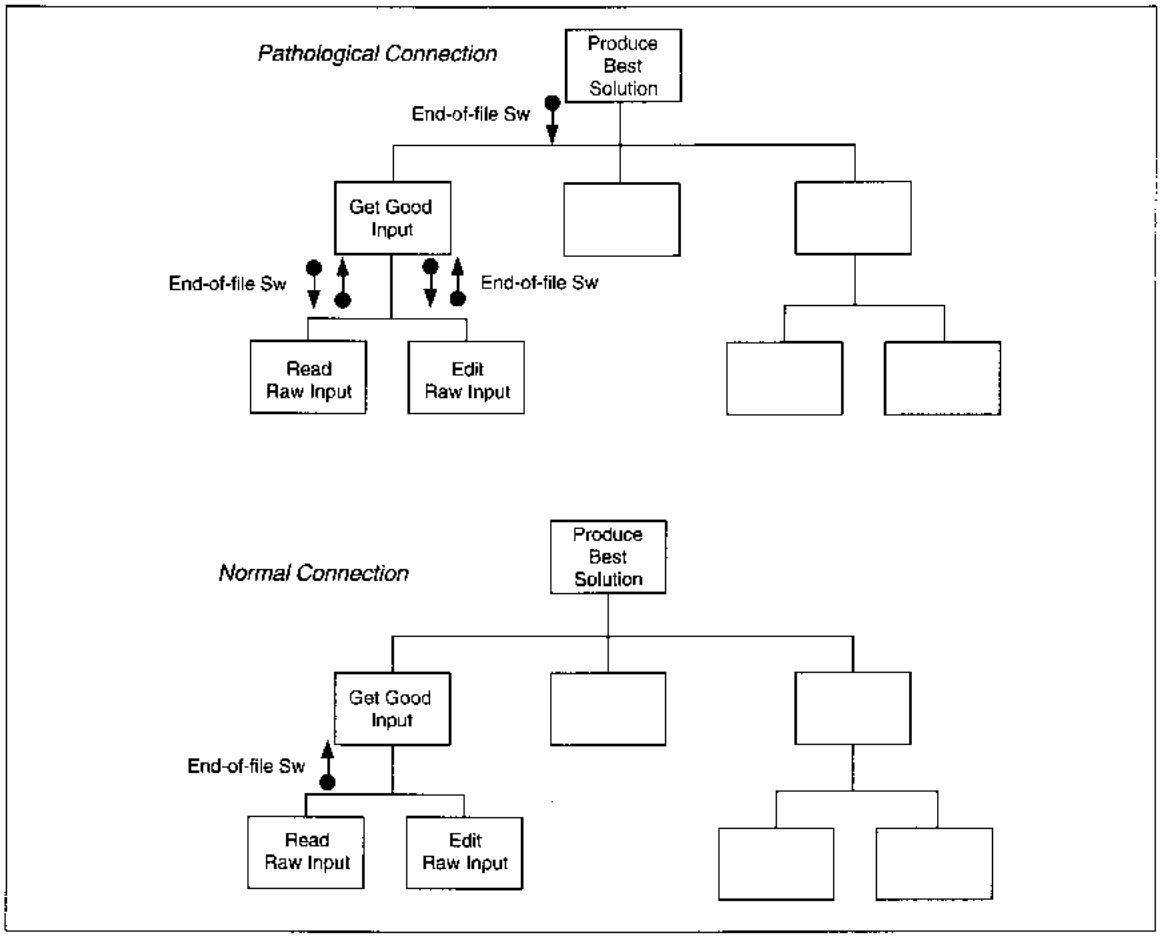


FIGURE 8-11 Example of Scope of Effect

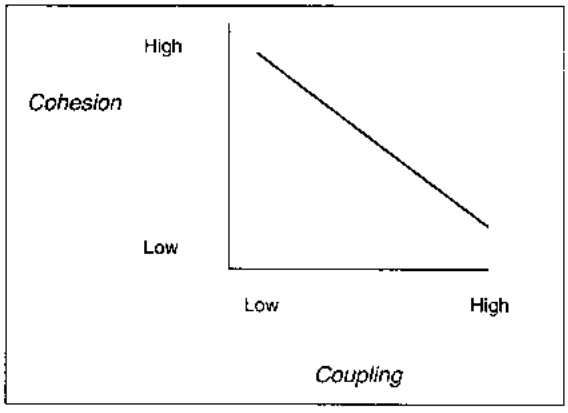


FIGURE 8-12 Relationship between Coupling and Cohesion

poor design. So, attention to both coupling and cohesion are required.

Factoring and evaluation are followed by **functional decomposition**, which is the further division of processes into self-contained IPO subprocesses. Balanced structure chart subprocesses might be further decomposed to specify all of the functions required to accomplish each subprocess. Fan-out, span of control, and excessive depth are to be avoided during this process.[1] The decision whether

1 Some companies have as a local convention (a policy in their company) that a lower-level DFD is developed to describe programmable individual functions before partitioning. This is decomposition at the DFD level and has the same effect as decomposition here.

to decompose further or not relates to the details needed for the implementation language and how well the SEs understand the details.

Structure charts are only one of many methods and techniques for documenting structured design results. Most of the alternatives would replace, rather than supplement, structure charts. Each technique has its own slightly different way of thinking about the processes to finalize a design, even though the goals are the same. Several alternatives are IBM **Hierarchic input-process-output diagrams (HIPO)** (see Figure 8-13), **Warnier diagrams** (see Figure 8-14), **Nassi-Schneiderman diagrams** (see Figure 8-15), and flow charts (see Figure 8-16).

To complete design, program specifications (specifications is abbreviated to 'specs') must be developed, but before specs can be developed, several other major activities are required. First, the physical database must be designed. Then, program package units are decided. Several activities not discussed here (these are covered in Chapter 14) are performed, including verification of adequate design for inputs, outputs, screens, reports, conversion, controls, and recoverability.

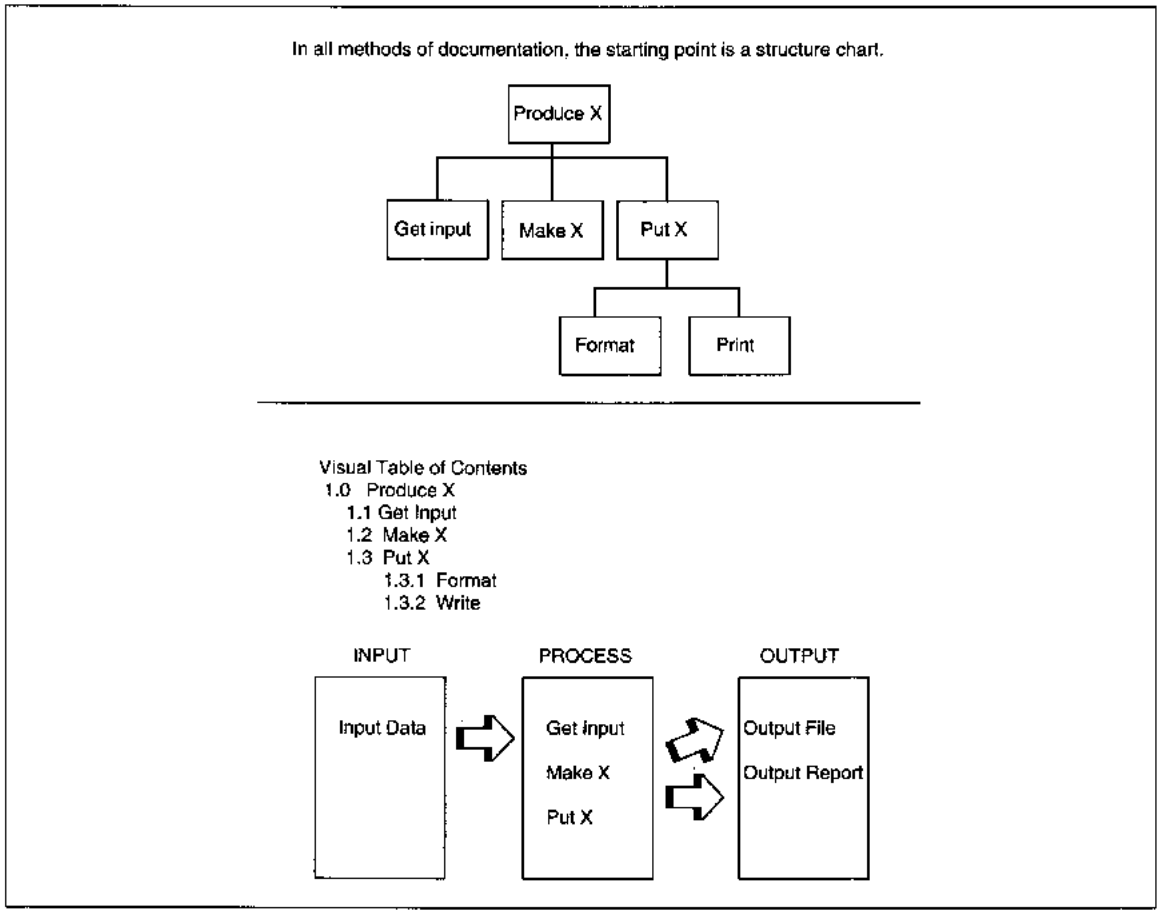
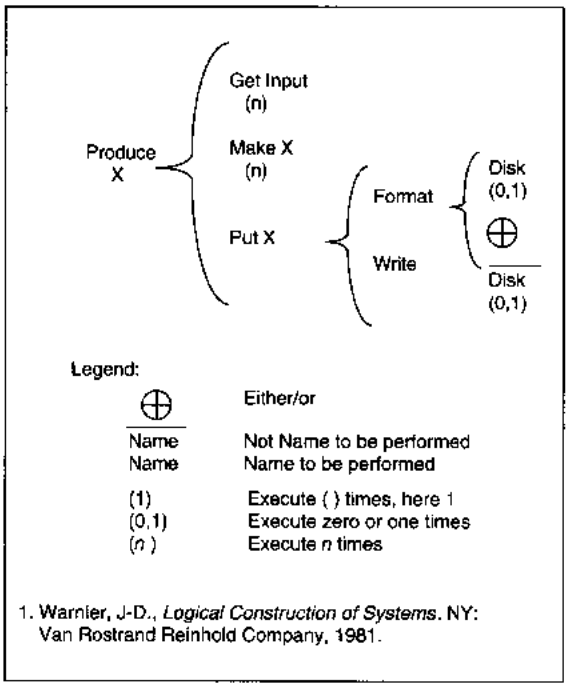


FIGURE 8-13 Other Structured Program Documentation Methods: IBM's Hierarchic Input-Process-Output (HIPO) Diagram Example



1. Warnier, J-D., *Logical Construction of Systems*. NY: Van Rostrand Reinhold Company, 1981.

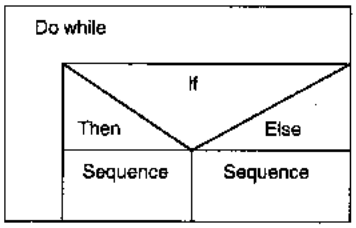
FIGURE 8-14 Warnier Diagram[1]

Physical database design is concurrent with factoring and decomposition. Several common physical database design activities are:

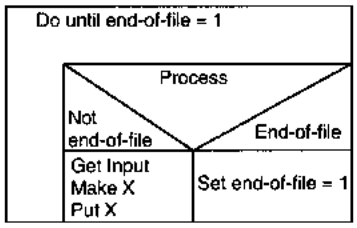
- design user views (if this is not already done)
- select the access method
- map user views to the access method and storage media
- walk-through the database design
- prototype the database
- document and distribute access information to all team members
- train team members in access requirements
- develop a test database
- develop the production database

Keep in mind that many other activities may be involved in designing a physical database for a specific implementation environment.

While the details of physical database design and decomposition are being finalized, project team members are also thinking about how to package the modules into program units. A **program unit** or a **program package** is one or more called modules,



General Form of Nassi-Schneiderman



Example of Nassi-Schneiderman Diagram

2. Nassi, I., and B. Schneiderman, "Flowchart techniques for structured programming," *ACM SIGPLAN Notices*, Vol. 8, #8, August 1973, pp. 12–26.

FIGURE 8-15 Nassi-Schneiderman[2] Diagram Example

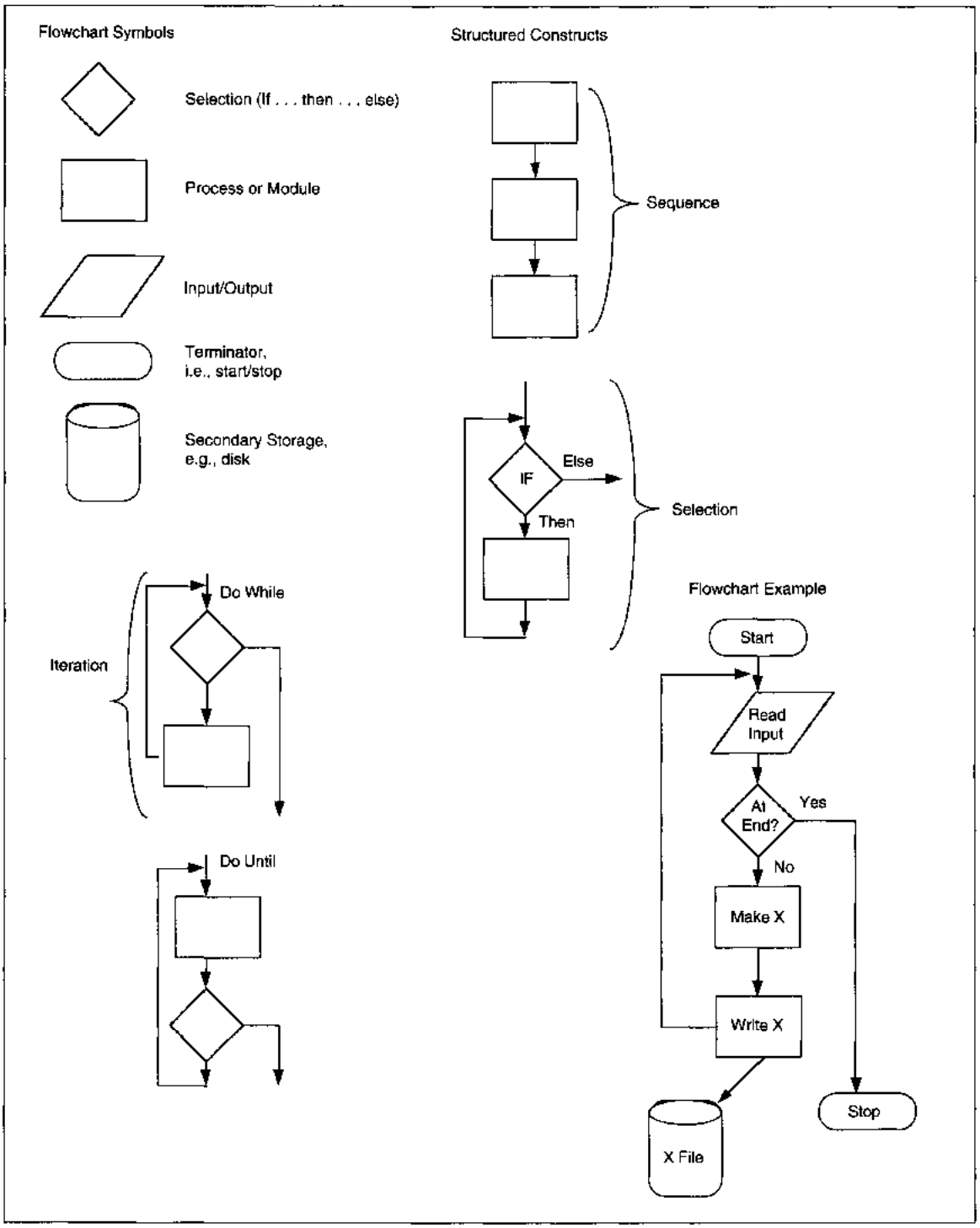


FIGURE 8-16 Flowchart Symbols, Structured Constructs, and Example

functions, and in-line code that will be an execute unit to perform some atomic process. In nonreal-time languages, an execute unit is a link-edited load module. In real-time languages, an execute unit identifies modules that can reside in memory at the same time and are closely related, usually by mutual communication. The guiding principles during these design activities are to minimize coupling and maximize cohesion (see Tables 8-2 and 8-3 for definition of the seven levels of coupling and cohesion).

An **atomic process** is a system process that cannot be further decomposed without losing its system-like qualities. An **execute unit** is a computer's unit of work (i.e., a task). A **module** is a 'small program' that is self-contained and may call other modules. Modules may be in-line, that is, in the actual program, or may be externally called modules. **In-line code** is the structured program code that controls and sequences execution of modules and functions. For instance, a 'read' module might do all file access; a screen interaction module might do all screen processing and have submodules that perform screen input and screen output.

A **function** is an external 'small program' that is self-contained and performs a well-defined, limited procedure. For example, a function might compute a square root of a number. Functions usually do not call other modules but there is no rule against it. Even though the definitions of modules and functions are similar, they are different entities. Functions sometimes come with a language, for instance, the mathematical and statistical functions that are part of Fortran. Modules are usually user-defined and have a broader range of applicability, such as a

TABLE 8-2 Definition of Cohesion Levels

Type of Cohesion	Definition
Functional	Elements of a procedure are combined because they are all required to complete one specific function. This is the strongest type of cohesion and is the goal.
Sequential	Elements of a common procedure are combined because they are in the same procedure and *data* flows from one step to the next. That is, the output of one module, for example, is passed in sequence as input to the next module. This is a strong form of cohesion and is acceptable.
Communicational	Elements of a procedure are combined because they all use the same data type. Modules that all relate to customer maintenance—add, delete, update, query—are related through communication because they all use the Customer File.
Procedural	Elements of a common procedure are combined because they are in the same procedure and *control* flows from one step to the next. This is weak cohesion because passing of control does not mean functions in the procedure are related.
Temporal	Statements are together because they occur at the same time. This usually refers to program modules, for example, 'housekeeping' in COBOL programs to initialize variables, open files, and prepare for processing. Temporal cohesion is weak and should be avoided wherever practical.
Logical	The elements of a module are grouped by their type of function. For instance, all edits, all reads from files, or all input operations are grouped. This is undesirable cohesion and should be avoided.
Coincidental	This is the random or accidental placement of functions. This lowest level of cohesion occurs when there is no real relationship between elements of a module. This is undesirable cohesion and should be avoided.

TABLE 8-3 Definition of Coupling Levels

Level of Coupling	Definition
Indirect relationship	No coupling is possible when modules are independent of each other and have neither a need nor a way to communicate. This is desirable when modules are independent. An example of no direct relationship is a date translate routine and a net present value routine. There is no reason for them to be related, so they should not be related.
Data	Only necessary data are passed between two modules. There are no redundant parameters or data items. This is the desirable form of coupling for related modules.
Stamp	The module is given access to a complete data structure such as a physical data record when it only needs one or two items. The module becomes unnecessarily dependent on the format and arrangement of data items in the structure. Usually, stamp coupling implies external coupling. The presence of unneeded data violates the principal of 'information hiding' which says that only data needed to perform a task should be available to the task.
Control	Control 'flags' are shared across modules. Control coupling is normal if the setting and resetting of the flag are done by the same module. It is a pathological connection to be avoided if practical when one module sets the flag and the other module resets the flag.
External	Two modules reference the same data item or group of items such as a physical data record. In traditional batch applications, external coupling is unavoidable since data are passive and not directly relating to modules. External coupling is to be minimized as much as possible and avoided whenever practical. External coupling violates the principal of information hiding.
Common	Modules have access to data through global or common data areas. This is frequently a language construct problem but it can be avoided by passing parameters with only a small amount of additional work. Common coupling violates the principal of information hiding.
Content	One module directly references and/or changes the *insides* of another module or when normal linkage mechanisms are bypassed. This is the highest level of coupling and is to be avoided.

screen interaction module. Functions are usually reusable across applications without alteration; modules are not.

When program packages are decided, program specifications are developed. **Program specifications** document the program's purpose, process requirements, the logical and physical data definitions, input and output formats, screen layouts, constraints, and special processing considerations that might complicate the program. Keep in mind that the term *program* might also mean a module within a program or an externally called function. There are two parts to a program specification: one identifies interprogram (including programs in other applications) relationships and communication; the other documents intraprogram processing that takes place within the individual program. Another term for interprogram relationships is **interface.**

PROCESS DESIGN ACTIVITIES

The steps in process design are transform (or transaction) analysis, develop a structure chart, design the physical database, package program units, and write

program specifications. Each of these steps is discussed in this section.

Since both transform and transaction analysis might be appropriate in a given system, the first activity is to identify all transactions and determine if they have any common processing. This activity can be done independently from the DFD and functional analysis, or it can be done as a side activity while you are doing functional analysis as the primary activity. If you cannot tell which is more appropriate, do a rough-cut structure chart using both methods and use the one which gives the best overall results in terms of coherence, understandability, and simplicity of design.

Transaction Analysis

Rules for Transaction Analysis

The basic steps in transaction analysis are to define transaction types and processing, develop a structure chart, and further define structure chart elements. A detailed list of transaction analysis activities follows.

1. Identify the transactions and their defining actions.
2. Note potential situations in which modules can be combined. For instance, the action is the same but the transaction is different—this identifies a reusable module.
3. Begin to draw the structure chart with a high-level coordination module as the top of the transaction hierarchy. The coordination module determines transaction type and dispatches processing to a lower level.
4. For each transaction, or cohesive collection of transactions, specify a transaction module to complete processing it.
5. For each transaction, decompose and create subordinate function module(s) to accomplish the function(s) of the transaction. If a transaction has only one unique function, then keep the unique action as part of the transaction module identified in the previous step.
6. For functions that are not unique, decompose them into common reusable modules. Make sure that the usage of the module is identical for all using transactions. Specifically identify which transactions use the module.
7. For each function module, specify subordinate detail module(s) to process whole detail steps as appropriate. If there is only one functional detail step, keep it as part of the function module defined in step 5.

A typical transaction application is money transfer for banks. Transactions for money transfer all have the same information: sending bank, receiving bank, sender, receiver, receiver account number, and amount. There might be other information, but this is required. What makes money transfer a transaction system is that transactions can come from phone, mail, TWX/Telex, fax, BankWire, FedWire, and private network sources. Each source of transaction has a different format. Phone, mail, and fax are all essentially manual so the application can require a person to parse the messages and enter them in one format. The other three are electronic messaging systems to be understood electronically. TWX/telex, which are electronic free-form messages, may have field identifiers but have no required order to the information. A summary DFD for a money transfer system might look like Figure 8-17, which shows a deceptively simple process. What makes the process difficult is that the data entry-parse-edit processes are different for each message type, having different edit criteria, formats, and acceptance parameters. The partitioning for the transaction DFD can be either a high-level summary or detailed. The summary partition (see Figure 8-17) shows afferent flows on the summary DFD, which is annotated that structuring is by transaction type. The detailed DFD (see Figure 8-18) shows each type of transaction with its own set of afferent and efferent flows.

To create a first-cut structure chart, one control module is defined for each transaction's afferent stream and efferent stream; there may be only one transform center. For each transaction, the afferent data flows are used to define data couples. The control couples relate to data passed between modules. When control is within a superior mod-

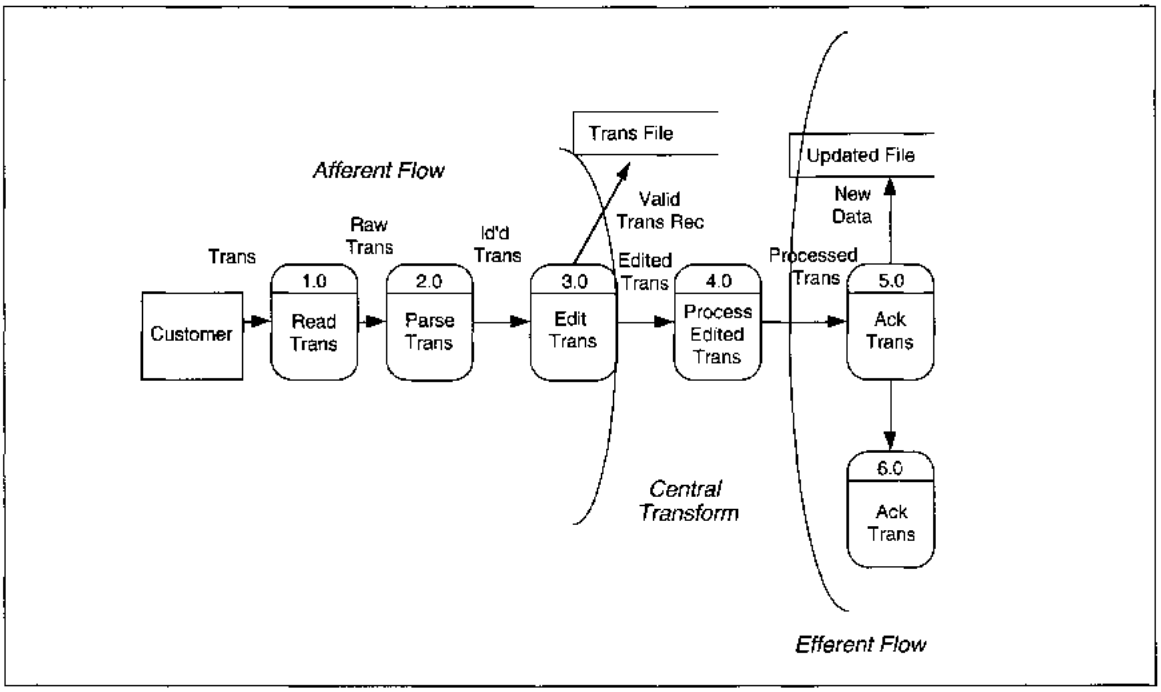


FIGURE 8-17 Summary Money Transfer DFD Partitioned

ule, it is shown via a diamond to indicate selection from among the transaction subprocesses (see Figure 8-19).

ABC Video Example Transaction Analysis

The first step to determining whether you have a transaction application or a transform centered application is to identify all sources of transactions and their types. Table 8-4 contains a list of transactions for ABC Video. As you can see from the list, there are maintenance transactions for customer and video information, there are rental and return transactions, and there are periodic transactions. The only common thread among the transactions is that they share some of the same data. The processing in which they are involved is different and there are no commonalities except reading and writing of files. Therefore, we conclude that ABC Video Rental processing is not a transaction-centered application and move to transform analysis to complete the structure chart.

Transform Analysis

Rules for Transform Analysis

In transform analysis we identify the central transform and afferent and efferent flows, create a first-cut structure chart, refine the chart as needed at this high level, decompose the processes into functions, and refine again as needed. These rules are summarized as follows:

1. Identify the central transform
2. Produce a first-cut structure chart
3. *Based on the design strategy*, decompose the processes into their component activities
4. Complete the structure chart
5. Evaluate the structure chart and redesign as required.

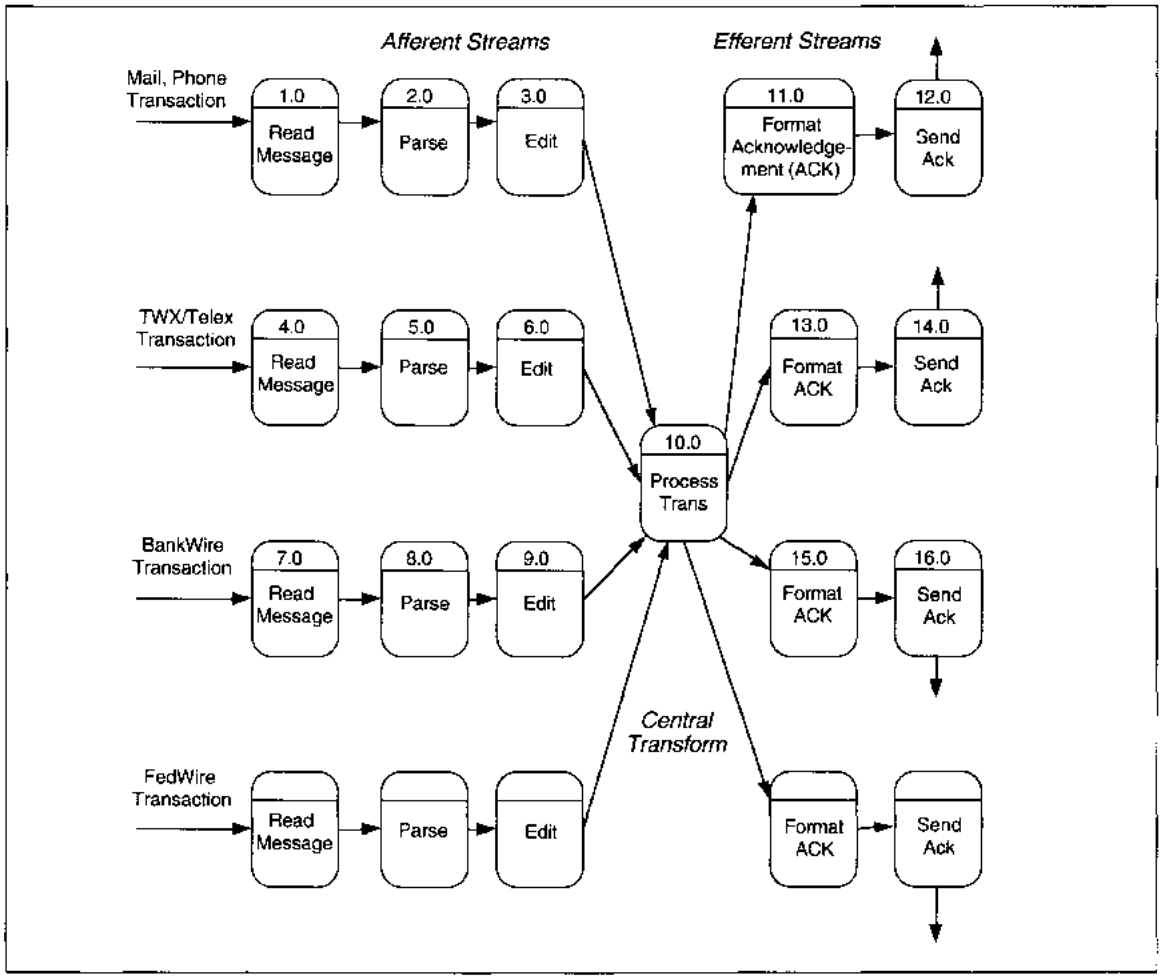


FIGURE 8-18 Detailed Money Transfer DFD Partitioned

To properly structure modules, their interrelationships and the nature of the application must be well understood. If a system concept has not yet been decided, design cannot be finalized until it is. The concept includes the timing of the application as batch, on-line or real-time for each process, and a definition of how the modules will work together in production. This activity may be concurrent with transform analysis, but should have been decided to structure and package processes for an efficient production environment. This activity is specific to the application and will be discussed again for ABC rental processing.

First, we identify the central transform and afferent and efferent flows. Look at the DFD and locate each stream of processing for each input. Trace each stream until you find the data flow that identifies valid, processable input that is the end of an afferent stream. The afferent and efferent arcs refer only to the processes in the diagram. During this part of the transform analysis, files and data flows are ignored except in determining afferent and efferent flows.

After identifying the afferent flows, trace backward from specific outputs (files or flows to entities) to identify the efferent flows. The net afferent and

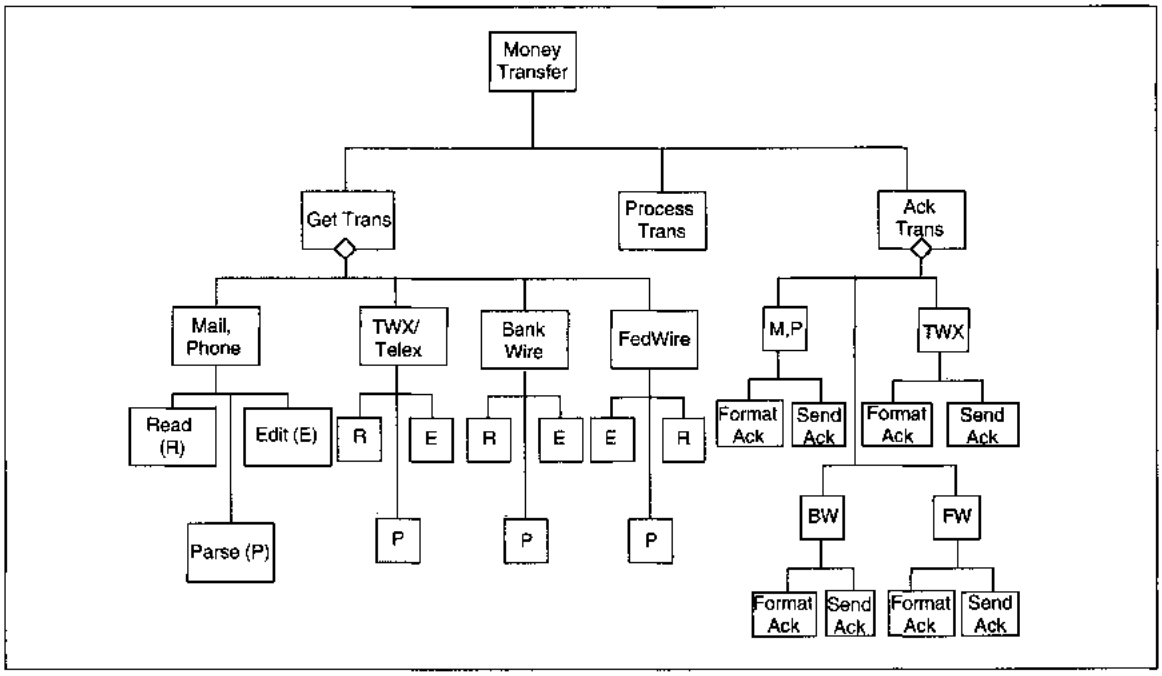


FIGURE 8-19 Sample Transaction Control Structure

TABLE 8-4 ABC Transaction List

Transaction	General Process	Data
Add Customer	Maintenance	Customer
Change Customer	Maintenance	Customer
Delete Customer	Maintenance	Customer
Query Customer	Periodic	Customer
Add Video	Maintenance	Video
Change Video	Maintenance	Video
Delete Video	Maintenance	Video
Query Video	Periodic	Video
Rent Video	Rent/Return	Video, Customer, History
Return Video	Rent/Return	Video, Customer, History
Assess special charges	Rent/Return	Customer
Query	Periodic	Video, Customer, History
Create History	Periodic	Video, Customer, History
Generate Reports	Periodic	Video, Customer, History

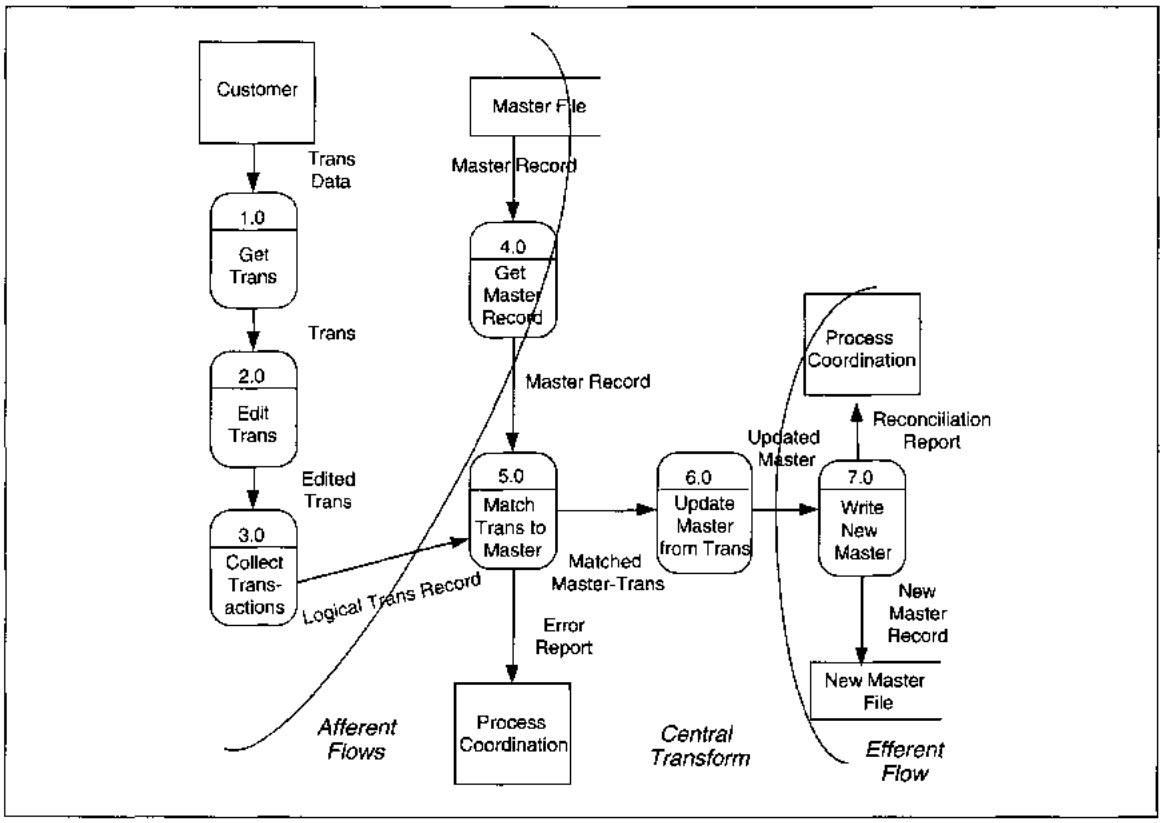


FIGURE 8-20 Master File Update DFD Partitioned

efferent outputs are used to determine the initial structure of the structure chart, using a process called factoring. **Factoring** is the act of placing each unbroken, single strand of processes into its own control structure, and of creating new control processes for split strands at the point of the split. The new control structure is placed under the input, process, or output controls as appropriate.

A master file update is shown as Figure 8-20 to trace the streams. In this diagram, we have two afferent data streams which come together at *Match Trans to Master*. The first input, *Trans Data* flows through process *Get Trans* and through *Edit Trans* to become *Edited Trans*. Successfully edited transaction parts flow through *Collect Transactions* to become *Logical Trans Record*.

The second input stream deals with the master file. The *Master Record* is input to *Get Master Record*; successfully read master records flow through the process. Once the *Logical Trans Record* and *Master Record* are both present, the input transformations are complete. These two afferent streams completely describe inputs, and the arc is drawn over the *Logical Trans Record* and *Master Record* data flows (see Figure 8-20).

The two streams of data are first processed together in *Match Trans to Master*. Information to be updated flows through *Update Master from Trans* to become *Updated Master*. The error report coming from the match process is considered a trivial output and does not change the essential transform nature of the process. The argument that *Match Trans*

to Master is part of the afferent stream might be made. While it could be treated as such, the input data is ready to be processed; that is, transactions by themselves, master records by themselves, and transactions with master records might all be processed. Here, we interpret the first transformation as matching.

The data flow out of *Update Master from Trans* is a net outflow, and *Write New Master* is an efferent process. The efferent arc is drawn over the data flow *Updated Master*.

Next, we factor three basic structures that relate to input-process-output processing (see Figure 8-21). If there is more than one process in a stream, getting the net output data may require some interprocess coordination. The coordination activities are grouped and identified by a name that identifies the net output data. So, in the example, the input stream is *Get Input*; the transform stream is *Process*; the output stream is *Write New Master*. Each stream represents the major elements of processing. Because the process and input streams both are compound, each has *at least* two streams beneath them—one for each sequential process stream to reach the net output data.

Notice that the DFD process names identify both data and transformation processes. Make sure that the lowest-level names on the structure chart are *identical* to the name on the data flow diagram to simplify completeness checking.

Notice also that there *is* transformation processing within the afferent and efferent streams. Modules frequently mix input/output and transform processing, and there is no absolute way to distinguish into

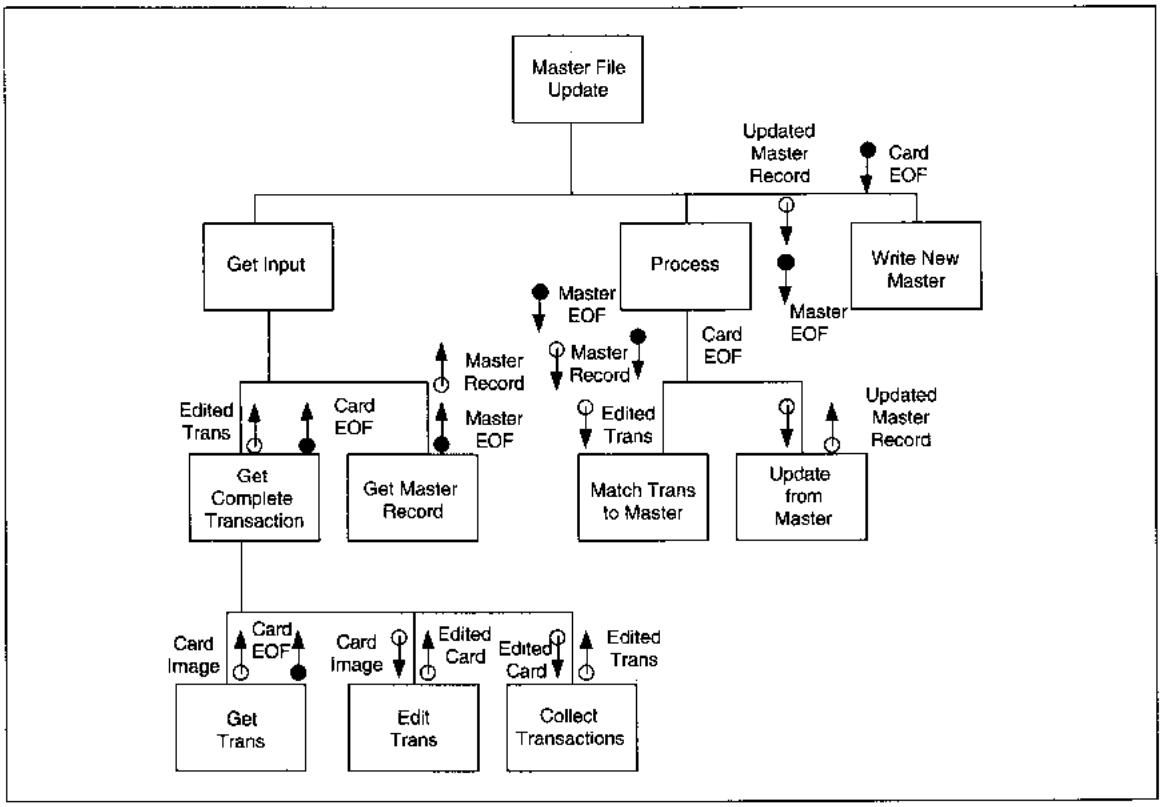


FIGURE 8-21 Master File Update Structure Diagram

which stream the module belongs. The rule of thumb is to place a module in the stream which *best describes* the majority of its processing.

Once the module is on the structure chart, we specifically evaluate it to ensure that it meets the principles of fan-out, span of control, maximal cohesion, and minimal coupling. If it violates even one principle, experiment with moving the module to the alternative streams and test if it better balances processing, without changing the processing. If so, leave it in the new location; otherwise note that the unbalanced part of the structure chart may need special design attention to avoid production bottlenecks.

Decompose the structure chart entries for each process. The three heuristics to guide the decomposition are:

- Is the decomposition also an IPO structure? If yes, continue; if no, do not decompose it.
- Does the control of the decomposed processing change? If yes, do not decompose it. If no, continue.
- Does the nature of the process change? That is, if the process is a date-validation, for instance, once it is decomposed is it still a date-validation? If no, continue. If yes, do not decompose it. In this example, I might try to decompose a date-validation into month-validate, day-validate, and year-validate. I would need to add a date-validate to check all three pieces together. Instead of a plain date-validate, I have (a) changed the nature of the process, and (b) added control logic that was not necessary.

The thought process in analyzing depth is similar to that used in analyzing the number of organizational levels in reengineering. We want only those levels that are required to control hierarchic complexity. Any extra levels of hierarchy should be omitted. Now let us turn to ABC rental processing to do transform analysis and develop the structure chart.

ABC Video Example Transform Analysis

The decisions about factoring are based on the principles of coupling and cohesion, but they also require a detailed understanding of the problem and a design approach that solves the whole problem. In ABC Video's case, we have to decide what the relationships of rent, return, history, and maintenance processing are to each other. If you have not done this yet, now is the time to do it. Before we continue with design of transform analysis, then, we first discuss the design approach and rationale.

DESIGN APPROACH AND RATIONALE. In Chapter 7, Table 7-5 identified the Structured English pseudo-code for ABC's rental processing and we did not discuss it in detail. Now, we want to examine it carefully to determine an efficient, cohesive, and minimally coupled decomposition of the process. When we partition the ABC Level 0 DFD from Figure 7-26, customer and video maintenance are afferent streams, reports are efferent, and rental and return are the central transforms (see Figure 8-22). We will attend only to create and return rentals since they are the essence and hardest portion of the application.

There is a design decision to have return processing as a subprocess of rental processing that needs some discussion. Then we will continue with the design. The overall design could be to separate rentals and returns as two different processes, but are they? Think in the context of the video store about how the interactions with customers takes place. Customers return tapes previously taken out. Then they select tapes for rental and pay all outstanding fees, including current and past returns that generate late fees. To have late fees, a tape must have been returned.[2] Rentals and returns are separated in time; they have separate actions taken on files. ABC has any combination of rentals with returns (with or without late fees) and open rentals. All open rentals are viewed during rental processing, but need not be during return processing. Adding a return date and late fees is a trivial addition. Returns could be

2 In a real video rental system, you would also have a delinquent or exceptional charges process to add fees for lost and damaged tapes. We do not consider that complexity here as it does not materially add to the discussion.

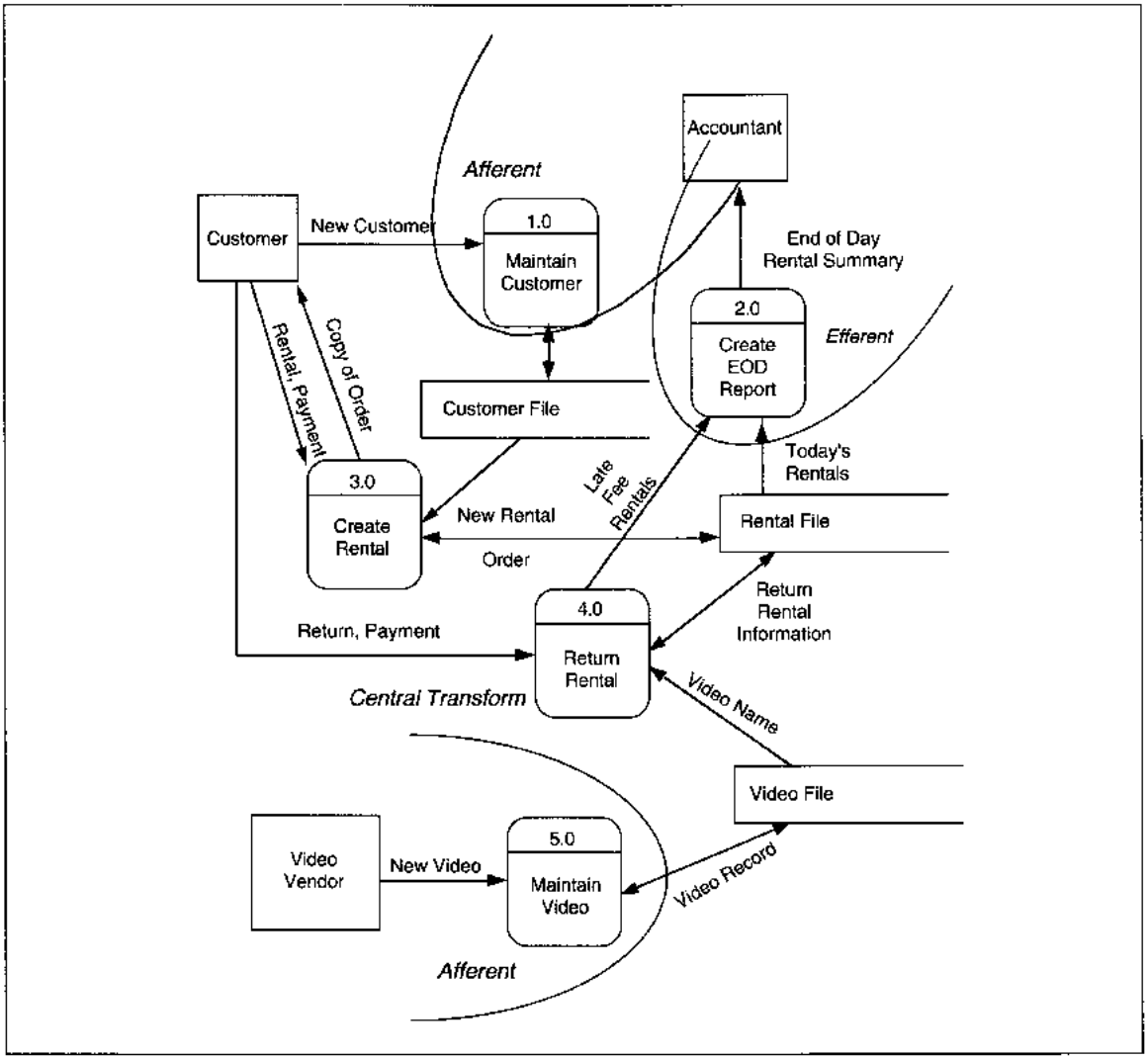


FIGURE 8-22 ABC Video Level 0 DFD Partitioned (same as Figure 7-26)

independent of rentals, so there are three design alternatives:

- Returns are separated from rentals.
- Rentals are a subset of returns.
- Returns are a subset of rentals.

If returns are separated from rentals, there would be two payment processes—one for the return and one for the rental. If a rental includes a return, this is not 'minimal bureaucracy' and is not desirable. However, since returns can be done independently from rentals, the system should not require rental processing to do a return. This alternative is an acceptable *partial* solution, but the rest of the solution must be included.

The second alternative is to treat rentals as part of the return process. This reasoning recognizes that a rental precedes a return. All returns would need a rental/no rental indicator entry and assume that more than 50% of the time, rentals accompany returns.

Which happens more frequently—returns with rentals, or rentals without returns? Let's say Vic does not know and reason through the process. Since returns can be any of three ways, only one of which is with rentals, coupling them as rental-within-return *should* be less efficient than either of the other two choices.

Last, we can treat returns as part of the rental process. If returns are within rentals, we have some different issues. What information identifies the beginning of a rental? What identifies the beginning of a return? A customer number could be used to signify rental processing and a video number could signify a return. If we do this, we need to make sure the numbering schemes are distinct and nonoverlapping. We could have menu selection for both rental and return that determines the start of processing; then return processing also could be called a subprocess of rentals. Either of these choices would work if we choose this option. For both alternatives, the software needs to be reevaluated to maximize reusable modules because many actions on rentals are also taken on returns, including reading and display of open rentals and customer information.

Having identified the alternatives and issues, we conduct observations and collect data to justify a selection. The results show that 90% of returns, or about 180 tapes per day, are on time. Of those, 50% are returned through the drop box, and 50% (90 tapes) are returned in person with new rentals. The remaining 10% of returns also have about 50% (10 tapes) accompanying new rentals. So, about 100 tapes a day, or 50% of rentals are the return-then-rent type. These numbers justify having returns as a subprocess of rentals. They also justify having returns as a stand-alone process. We will allow both.

Deciding to support both separate and return-within-rental processing means that we must consciously decide on reusable modules for the activities the two functions both perform: reading and display of open rentals and customer information, payment processing, and writing of processing results to the open rental files. We will try to design with at least these functions as reusable modules.

DEVELOP AND DECOMPOSE THE STRUCTURE CHART. To begin transform analysis, we start with the last DFD created in the analysis phase, and the data dictionary entries that define the DFD details. Figure 7-28 is reproduced here as Figure 8-23, with a first-cut partitioning to identify the central transform.

First, we evaluate each process. We will use the pseudo-code that is in the data dictionary (see Figure 8-24). The DFD shows three rental subprocesses: *Get Valid Rental*, *Process Fees and Money*, and *Create and Print Rental*. Each of the subprocesses might be further divided into logical components. Try to split a routine into a subroutine for each function or data change. First, evaluate the potential split to make sure the subroutines are all still needed to do the routine. This double-checks that the original thinking was correct. Then, evaluate each potential split asking if adding the subroutine changes the control, nature, or processing of the routine. If yes, do not separate the routine from the rest of the logic; if no, abstract out the subroutine.

For ABC, *Get Valid Rental* is the most complex of the routines and is evaluated in detail. *Get Valid Rental* has three subroutines that we evaluate: *Get Valid Customer*, *Get Open Rentals*, and *Get Valid Video*. These splits are based on the different files that are read to obtain data for processing a rental. Without all three of these actions, we do not have a valid rental, so the original designation of *Get Valid Rental* appears correct. Figure 8-25 shows refined pseudo-code for ABC rental processing with clearer structure and only structured constructs. Subroutines are shown with their own headings.

If we are to accommodate returns during rental processing, we have to decide where and how rentals fit into the pseudo-code. We want to allow return dates to be added to open rentals. We also want to allow returns *before* rentals and returns *within* rentals. This implies that there are two places in the process where a rental *Video ID* might be entered: before or after the *Customer ID*. If the *Video ID* is entered first, the application would initiate in the *Return* process; from there, we need to allow additional rentals. If the *Customer ID* is entered first, the application would initiate rental; from there, we need to allow returns. To allow both of these actions to lead to rental and/or return processing, we need to add some control structure to the pseudo-code (see

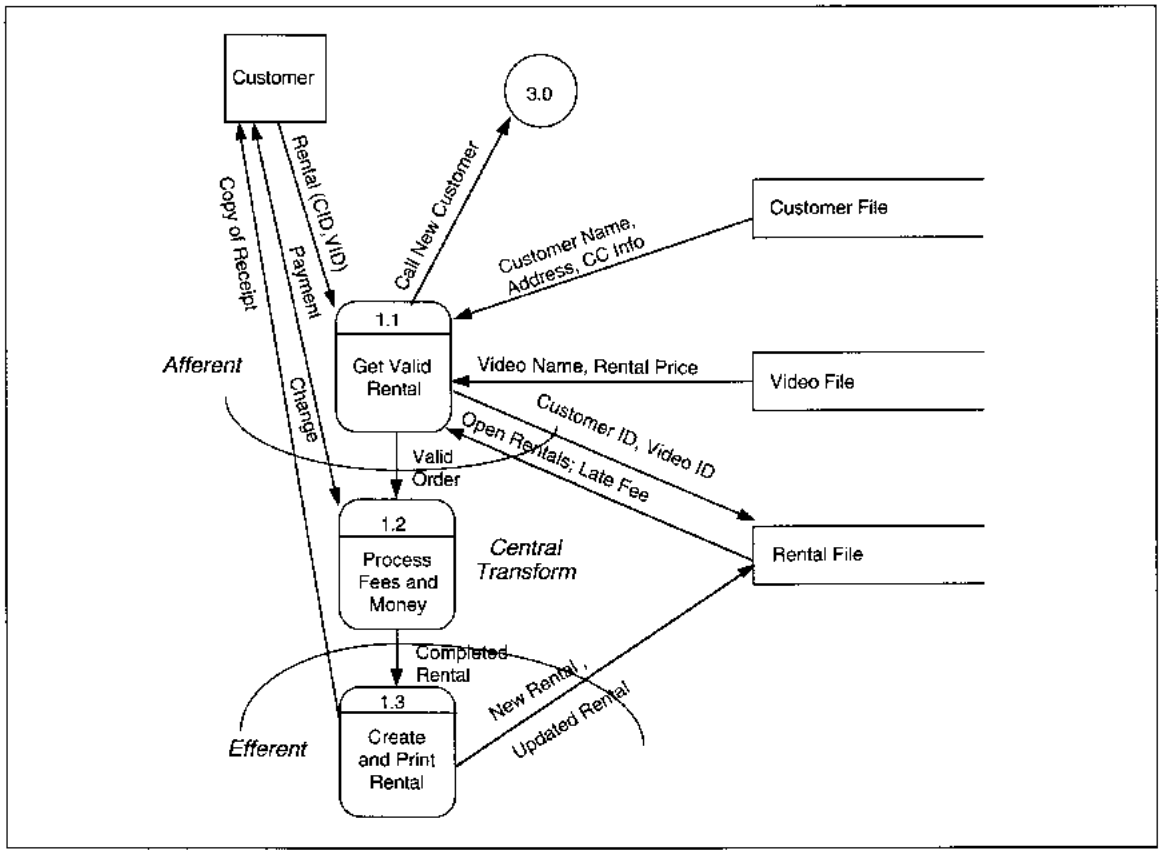


FIGURE 8-23 ABC Video Level 1 DFD Partitioned (same as Figure 7-28)

Figure 8-26). The control structure also changes the resulting structure chart somewhat even though the DFDs are not changed.

Next, we evaluate the refined pseudo-code and inspect each subroutine individually to determine if further decomposition is feasible (see Figure 8-27). For *Get Valid Customer*, does the processing stay the same? That is, are the detail lines of procedure information the same? By adding the subroutine we want to add a level of abstraction but not new logic. In this case, the answer is yes. Now look at the details of *Get Valid Customer*. The subprocesses are *Get Customer Number*—a screen input process, *Read and Test Customer File*—a disk input process with logic to test read success and determine credit worthiness, and *Display Customer Info*—a screen output process. Again, we have decomposed *Get Valid Customer* without changing the logic or adding any new functions.

The results of the other evaluations are presented. Walk-through the same procedure and see if you develop the same subroutines. Here we used the pseudo-code to decompose, but we could have used text or only our knowledge of processing to describe this thinking. When the decomposition is complete for a particular process stream, it is translated to a structure chart.

Complete the Structure Chart

Rules for Completing the Structure Chart

Completion of the structure chart includes adding data and control couples and evaluating the diagram.

```
Get Valid Rental.
  For all customer
    Get customer #
    Read Customer File
    If not present,
      Cancel
    else
      Create customer
      Display Customer info.

  Read Open-Rentals
  For all Open Rentals,
    Compute late fees
    Add price to total price
    Display open rentals
    Display total price.

  For all video
    Read Video file
    If not present
      Cancel this video
    else
      Create Video
      Display Video
      Add price to total price
      Display total price.

Process Fees and Money.
  Get amount paid.
  Subtract total from about paid giving change.
  Display change.
  If change = zero and total = zero,
    mark all items paid
  else
    go to process fees and money.

Create and Print Rental.
  For all open rentals
    if item paid
      rewrite open rental.
  For all new rentals
    write new open rental.
  Print screen as rental confirmation.
```

FIGURE 8-24 ABC Rental Pseudo-code

```
Get Valid Rental.
  Get Valid Customer.
    For all customer
      Get customer #
      Read Customer File
      If not present,
        Cancel
      else
        Create customer
        Display Customer info.

  Get Open Rentals.
    Read Open-Rentals
    For all Open Rentals,
      Compute late fees
      Add price to total price
      Display open rentals
      Display total price.

  Get Valid Video.
    For all video
      Read Video file
      If not present
        Cancel this video
      else
        Call Create Video
        Display Video
        Add price to total price
        Display total price, change.

Process Fees and Money.
  Get amount paid.
  Subtract total price from about paid giving
  change.
  Display total price, change.
  If change = zero and total = zero,
    mark all items paid
  else
    go to process fees and money.

Create and Print Rental.
  Update Open Rentals.
    For all open rentals
      if item paid
        rewrite open rental.

  Create New Rentals.
    For all new rentals
      write new open rental.
    Print screen as rental confirmation.
```

FIGURE 8-25 ABC Rental Pseudo-code Refined

```
Get Valid Rental.
  Get entry.
  If entry is Video
    Call Return
  else
    Call Rental.

  Rental.
    Get Valid Customer.
      For all customer
        Get customer #
        Read Customer File
        If not present,
          Cancel
        else
          Create customer
          Display Customer info.

    Get Open Rentals.
      Read Open-Rentals
      For all Open Rentals,
        Compute late fees
        Add late fees to total price
        Display open rentals
        Display total price.

    Get Valid Video.
      For all video
        Read Video file
        If not present
          Cancel this video
        else
          Call Create Video
          Display Video
          Add price to total price
          Display total price, change.

    Process Fees and Payment.
    Create and Print Receipt.

  Return.
    Get Open Rental.
      Read Open-Rentals
      Read Customer
      Display Customer
      Display Open Rental
      Add return date.
    Using customer ID, Read Open Rentals.
      For all Open Rentals
        Display open rentals.
        For all return request
          Add return date to rental.
          Compute late fees
          Add late fees to total price
          Display total price.
    If rental
      Call Get Valid Video.
    Call Process Fees and Payment.
    Call Create and Print Receipt.
```

FIGURE 8-26 Get Valid Rental Pseudo-code with Control Structure for Returns

Structure chart completion rules are:

1. For each data flow on the DFD add exactly one data couple. Use exactly the same data flow name for the data couple.
2. For each control module, decide how it will control its subs. If you need to refine the pseudo-code to decide control, do this. Add control couples to the diagram when they are required between modules.
3. For modules that select one of several paths for processing, show the selection logic with a diamond in the module with the logic attached to the task transfer line.

Rules of thumb for developing the structure chart are:

1. Evaluate the diagram for cohesion. Does each module do one thing and do it completely?
2. Evaluate the diagram for fan-out, fan-in, skew, and redesign as required, adding new levels of control. Note skewed processing for attention during program design.
3. Evaluate the diagram for minimal coupling. Is the same data used by many modules? Do control modules pass only data needed for processing? Do control modules minimize their scope of effect?

These are all discussed in this section.

First, the structure chart is drawn based on the decomposition exercises. Then data couples are added to the diagram for each data flow on the DFD. If the

```
Get Valid Rental.
  Get entry.
  If entry is Video
    Call Return
  else
    Call Rental.

  Rental.
    Call Get Valid Customer.
    Call Get Open Rentals.
    Call Get Valid Video.
  Return.
    Call Get First Return.
    Call Get Open Rentals.
    If rental
      Call Get Valid Video.

Process Fees and Money.

Create and Print Rental.
  Update Open Rentals.
  Create New Rentals.
  Print receipt.
-------------------------------------------
Get Valid Customer.
    Get customer #
    Read Customer File
    If not present,
      Create Customer.
    If CCredit not zero, display CCredit
    Display Customer info.
-------------------------------------------
Get Open Rentals.
  Read Open-Rentals
  For all Open Rentals,
    Compute late fees
    Add late fees to total price
    Display open rentals
    Display total price, change.
  For all return request
   Call Update Returns.
-------------------------------------------
Get Valid Video.
  For all video
    Read Video file
    If not present
      Cancel this video
    else
      Call Create Video
      Display Video
      Add price to total price
      Display total price, change.
-------------------------------------------
Get First Return.
  Read Open-Rentals
  Read Customer
  Display Customer
  Display Open Rental
  Call Update Returns.
-------------------------------------------
Update Returns.
  Move return date to rental.
  Update video history.
  Compute late fees.
  Add late fees to total price.
  Display total price.
-------------------------------------------
Process Fees and Money.
  Get amount paid.
  Subtract total price from about paid giving change.
  Display total price, change.
  If change = zero and total = zero,
    mark all items paid
  else
    go to process fees and money.
-------------------------------------------
Update Open Rentals.
  For all open rentals
    rewrite open rental.
-------------------------------------------
Create New Rentals.
  For all new rentals
    write new open rental.
```

FIGURE 8-27 Complete Pseudo-code for Rentals and Returns

structure chart is at a lower level of detail, use the data flow as a starting point and define the specific data passed to and from each module. Show *all* data requirements for each module completely. Make sure that all names are *exactly* as they are in the dictionary.

Next, for each control module, decide *how* it will control its subprocesses and add the control couples to the diagram. Decide whether the logic will be in the control module or in the subprocess. If the logic is in the control module, the goal is for the controller to simply call the subordinate module, pass data to

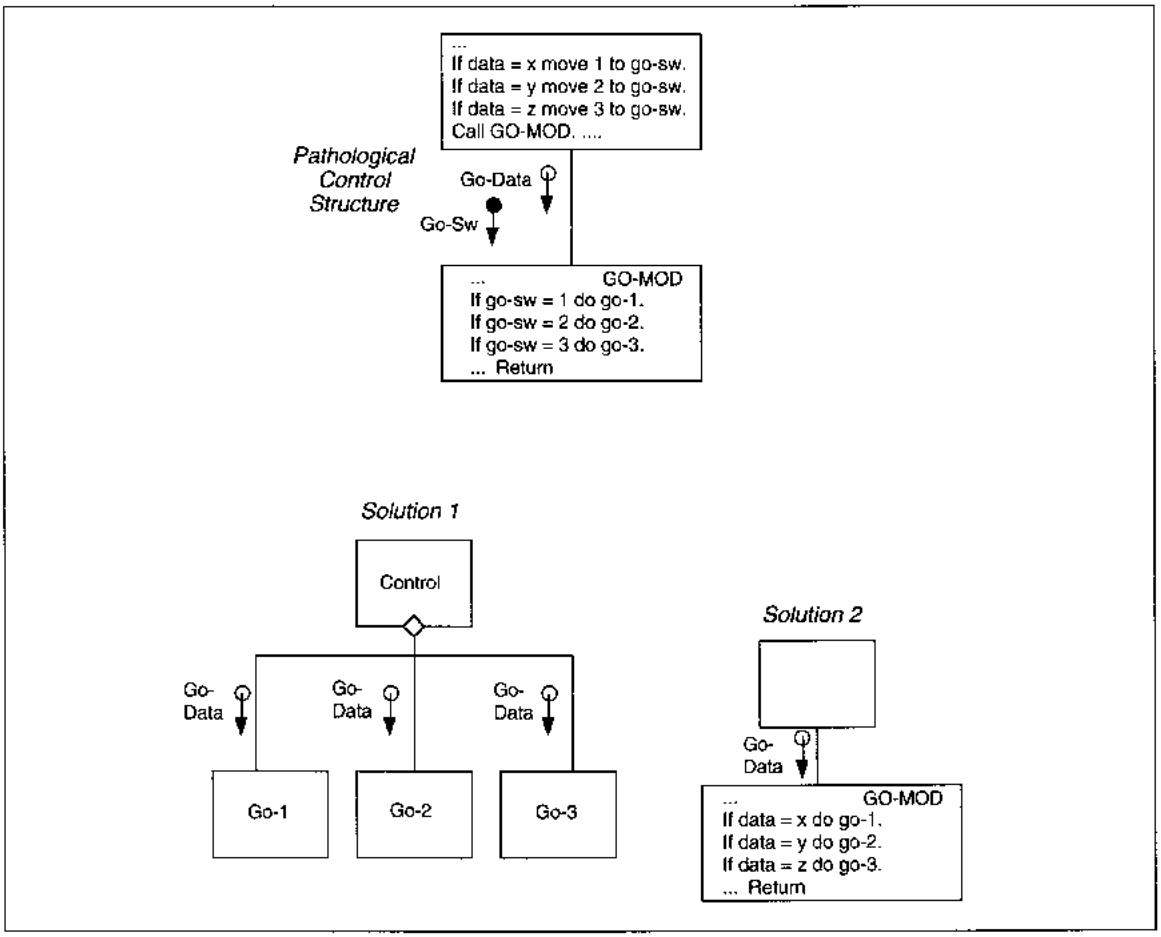


FIGURE 8-28 Pathological Control Structure and Two Solutions

transform, and receive the transform's data back. If any other processing takes place, rethink the control process because it is not minimally coupled.

A control couple might be sent to the subprocess for it to determine what to do. This may or may not be okay. Where is the control couple 'set' and 'reset'? If in the control module, this is acceptable. If somewhere else, rethink the control process and simplify it. Any time you *must* send a control couple for a module to decide which action to take, you identify a potential problem. The lower-level module may be doing too many things; otherwise it would not need to decide what to do, or the control may be in the wrong module.

An example of this problem and two solutions are illustrated in Figure 8-28. If the lower level is doing too many things, then decompose them to create several single-purpose modules. If the lower level is not doing multiple functions, then move control for the module into the module itself. In both cases, the goal of minimal coupling is attained.

Next, the diagram is evaluated for cohesion, coupling, hierarchy width, hierarchy depth, fan-out, fan-in, span of control, and skew. Evaluate the diagram for cohesion (see Table 8-2 for definition of cohesion types). Check that each module does one thing and does it completely. If several modules must be taken together to perform a whole function, the structure is

excessively decomposed. Regroup the processes and restructure the diagram.

Evaluate the diagram for width, depth, fan-out, fan-in, and skew. These are visual checks to see if some portion of the structure is inconsistent with the rest of the structure. The inconsistency does not necessarily mean that the diagram is wrong, only that there may be production bottlenecks relating to the out-of-balance processes. For a wide structure, double check that the subprocesses really all relate to one and only one process. If not, add a new control module, else leave as is.

For deep structures, check to see if each level of depth is performing some function beyond control. Ask yourself why all the levels are needed. If there is no good reason, get rid of the level and move its functions either up or down in the hierarchy, preferably up. Ask yourself if fewer levels can accomplish the same process. If the answer suggests reducing the levels of hierarchy, restructure the diagram and keep only essential levels.

For fan-in modules, check that each using module has the same type of data being passed and expects the same type of results from the fan-in module. If there are *any* differences, then either make the using modules consistent, or add a new module to replace the fan-in module for the inconsistent user module.

Skewed diagrams identify a fundamental imbalance of the application that may have been hidden before: that it is input-bound, output-bound, I/O-bound, or process-bound. Skew is not necessarily a problem that results in restructuring a diagram. When skewed processing is identified, you should verify that it is not an artifact of your factoring. If it is, remove the skew from the diagram by restructuring the modules.

Skew is not always a problem. When a skewed application is being designed, the designers normally spend more time designing the code for the bound portion of the problem to ensure that it does not cause process inefficiencies. For instance, Fortran is notoriously inefficient at physical input/output (i.e., reading and writing files). For anything but a process-bound application, Fortran is not the best language used. For a process-bound Fortran application, with many I/Os, another language, such as assembler or Cobol, might be used to make read/write processing efficient. The opposite is true of Cobol. Cobol is not good at high precision, scientific, mathematical processing. In a Cobol application, process-bound modules and their data would be designed either for another language, or to minimize the language effects.

Finally, evaluate the diagram for minimal coupling. First look at data couples. If you see the same data all over the diagram, there may be a problem. Either you are not specifying the data at the element level, or data coupling is the least coupling you will be able to attain. Make sure that only needed data is identified for passing to modules. Data coupling is not the best coupling, but it is tolerable.

Next look at control couples one last time. Make sure that they are set and reset in the same or directly-related modules, and make sure that, if passed, they are passed for a reason. If either of these conditions are violated, change the coupling.

To summarize so far, decide the system concept; partition the DFD; develop a first-cut structure chart; decompose the structure chart using pseudo-code of the functions as needed to guide the process; add data couples; add control couples; evaluate and revise as needed.

ABC Video Example Structure Chart

ABC's structure chart will begin with the Level 1 DFD factoring and progress to provide the detail for modules as expressed in the pseudo-code. There are three first level modules: *Get Valid Rental*, *Process Fees & Money*, and *Create and Print Rental* (see Figure 8-29). To get the next level of detail, we use the pseudo-code or decomposed structure charts. In our case, we use the pseudo-code. In Figure 8-27, the high level pseudo-code has only module names. We simply transfer those names to modules on the structure chart, attending to the control logic present in the diagram.

For each *if* statement, we need to decide whether that statement will result in a direct call (our choice, here) or whether it will result in a control couple being passed. Direct calls are preferred to minimize coupling. When a direct call is used, the module is executed in its entirety every time it is called.

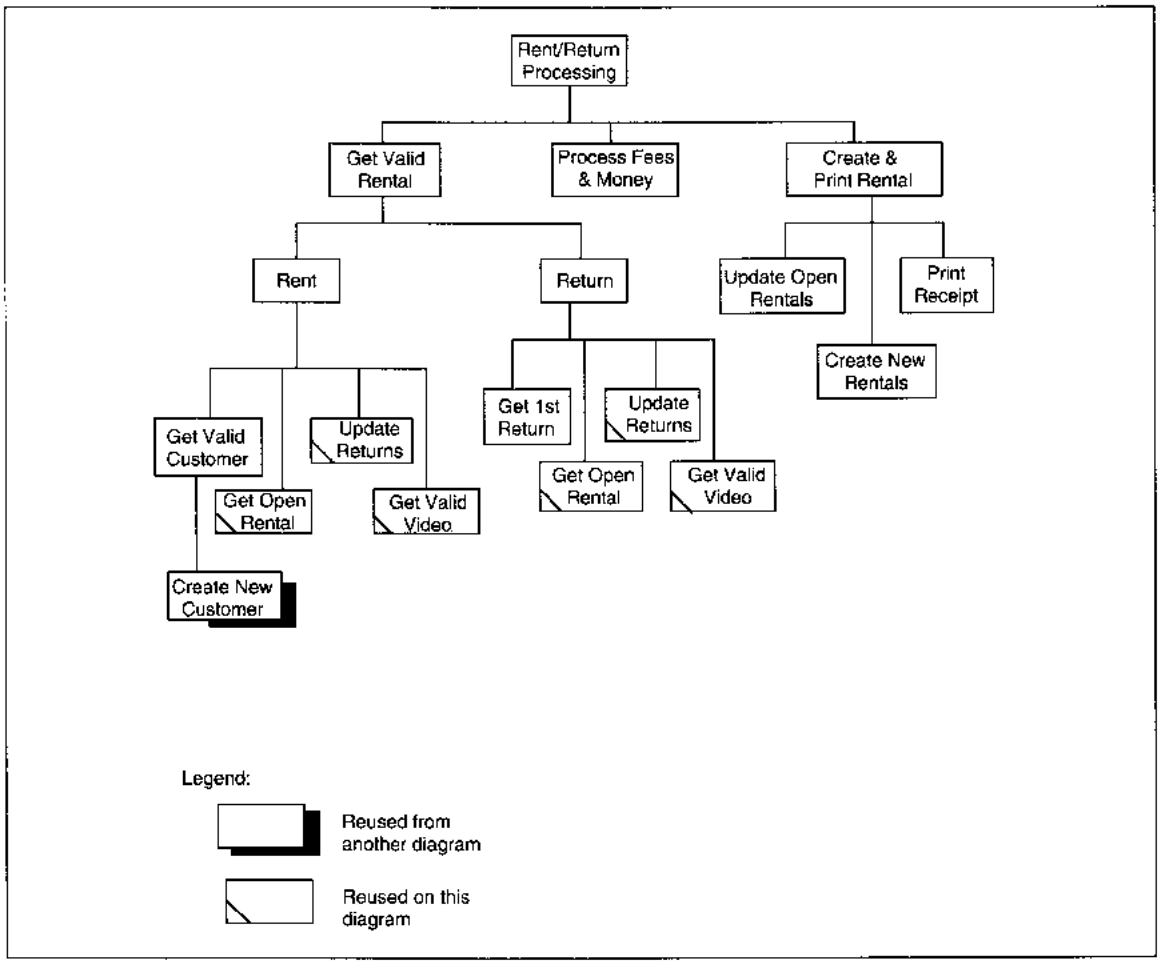


FIGURE 8-29 Rent/Return First-Cut Structure Chart

We identify reused modules by a slash in the lower left corner of the rectangles to show the complete morphology of the diagram. The first-cut structure chart shows that the processing is skewed toward input. Because there are three data stores affected by every process, there is no way to get rid of the skew without getting rid of the control level. Is the control level essential? If we omit the control level is the processing the same? Do we violate fan-out if we remove the control level? The answers are no, mostly, and no, respectively. If we remove the control level, its logic must go somewhere. The logic can move up a module and not violate fan-out. The change may have a language impact, so we will not change it until we decide program packages.

We note it for attention during packaging and programming. There are no other obvious problems with the first-cut structure chart. Since we have developed it bottom-up, using the pseudo-code as the basis, it is as good as our pseudo-code.

Next, we add the data and control couples needed to manage processing. The final diagram is shown in Figure 8-30, which we evaluate next.

Each module appears to do only one thing. The diagram is input-skewed as already discussed. The span of control and fan-out seem reasonable.

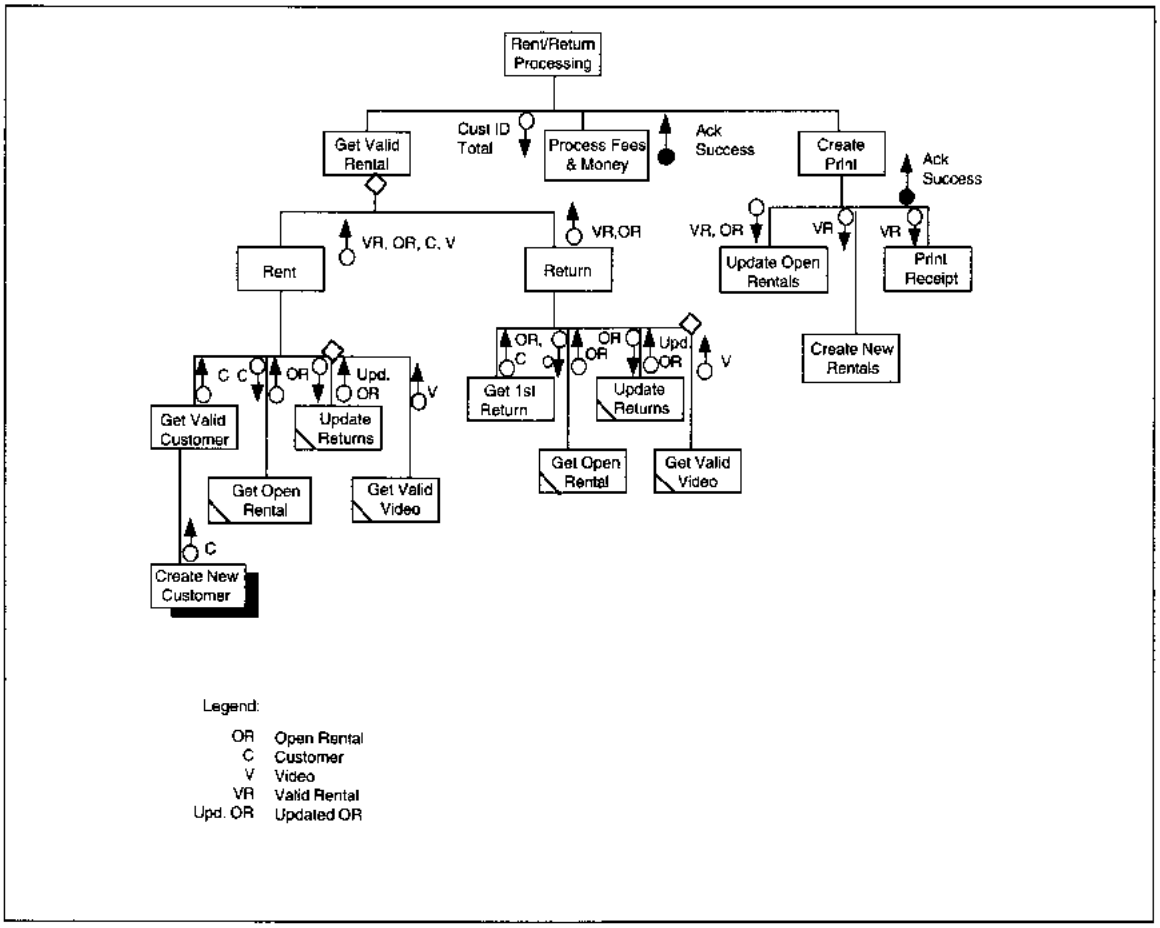


FIGURE 8-30 Completed Rent/Return Structure Chart

The reused modules each have the same input data. The hierarchy is not unnecessarily deep, although the control code for *Get Valid Rental, Rent*, and *Return* might be able to be combined depending on the language. Coupling is at the data level and is acceptable. Next, we turn to designing the physical database.

Design the Physical Database

Physical database design takes place concurrently with factoring and decomposition. A person with special skills, usually a database administrator (DBA), actually does physical database design. In companies without job specialization, a project team member acts as the DBA to design the physical database. Physical database design is a nontrivial task that may take several weeks or even months.

Rules for Designing the Physical Database

The general physical database design activities are summarized below. Keep in mind that many other activities may be involved in designing a physical database that relate to a specific implementation environment.

1. Define user views based on transaction types and data accessed for each transaction.

2. Identify access method if choices exist.
3. Map user views to access method and storage technology to optimize disk space and to minimize access time.
4. Build prototype and test, revising as indicated.
5. Develop database for application testing.
6. Document physical database design and distribute user view information to all project team members.
7. Work with conversion team to build production databases.

Designing user views means to analyze the transactions or inputs of each process to define which database items are required. In general, the data items processed together should be stored together. These logical design activities constrain the physical design and help the person mapping to hardware and software.

In selecting the access method, the physical data designer seeks to optimize matching available access methods to access requirements. Access method choices usually are data sequenced (i.e., indexed), entry sequenced (i.e., direct), inverted lists, or some type of b-tree processing. Each DBMS and operating system has its own access method(s) from which selection is made. The details of these access methods are beyond the scope of this text.[3]

User views are mapped to the access method and a specific media. Media mapping seeks to optimize access time for individual items and sets of items. It also seeks to minimize wasted space while providing for growth of the database. Since media have become one of the major expenses in the computing environment, there may be political issues involved with physical database design. At this point, a database walk-through reviews all database design before a prototype is built.

The DBA documents and trains team members in data access requirements. The DBA, working from the application specification, maps data requirements to user views to processes. Each process, then, has specific data items assigned. Every team member must know exactly what data items to access and how to access them. If a module or program accesses the wrong data item, an inconsistent database might result. Also, minimal data coupling requires that each process access only data that it requires. Incorrect use of access methods can lead to process bottlenecks or an inconsistent database. To assure that programs are using the data correctly, the DBA may participate in walk-throughs to monitor data access.

The DBA works with the test team to load the data needed for testing. The DBA also works with the conversion team to load the initial production database. These activities may be trivial or may require hiring of temporary clerks to input information to the database. The DBA and the two teams work together to verify the correctness of the data, to provide program test database access to the rest of the development team, and to provide easily accessed backup when the test database is compromised. After the test database is loaded, the backup and recovery procedures, transaction logic procedures, and other database integrity procedures are all finalized and tested.

To summarize, a person who intimately knows the technical production data environment acts as a DBA, mapping the database to a physical environment and building both test and production databases. The DBA provides training and guidance to the other team members for data access, and participates in data related walk-throughs.

ABC Video Example Physical Database Design

In order to do the physical database design, a DBMS must be selected. We will design as if some SQL engine were being used. SQL's physical design is closely tied to the logical design so the design activity becomes less DBMS software sensitive. In addition, SQL data definition is the same in both mainframe and micro environments so the design activity does not need to be hardware platform sensitive. The amount of storage space (i.e., number of tracks or cylinders) will vary, of course, since disks

3 For more on access methods and storage considerations, see references to Fabbri and Schwab [1992], Codd [1990], Bohl [1981], and Claybrook [1983] in the references.

on PCs do not yet hold as much information as mainframe disks.

Beginning with the logical design from Table 7-7, we define the relations and data items that are required to develop user views. Remember from database class, that the logical database design can map directly to the physical database. The relations defining the actual database may or may not be accessed by users. For security reasons, user views may be used to control access to data and only the DBA would even know the real relation names.

To define user views, we examine each process and identify the data requirements. List the requirements by process (see Table 8-5). Match similar data requirements across processes to identify shared user views. The problem is to balance the number of views against the number of processes. Ideally a handful of user views are defined; a heuristic for large applications is about 20 user views. Beyond that, more DBAs are required and database maintenance becomes difficult. In a large application, keeping the number of user views manageable may be difficult and require several design and walk-through iterations.

For ABC rental processing, we need a user view for each major data store: *Customer*, *Video Inventory*, and *Open Rentals*. We also need user views for the minor files: *Video History*, *Customer History*, and *End Of Day Totals*. If data coupling and memory usage are not an issue, using a SQL database, we can create one user view for each of *Customer*, *Video*, and *Open Rental*, and create one joined user view using the common fields to link them together. The individual views are used for processes that do not need all of the data together; the joined view can be used for query processing and for processes that need all of the data. The resulting data definitions for customer, video, open rentals, and the related user views are shown in Table 8-6. We also need separate user views for the history files and EOD totals. They are included in the table.

At this point, with SQL software, we are ready to prototype the database. If either access method selection or storage mapping is an issue, a prototype should be built. Otherwise, the next step is to map user views to access methods and storage media. This activity depends on the implementation environment and is beyond this text. The database may be walked through again at this point to verify processing requirements for the database. The database is then prototyped and documented. The information needed for each program is included in program specifications. Team members are usually given an overview of the database environment either as part of the last walk-through or as a separate training session. When the prototype appears complete and workable, test and production databases are developed.

Design Program Packages

Rules for Designing Program Packages

The activities for grouping modules into program packages are listed below; as you can see, they are general guidelines, not rules. There are no rules for packaging because it is an environment-dependent activity. Packages for firmware or an 8K micro computer are entirely different than packages for a mainframe. Also, the implementation language determines how and when some types of coupling are done. With these ideas in mind, the guidelines apply common sense to identifying program execute units.

1. Identify modules that perform functionally related activities, are part of iteration units, or which access the same data. The related modules identified should be considered for packaging together for execution.
2. Develop pseudo-code for the logic functions being performed. Use only structured programming constructs: iteration, selection, and sequence. Document complex logic using decision tables or decision trees.
3. Logically test the user views developed with the DBA to reevaluate their usefulness for each program package.
4. Design each module to have one entry and one exit.
5. Design each module such that its contents are unchanged from one execution to the next.
6. Design and document messages for called modules. Reevaluate the messages to minimize coupling.
7. Draw a diagram of the module and all other modules with which it interacts.

TABLE 8-5 ABC Data Requirements by Process

Process	Customer	Video Inventory	Open Rental	Other
Get Valid Customer	Customer Phone, Name, Address, Credit Rating			
Get Open Rentals			Customer Phone, Video ID, Copy ID, Video Name, Rent Date, Return Date, Late Days, Fees Owed	
Get Valid Video		Video ID, Copy ID, Video Name, Rental Price		
Get First Return			Customer Phone, Video ID, Copy ID, Video Name, Rent Date, Return Date, Late Days, Fees Owed	
Get Valid Video		Video ID, Copy ID, Video Name, Rental Price		
Update Rentals			Customer Phone, Video ID, Copy ID, Video Name, Rent Date, Return Date, Late Days, Fees Owed	
Process Fees and Money				End of Day Totals Total Price + Rental Information
Create Video history				Video History File: Year, month, Video ID, Copy ID
Create Customer history				Customer History File: Customer Phone, Video ID
Update Open Rentals			Customer Phone, Video ID, Copy ID, Video Name, Rent Date, Return Date Late Days, Fees Owed	
Create New Rentals			Customer Phone, Video ID, Copy ID, Video Name Rent Date, Return Date, Late Days, Fees Owed	
Print receipt				Customer Phone, Name, Address, For each Video: Video ID, Copy ID, Video Name, Rent Date, Return Date, Late Days, Fees Owed, Total Price

TABLE 8-6 SQL Data Definitions and User Views

```
Create Table Customer
   (Cphone        Char(10)          Not null,
   Clast          VarChar(50)       Not null,
   Cfirst         VarChar(25)       Not null,
   Cline1         VarChar(50)       Not null,
   Cline2         VarChar(50)       Not null,
   City           VarChar(530)      Not null,
   State          Char(2)           Not null,
   Zip            Char(10)          Not null,
   CCtype         Char(1)           Not null,
   Ccno           Char(17)          Not null,
   Ccexp          Date              Not null,
   CCredit        Char(1),
   Primary key    (Cphone));

Create Table Video
   (VideoID       Char(7)           Not null,
   VideoNam       Varchar(50)       Not null,
   VendorNo       Char(4)
   TotCopies      Smallint          Not null,
   RentPrice      Decimal(1,2)      Not null,
   Primary key    (videoID);

Create Table Copy
   (VideoID       Char(7)           Not null,
   CopyID         (Char(2)          Not null,
   DateRecd       Date
   Primary key    (VideoID, CopyID),
   Foreign Key    ((VideoID) References Video);

Create Table Rental
   Cphone         Char(10)          Not null,
   RentDate       Date              Not null,
   VideoID        Char(7)           Not null,
   CopyID         (Char(2)          Not null,
   RentPaid       Decimal(2,2)      Not null,
   FeesOwed       Decimal(2,2)
   Primary Key    (CPhone, VideoID, CopyID),
   Foreign Key    ((VideoID) References Video)
   Foreign Key    ((VideoID, CopyId) References
                    Copy),
   Foreign Key    (CPhone) References Customer);

Create view VidCrsRef
   as select VideoID, CopyID, VideoName, RentPric
     from Customer, Video, Copy
     where Video.VideoID = Copy.VideoID;

Create view RentRef
     as select Cphone, Clast, Cfirst, VideoID, CopyID,
       VideoNam,
     RentPaid, RentPric, FeesOwed
     from Customer, VidCrsRef, Rental
     where VidCrsRef.VideoID = Rental.VideoID
     and VidCrsRef.CopyID = Rental.CopyID
     and Customer.Cphone = Rental.Cphone;
```

A program package is a collection of called modules, called functions, and in-line code that does some atomic process, and that will become an execute unit. The hierarchy of criteria for designing packages is to package by function, by iteration clusters, or by need to access the same data. At all times, you must keep in mind any production environment constraints that must also be part of the design. For instance, if the application will be on a LAN, you may want to design packages to minimize the possibility of multiple users for a process.

Functional grouping is, by far, the most important. Functional grouping ensures high cohesion for the program. Any modules that are required to perform some whole function should be grouped together. The other two considerations frequently apply to functional groups as well.

If a group of activities repeat as part of an iterative sequence, all activities in the group should be together in the program package. Individual modules can be coded and unit tested alone, but they should be packaged for integration testing and implementation.

Grouping modules that access the same data minimizes physical reading and writing of files. The major goal is to read the same data record in any one pass of the processes no more than once. We want to minimize physical I/O because it is the slowest process the computer performs. Grouping modules by data accessed minimizes the frequency of read-

ing. Real-time applications, especially, are vulnerable to multiple reads and writes of the same data, slowing down response time.

Grouping modules by data access is a form of data coupling that minimizes the chance of unexpected changes to data. If we do not package modules together, but only read and write data once, the major alternative to common packaging is to use global data areas in memory. Global data is not protected and is vulnerable to corruption.

When the packages are complete, develop Structured English pseudo-code for the logic functions being performed. Use only structured programming constructs—iteration, selection, sequence. Document complex logic using decision tables or decision trees. Include control structures and names for all modules. Pseudo-code may have been done as part of analysis, or earlier in design, as we did for ABC rental and return. Incidental activities, or less crucial activities, may have been overlooked or not refined. Pseudo-code is completed now and structured for use in program specifications.

Decision tables and trees might be used to document complex decisions. While a discussion of them is beyond this text, an example of each is shown in Figure 8-31.

As we design the program packages, we logically test the user views developed with the DBA to reevaluate their usefulness. The questions to ask are: Is all the needed data available? Is security adequate? Is extra data present? If any of these answers indicate a problem, discuss it with the DBA and determine his or her reasons for the design. If the design should change, the DBA is the person to do it.

Design each module to have one entry and one exit. Multiple entrances and exits to program modules imply problems because of selection and goto logic required to implement multiple exits and entrances. If each module is kept simple with one of each, there are fewer testing, debugging, and maintenance problems.

Ideally, each module should have its internal data contents the same before and after a given execution. That is, the state and contents of the module should be unchanged from one execution to the next. This does not mean that no changes take place during an execution, only that all traces of changes are removed when the execution is complete. When a module must maintain a 'memory' of its last actions, coupling is not minimized.

Design and document messages for called modules. Messages should contain, at most, calling/called module names, data needed for execution, control couples, and variable names for results of execution.

You might draw a diagram of the module and all other modules with which it interacts to facilitate visual understanding of the module and its role in the application.

ABC Video Example Program Package Design

Working with the final structure chart in Figure 8-30, our biggest decision is whether or not to package all of rental/return processing together, and how. Do we write one program with performed modules, one with called modules, or a combination of the two?

ABC is going to be in a SQL-compatible database environment, on a LAN, and requires access by PCs. The choice of language is not limited with these requirements, but packaging without knowing the language is not recommended. For this exercise, we will assume that Focus,[4] the 4GL, will be used.

Focus' application generator, called the "Dialogue Manager," allows both in-line and called modules to be used. Calling modules of nonFocus languages are allowed but can be tricky. The language has its own DBMS that is SQL-compatible, but it is not fully relational. It falls in the category of DBMSs called 'born again relational,' that is, the DBMS is hierarchic, networked, or relational at the DBA's discretion. Relationality is allowed but not required in Focus. Focus does not support the integrity rules. We will not redesign the database here since the SQL code above could be recoded without design changes in the Focus DBMS language.

At this point, we need to step back and decide how to package the entire application. What kind of 'glue' will hold customer maintenance, video

4 Focus is a trademark of Information Builders Inc., New York. Focus is representative of PC-based application generators, including Rbase, Dbase IV, Informix, etc.

Decision Table Format:

Conditions—Possible occurrences	**Rules**—Specific occurrences
Actions—Possible outcomes	**Entries**—Specific outcomes for rule combinations.

Decision Table Example:

Conditions										
Customer	Old	Old	Old	Old	Old	Old	Old	Old	New	New
Open Rentals	Y	Y	Y	Y	N	N	N	N	—	—
Returns	Y	Y	N	N	Y	Y	N	N	—	—
New Rentals	Y	N	Y	N	Y	N	Y	N	Y	N
Actions										
Create Customer	N	N	N	N	N	N	N	N	Y	Y
Check Late Fees	Y	Y	Y	Y	Y	Y	N	N	Y	N
Process Return	Y	Y	N	N	Y	Y	N	N	N	N
Process New Rental	Y	N	Y	N	Y	N	Y	N	Y	N
Process Fees and Money	Y	Y	Y	Y	Y	Y	Y	N	Y	N
Update Open Rental	Y	Y	Y	Y	Y	Y	N	N	N	N
Create Open Rental	Y	N	Y	N	Y	N	Y	N	Y	N
Print Receipt	Y	Y	Y	Y	Y	Y	Y	N	Y	N

Decision Tree Format: Tree structure showing conditions and actions.

Decision Tree Example:

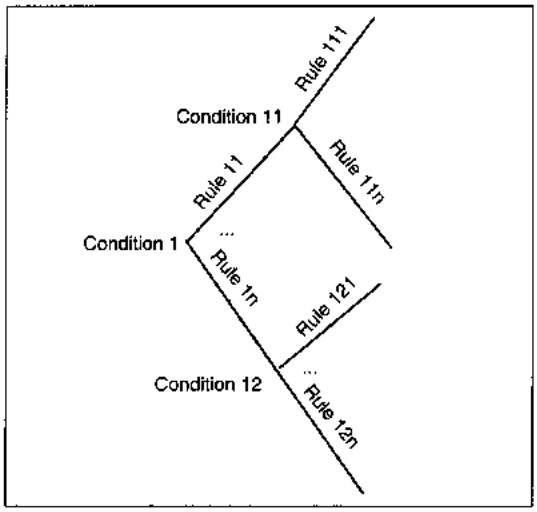


FIGURE 8-31 Decision Table and Decision Tree

maintenance, end-of-day, and rental/return processing together. We do not discuss screen design here because it is not in the methodology (it is in Chapter 14), but we would finalize screens while these decisions are being made. We need all of the above functions to do this application, so all of the functions must be available in a unified environment. This means that all functions must be available for execution within the same run environment. Screens are the 'glue' that users see that unify application processing. The code behind the screens may or may not be unified depending on the design techniques and language. With Focus, unification is done through the Dialogue Manager.

4GL and PC-DBMS languages are deceptively simple. To perform trivial tasks is easy, but to build application requires expertise. Focus is no different. The complexities with Focus relate to when, where, and how often the databases are opened and processed, how the databases are related, and how many concurrent users are allowed. The concurrent environment increases DBA complexity but changes the answers to the questions about databases; it does not change the application code. So, we will assume one user at a time for processing.

Skeleton Focus code for the application is shown in Figure 8-32. Each DFD Level 0 process is accounted for at this level; we even have a query function that is new. Most applications require interactive file query and we have not talked about it at all as part of the rental return application. The trend in business today is for users to develop their own reports and queries using some 4GL. When the language has a built-in query facility, you can add it to the processing without any analysis or design work, as shown here with Focus. User developed queries allow users to 'stay in touch' with their data and remove a major design burden from IS personnel.

Now that the application is accommodated within one execute environment, we return to the problem of how to package rent/return processing. The ideal is to code and unit test each lowest level box on a structure chart as an independent module. Then, using the 'call' feature of the language, build a control structure, based on the design of the control and coordination boxes on the structure chart that calls modules as needed for execution. We will use this approach here as Figure 8-32 shows for the application, and Figure 8-33 shows for *rental* and *return* processing.

The alternative to called modules is in-line code that is 'performed' or executed as a pseudo-called module. This choice is selected with 3GL languages such as Cobol, Fortran, or PL/1 because it can be easier to code, test, and maintain.

Specify Programs

Rules for Specifying Programs

The specification documents all known information about programs. Program specifications document the program's purpose, process requirements, the logical and physical data definitions, input and output formats, screen layouts, constraints, and special processing considerations that might complicate the program. Keep in mind that the term *program* might also mean a module within a program or an externally called function, or even a code fragment (e.g., DB call). A program specification should include the items shown in Table 8-7. As with program packaging, there are no 'rules.' Rather there are items that should be included if they relate to the item being specified.

There are two parts to a program specification: one identifies interprogram relationships and communication, the other documents intraprogram processing that takes place within the individual program. Interfaces to other programs generally document who, what, when, where, and how communication takes place. *Who* identifies who initiates the communication and who, in the real world, is responsible for the interface. *What* identifies the message(s) content that is used for communication. *When* identifies the frequency and timing of the interface. *Where* locates the application and system in a hardware environment; where becomes complicated and is crucial to processing of distributed applications. *How* describes the nature of the interface—internal message, external diskette, and so forth.

Internal program processing information includes the data, processes, formats, controls, security, and constraints that define a particular program.

Focus Code	Explanation
-Set &&Globalvariables	Set variables needed for intermodule communication.
-Include Security	Check password in a security module.
-Run	Check password before any other processing.
-*	Comment indicator
-Mainline	A label identifying the main routine.
-Include Mainmenu	The call statement in Focus is 'INCLUDE.' Mainmenu is a module name.
-Run	Perform Mainmenu before any other processing.
-If &&Choice eq 'R' goto RentRet else -If &&Choice eq 'V' goto Vidmain else -If &&Choice eq 'D' goto EndOfDay else -If &&Choice eq 'Q' goto Query else -If &&Choice eq 'S' goto StopSystem else	Interrogate the choices from Mainmenu to decide what to do next. If in error, go back to the Mainmenu screen.
-Goto Mainmenu; -*	
-RentRet	RentReturn Label
-Include RentRet	Call Rent/Return processing.
-Run -Goto Mainline -*	When Rent/Ret is complete, return to the Mainmenu.
-Vidmain -Include Vidmain -Run -Goto Mainline -*	Video Maintenance Label and Processing
-Custmain -Include Custmain -Run -Goto Mainline -*	Customer Maintenance Label and Processing
Query -Include Tabltalk -Run -Goto Mainline -*	Query Label and Processing
-EndOfDay -Include Endofday -Run -Goto Mainline -*	End-of-Day Label and Processing
-StopSystem	Stop System Label
-End	End Processing

FIGURE 8-32 ABC Video Processing Focus Mainline

RentRet Focus Mainline Code

```
-Set &&Globalvariables
-*Rental and Return Processing
-Crtform Line 1
-"    ABC Video Rental Processing System <d.&date"
-"          Rentals and Returns"
-""
-""
-"        Scan or enter a card or video: <&&Entry"
-If &&entry like 't&' goto Return   else
-If &&entry like 'c&' goto Rental   else
-Include Entryerr;
-Run
-Return
-Include ValidCus
-Include OpenRent
-Include ValidVid
-Goto exit
-Run
-Rental
-Include FirstRet
-Include OpenRent
-Crtform Line 15
-"        Do you want to do rentals? <&&Rentresp/1"
-If &&Rentresp ne 'y' goto exit   else
-Include ValidVid
-Goto exit
-Run
-Exit
-End
```

FIGURE 8-33 ABC Rent/Return Focus Mainline

Frequently, program specifications also include a flowchart of the program logic, a system flowchart showing the system names of the files, and a detailed specification of timing and other constraints.

ABC Video Example Program Specification

The program specification for one program to perform *Get Valid Customer* is shown as an example (see Table 8-8). Since this is a compilation of already known information there is no discussion.

TABLE 8-7 Program Specification Contents

Identification
Purpose
Characteristics
Reference to Applicable Documents
DFD and Structure Chart (possibly also System Flowchart and Program Flowchart)
Narrative of procedures in Structured English, Decision Tables, Decision Trees
Automated Interface Definition
Screen Interface
Screen Design, Dialog Design, Error Messages
Application Interface
Communications Messages, Error Procedures Frequency, Format, Type, Responsible person
Input, Output, and System Files
Logical data design
User views, internal name, graphic of physical data structure
List of physical data structures
Tables and Internal Data
Internal name, graphic of physical data structure
List of physical data structures
Reports
Frequency, Format, Recipients, Special processing

AUTOMATED SUPPORT FOR PROCESS-ORIENTED DESIGN

Automated support in the form of CASE tools is also available, although fewer products support structured design than support structured analysis. Several entries provide *Lower CASE* support that begins

TABLE 8-8 ABC Example *Get Valid Customer* Program Specification

Identification:	*Get Valid Customer*, (ValidCus)
Purpose:	Retrieve Customer Record and verify credit worthiness
Characteristics:	Focus Included module
References:	See System Specification, Pseudo-code for CustMain
DFD:	Attached as Appendix 1
Structure Chart:	Attached as Appendix 2

Narrative:

```
Accept CPhone
Read Customer Using CPhone
If read is successful
   If CCredit lc '1'
      continue
   else
   Display "Customer has a credit problem; rating = <CCredit"
   Display "Override or cancel? : <&Custcredit"
   If &Custcredit eq 'C'
      include Cancel1
      Return
   else
   If &Custcredit eq 'O'
      continue
   else
   include crediterr
   return
else
Include CreatCus.
Set &&ValidCus to 'Yes.'
Set global customer data to values for all fields.
Return.
```

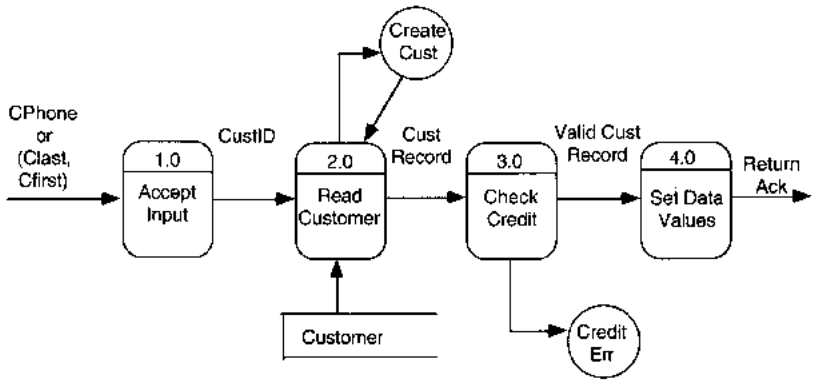
Screen Interface

```
Screen Design: None
Dialog Design: None
Error Messages:
  "Customer has a credit problem; rating = <CusCredit"
  "Override or cancel? : <&Custcredit"
```

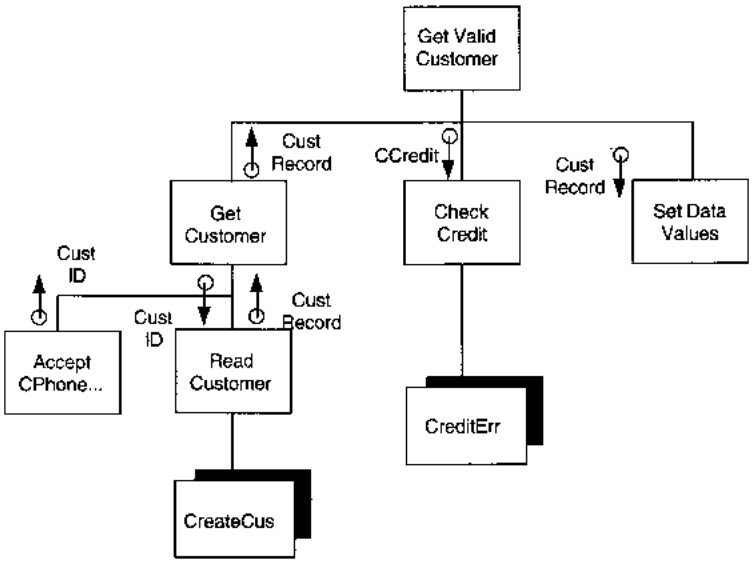
Application Interface	None
Input:	Customer File
User views	Customer
Internal data names:	Customer Contents in Data Dictionary
Tables and Internal Data	
	Global fields correspond to all Customer File fields. Set all fields to customer record values upon successful processing.
Reports:	None

TABLE 8-8 ABC Example *Get Valid Customer* Program Specification (*Continued*)

Appendix 1: Data Flow Diagram



Appendix 2: Structure Chart



Appendix 3: User View with Data Names

Table Customer

			State	Char(2)	Not null,
(Cphone	Char(10)	Not null,	Zip	Char(10)	Not null,
Clast	VarChar(50)	Not null,	CCtype	Char(1)	Not null,
Cfirst	VarChar(25)	Not null,	Ccno	Char(17)	Not null,
Cline1	VarChar(50)	Not null,	Ccexp	Date	Not null,
Cline2	VarChar(50)	Not null,	CCredit	Char(1),	
City	VarChar(530)	Not null,	Primary key	(Cphone));	

TABLE 8-9 CASE Tools for Structured Design

Product	Company	Technique
Analyst/Designer Toolkit	Yourdon, Inc. New York, NY	Structure Chart
Anatool, Blue/60, MacDesigner	Advanced Logical SW Beverly Hills, CA	Structure Charts Structured English
The Developer	ASYST Technology, Inc Napierville, IL	Structure Chart Operations Process Diagram Systems Flowchart
Excelerator	Index Tech. Cambridge, MA	Structure Chart Flowchart
IEW, ADW (PS/2 Version)	Knowledgeware Atlanta, GA	Structure Chart
Maestro	SoftLab San Francisco, CA	Nassi-Schneiderman Hierarchic input-process-output charts (HIPO) User Defined Functions
MacAnalyst, MacDesigner	Excel Software Marshalltown, IA	Decision Table Structured English Structure Chart
Multi-Cam	AGS Mgmt Systems King of Prussia, PA	Structure Chart

with program specification or code generation (see Table 8-9).

STRENGTHS AND WEAKNESSES OF PROCESS ANALYSIS AND DESIGN METHODOLOGIES

The objectives of structured analysis and design are reasonably clear; the manner of obtaining the objectives is much less clear. Structured methods rely on the individual SE's expertise to design the technical details of the application. For implementation specific details, that makes sense, but the heuristics for evaluation cannot be applied in every situation. Consequently, the SE must know what situations apply and don't apply. More than the other methods discussed in this book, you must know when to adhere to, bend, and break the rules of structured methods.

The methodology's ability to result in minimal coupling and maximal cohesion is low because of its reliance on the SE's ability. If coupling and cohesion are not optimal, maintenance will cost more than it should, and the application will be difficult to test. In 1972, D. Parnas wrote about maximal cohesion and minimal coupling as desirable characteristics of programs. In 1968, Dijkstra wrote about the problems with 'go to' statements in programs and proposed goto-less programming. In 1966, Böhm and Jacopini

TABLE 8-9 CASE Tools for Structured Design (*Continued*)

Product	Company	Technique
PacBase	CGI Systems, Inc. Pearl River, NY	Process Decomposition Structure Chart Flowchart
ProKit Workbench	McDonnell Douglas St. Louis, MO	Structure Chart
ProMod	Promod, Inc. Lake Forest, CA	Module Networks Function Networks Structure Chart
SW Thru Pictures	Interactive Dev. Env. San Francisco, CA	Control Flow Structure Chart
System Architect	Popkin Software and Systems, Inc. NY, NY	Flowchart Structure Chart
Teamwork	Cadre Technologies, Inc. Providence, RI	Control Flow Decision Table Structure Chart
Visible Analyst	Visible Systems Corp. Newton, MA	Structure Chart
Telon, and other products	Intersolv Cambridge, MA	Code Generation for Cobol-SQL, C and others
vs Designer	Visual Software, Inc. Santa Clara, CA	Structure Chart Warnier-Orr

proposed structured programming's minimalist contents as sequence, iteration (e.g., if . . . then . . . else) and selection (e.g., do while and do until). By the time structured analysis and design were documented in books, the notions of coupling and cohesion were understood fairly well; but how to obtain them was not.

General statements about keeping the pieces small and related to one part of the problem domain rely on the analyst to know what to do and when to start and stop doing it. Unfortunately, only experience can guide such vague suggestions. While novices can learn to rely on the methodology to guide their actions, they have no basis for evaluating the correctness or incorrectness of their work. Thus, the apprenticeship approach, with a junior person working with a more senior one to learn how to evaluate designs, is required. The more complex the application, the more important having experienced senior analysts becomes.

Another problem is that structured design does not encompass enough of the activities to make it a complete methodology. We must have screen designs in order to develop a program specification. We must know the details of interfaces to other applications and messages to/from them to be able to develop program specifications. Structured methods do not pay any attention to either of these issues. To develop an application, the SE needs to analyze requirements and design for control, input, output, security, and recoverability. None of these are encompassed in the process-oriented methods. To summarize, process methods are useful in analyzing and designing applications that are procedural in nature;

but the methods omit a great many required analysis and design activities.

SUMMARY

In this chapter, structured design which follows structured analysis in development, was discussed. The results of structured analysis—a set of leveled data flow diagrams, data dictionary, and procedural requirements—are the inputs to the design process. The major results of structured design are program specifications which detail the mapping of functional requirements into the production hardware and software environment.

First, using either transaction or transform analysis, the DFD is partitioned into afferent, efferent, and central transform processes. The streams of processing are factored to develop a structure chart. The processes are further decomposed into system-like subprocesses until further decomposition would change the nature of the process. Data requirements are documented in data couples; control is documented in control couples. The chart is evaluated for fan-out, fan-in, skew, cohesion, coupling, scope of effect, and scope of control. The structure chart is revised and reevaluated as required.

The physical database is designed. Data needs for each data flow in the application are listed by process. Data similarities are matched and used to define user views. The access method and physical mapping are then decided. Physical database design walk-throughs may be held to validate the design. Test and production databases are created.

Program packages are decided based on the application concept and timing. The packages define which modules will communicate and how. Pseudo-code for processes is finalized and uses only structured programming constructs—iteration, sequence, and selection. Decision tables and trees are used, as necessary, to document complex decisions.

Finally, program specifications are written to document all known information about each module, function, or program. Specifications include data, process, interface, constraint, and production information needed for a programmer to code and unit test the work.

REFERENCES

Alexander, Christopher, *Notes on the Synthesis of Form*. Cambridge, MA: Harvard University Press, 1971.

Böhm, Corrado, and Guiseppe Jacopini, "Flow diagrams, Turing machines, and languages with only two formation rules," *Communications of the ACM*, Vol. 9, #5, May 1966, pp. 366–371.

Couger, J. D., M. A. Colter, and R. W. Knapp, *Advanced System Development/Feasibility Techniques*. NY: John Wiley & Sons, 1982.

Curtis, B., M. I. Kellner, and J. Over, "Process modeling," *Communications of the ACM*, Vol. 35, #9, September 1992, pp. 75–90.

DeMarco, T., *Structured Analysis and System Specification*. NY: Yourdon, Inc., 1978.

Dijkstra, Edsgar W., "Go to statement considered harmful," *Communications of the ACM*, Vol. 11, #3, March 1968, pp. 147–148.

Flaatten, P. O., D. J. McCubbrey, P. D. O'Riordan, and K. Burgess, *Foundations of Business Systems*, 2nd ed. NY: The Dryden Press, 1992.

Frances, B., "A window into CASE," *Datamation*, March 1, 1992, pp. 43–44.

Krasner, J., J. Terrel, A. Lindhan, P. Arnold, and W. H. Ett, "Lessons learned from a software process modeling system," *Communications of the ACM*, Vol. 35, #9, September 1992, pp. 91–100.

Lindholm, E., "A world of CASE tools," *Datamation*, March 1, 1992, pp. 75–81.

McClure, C., *The Three R's of Software Automation: Re-Engineering, Repository and Reusability*. Englewood Cliffs, NJ: Prentice-Hall, 1992.

McMenamin, S. M., and J. F. Palmer, *Essential Systems Analysis*. NY: Yourdon, Inc., 1984.

Olle, T. W., J. Hagelstein, I. G. MacDonald, C. Rolland, H. G. Sol, F. J. M. Van Assche, and A. A. Verrijn-Stuart, *Information Systems Methodology: A Framework for Understanding*. Workingham, England: Addison-Wesley, 1988.

Page-Jones, M., *The Practical Guide to Structured System Design*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall. 1988.

Parnas, David L., "One of the criteria to be used in decomposing systems into modules," *Communications of the ACM*, Vol. 15, #12, December 1972, pp. 1053–1058.

Swartout, W., and R. Balzer, "On the inevitable intertwining of specification and implementation," *Communications of the ACM*, Vol. 25, #7, July 1982, pp. 438–440.

Yourdon, E., and L. L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Englewood Cliffs, NJ: Prentice-Hall, 1979.

Yourdon, E., *Modern Structured Analysis*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

BIBLIOGRAPHY

Bohl, M., *Introduction to IBM Direct Access Storage Devices*. Chicago, IL: SRA, 1981.
This booklet gives the clearest explanation of VSAM and the differences between data sequenced and entry sequenced storage options that I have seen.

Claybrook, B., *File Management Techniques*. NY: John Wiley & Sons, 1983.
This book provides a good general discussion of indexed, direct, and inverted list files.

Codd, E. F., *The Relational Model for Database Management, Version 2*. Reading, MA: Addison-Wesley Publishing Co., Inc., 1990.
Codd, the father of relationship database theory, argues the merits of an almost direct translation of the logical database to the physical database.

Fabbri, A. J. and A. R. Schwab, *Practical Database Management*. Boston, MA: PWS-Kent Publishing Co., 1992.
This book discusses physical mapping for relational databases and has some discussion of the issues involved for hierarchic and network databases.

KEY TERMS

afferent
afferent flows
atomic process
central transform
cohesion
coincidental cohesion
common coupling
communicational cohesion
content coupling
control coupling
coupling
data coupling
depth of hierarchy
efferent
efferent flows
executable unit
external coupling
factoring
fan-in
fan-out
function
functional cohesion
functional decomposition
HIPO
I/O-bound
indirect coupling
information hiding
in-line code
input-bound
interface
logical cohesion
modularity
module
morphology
Nassi-Schneiderman diagrams
output-bound
partitioning
physical database design
procedural cohesion
process-bound
program package
program specification
program unit
scope of effect
sequential cohesion
skew
span of control
stamp coupling
structure chart
structured design
temporal cohesion
transaction analysis
transaction-centered
transform analysis
Warnier Diagram
width of hierarchy

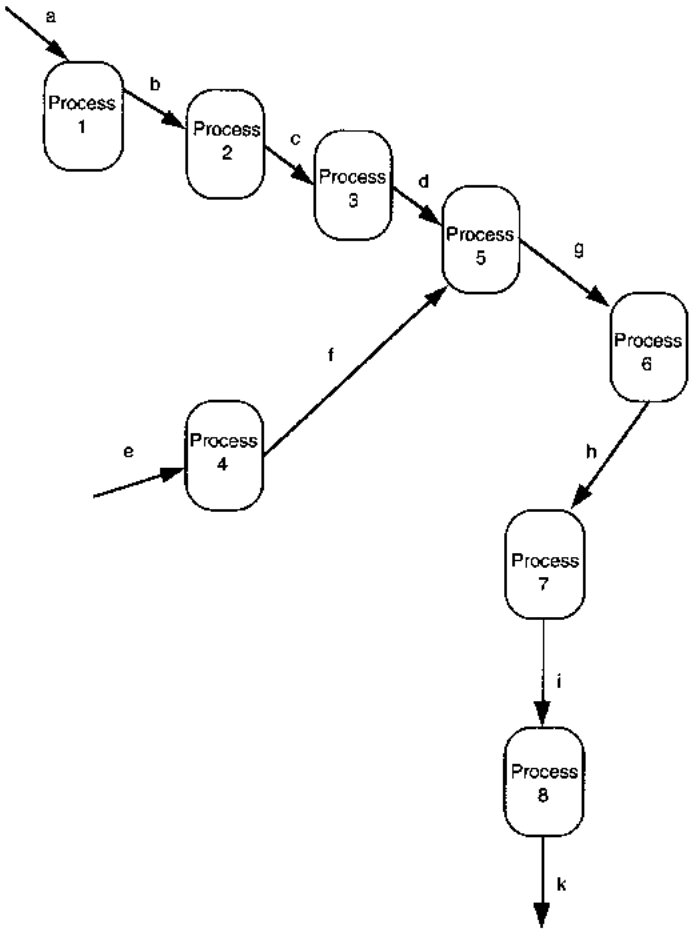
EXERCISES

1. Complete the design for the ancillary processes of ABC rental: customer maintenance, video maintenance, and end-of-day processing. Develop structure charts, including all of the required data and control couples. Evaluate the diagrams and revise as required. Refine the pseudo-code for these functions from Chapter 7. Develop program specifications and identify how the modules will be packaged. Make sure that you state your assumptions about the production environment clearly as part of the explanation of your decisions.
2. What is the linkage between structured analysis and structured design? How do you use the information and documentation from analysis to develop an application design? Do you think analysts and designers should be separate people? Why, or why not?

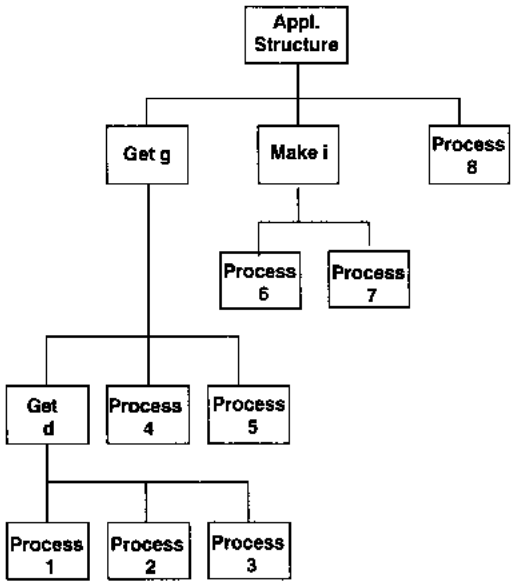
STUDY QUESTIONS

1. Define the following terms:
   cohesion
   coupling
   decomposition
   factor
   function
   input-bound
   module
   morphology
   partitioning
   program package
   program unit
   transaction analysis
   transform analysis
2. How does systems theory relate to structured design?

3. How do you know the difference between a transform centered application and a transaction-centered application?
4. What is the role cohesion plays in the partitioning process? in the decomposition process? in physical database design? in deciding program packages? in program specification?
5. What is the role coupling plays in the partitioning process? in the decomposition process? in physical database design? in deciding program packages? in program specification?
6. What are the major diagrams in the design phase? How are they derived? How do they relate to the work done in structured analysis?
7. What is the reasoning process for packaging program elements?
8. What is the purpose of Structured English? What are alternatives? For what are Structured English and its alternatives used? Why?
9. List the contents of a program specification.
10. Who usually does physical database design? Why would a specialist perform this task? Can SEs do physical database design as well? Why or why not?
11. Partition the following DFD and draw a structure chart. Identify potential afferent and efferent flows. (There are several alternatives for afferents.) Label the flows you decide best describe the processes you see. List other information you need to decide what the best partitioning should be.



12. Evaluate the following structure chart. Describe the morphology. Is this diagram final or does it have problems? If so, what are the problems and how would you fix them?



★ EXTRA-CREDIT QUESTION

1. Perform transform analysis on a case in Appendix A. Design the processing for the central transform from the high-level DFD. Develop lower level DFDs as required to assist your thinking. Factor and develop a first-cut structure chart. Develop pseudo-code for the processes you define. Refine the pseudo-code and finalize the structure chart, giving reasons for your design decisions. Develop program specifications and identify how the modules will be packaged. Make sure that you state your assumptions about the production environment clearly as part of the explanation of your decisions.

CHAPTER 9

DATA-ORIENTED ANALYSIS

INTRODUCTION

Unlike process orientation, data-oriented analysis is not the result of the vision of a small set of people. Rather, it is the collective wisdom of many sources: computer vendors, MIS researchers, and consultants. The philosophy that underlies the data-oriented approach is that *data* are stable and more unchanging than *processes*. Processes can be revised with every reorganization. Data entities, on the other hand, rarely change in the lifetime of a business. Attributes of entities also rarely change. Even though the values of data do change constantly, the structure of the data does not. If data are stable, then they should be examined closely and first.

Data-oriented methodologies teach that data redundancy is to be minimized to best manage it in an organization. Database management software is assumed, but not required, in this approach. **Data administration**, that is, the conscious management of data as a resource of the business, is also assumed.

Information engineering (IE) is the methodology we use to discuss data-oriented analysis. IE teaches that to know *which* data should be the focus, we need architectures of data, business functions, and even organizational technology to guide the process. **Architectures** are conceptual descriptions of the items they define. Architectures are developed at the enterprise level (see Chapter 5). Data and functional architectures are defined further during business area analysis, then are divided into application areas and prioritized. Therefore, multiple application areas can result from one or more **business areas**.

IE methodology defines activities from the strategic organizational level through to implementation of individual applications. The major phases of information engineering are:

1. Enterprise Analysis
2. Business Area Analysis
3. Business System Design
4. Construction
5. Maintenance

In this chapter we discuss the Business Area Analysis (BAA) component of information engineering, which contains the activities that are most similar to analysis in other methodologies. IE analysis is called **Business Area Analysis** (BAA), rather than just *analysis*, because the focus is on *business* data and functions required to do the work. A departure from process-oriented analysis is that information engineering specifically ignores the current business organization, applications, and procedures. IE focuses on how the business *should* work, rather than on how it *does* work. Reengineering of the organization and its applications are common adjunct activities to information engineering (see Chapter 5). In the next

section, we describe the conceptual foundations of data-oriented analysis. Then, the terminology of business area analysis is defined. This is followed by the rules and examples of how to conduct each activity.

Conceptual Foundations

Data-oriented analysis is based mainly on theories about data. Process activities are based on the same systems theory which was the basis for the process development paradigm of Chapters 7 and 8.

The data-related theories are semantic information theory and relational database theory. Semantic information theory seeks to understand the meaning behind the data in applications and is most obvious in the depiction of meaning underlying entity relationship diagrams. By understanding the entities, or *things*, in the application, we know more about their domains—the allowable sets of values they may take. Eventually, rules about domain matching and entity integrity are applied to include domain processing along with data processing of the individual attributes of entities. Relationships between entities are as important as entities and domains. By knowing allowable business relationships, we can constrain processing naturally, by applying business rules, without regard to organizational design. Relationship cardinality, or number, is important to knowing how many of each related item should be evaluated. Cardinality prescribes either individual entity instances or sets of instances for processing. By knowing more about the meaning underlying the data in an application, constraints can be automated and made more general, thus, simplifying the application development process.

Relational database theory is based on mathematical set theory (or relational calculus) which describes allowable operations on sets of data items. Relational theory was developed to support provably correct processing of data items, something that *cannot* be guaranteed by either hierarchic or network database architectures. Set theory is the basis for relational theory which replaces the notion of 'record' processing with 'set' processing. Record processing constrains languages and applications to one-at-a-time record read-manipulate-write processing actions even though most records receive identical treatment in programs. By specifying the rules for processing once and applying those rules to the *set* of data records, or *tuples* as they are called in relational theory, the individual program no longer does any read-write processing—it is performed by the DBMS. Applying set theory, the result of any operation is always a set. Thus, using mathematically based rules, the results of database processing can be known in advance and are provable.

Process activities performed are attributed to consulting practices that work and build on the systems theory underlying the process development paradigm. Some problems with DFDs are:

- DFDs do not accommodate time.
- DFDs have no implied sequence to processing.
- DFDs assign media to data early in analysis without any real deliberation.

These problems are eliminated in *process data flow diagrams (PDFDs)* that are built during IE analysis. Process methods of decomposition rely on analyst experience in process orientation. Data methods, such as information engineering (IE), provide a business-oriented approach to defining processes. Structured process constructs—selection, iteration, and sequence—are not consciously considered in process methods until structured design. Structured constructs are used in IE analysis to describe process relationships.

Definition of Business Area Analysis Terms

The tasks performed during business area analysis (BAA) are:

1. Data modeling
2. Data analysis
3. Functional decomposition (i.e., process modeling)

4. Process dependency analysis
5. Process data flow diagramming
6. Process/data interaction mapping and analysis

Throughout the analysis, a data dictionary or repository is assumed to be used for documentation. The final step of BAA is completion of the repository for all information found during analysis.

For data modeling, the two major activities are the creation and refinement of an entity-relationship diagram (ERD) and entity structure analysis, along with an accompanying repository. When complete, the **ERD** describes the normalized data environment and data scope of the application. Each part of an ERD requires definition. An **entity type** (shortened to *entity* in this discussion)[1] is some person, object, concept, application, or event from the real world about which we want to maintain data (see Figure 9-1). There are three kinds of entities: fundamental, attributive, and associative. A **fundamental entity**, for instance, an order, is independent of all other entities and can be defined without thinking about other entities. An **attributive entity** is an entity whose existence *depends* on the presence of a fundamental entity. If *order* is the fundamental entity, then *order item* would be an attributive entity related to order (see Figure 9-2). Technically, you wouldn't have an order without any items, but you *cannot* have an order item without an order. Attributive entities contain repeating information relating to a fundamental entity. An **associative entity** is used to simplify and define complex relationships between entities. All entities are drawn on the entity relationship diagram (ERD) as rectangles.[2]

1 Technically, a customer is an entity who is uniquely described by a set of attributes. The set of all customers describes an entity type which is described by having the same attributes. A specific entity, e.g., customer 'Wells,' is an entity instance. In this text we use entity to be synonymous with entity type.

2 One method of diagramming is to show relationships with a diamond bisecting the line connecting entities. An associative entity, promoting a many-to-many relationship, is drawn as a rectangle with the diamond inside.

	EXAMPLES		
Entity Type	**Insurance**	**ABC Video**	**Manufacturing**
Person	Policyholder	Customer	Customer
Object	Policy	Video	Bill of Lading
Concept	Policyholder Services	Accounting Department	Order
Event	Purchase of Policy	Rental of Video	Shipment of Goods
Organization	State Bureau of Insurance	Vendor	IRS, OSHA

FIGURE 9-1 Entity Type Examples

Entity	**ABC Video**	**Human Resources**	**Manufacturing**
Fundamental	Customer	Employee	Work Order
Attributive	Customer Rental History	Employee Work History	Work Order Detail Items
Associative	Vendor-Video	Employee-Job History	Work Order Item-Finished Part

FIGURE 9-2 Entity Examples

Number		Education Examples	Manufacturing Examples
One-to-One	1:1	Student to Transcript	Work Order Detail Item to Machine/Day/Time Operator
		Course Section to Room/Day/Time	
One-to-Many	1:N	Course to Section	Work Order to Work Order Detail Item
		Transcript to Course	
		Course to Room/Day/Time	Customer Order to Work Order
		Students to Major	Salesman to Customer
		Advisor to Student	
Many-to-Many	N:M	Student to Course	Part to Work Order Detail Item
		Professor to Course	Vendor to Inventory Part
		Professor to Section	

FIGURE 9-3 Relationship Cardinality Examples

A **relationship** is a mutual association between two or more entities. It is shown as a line connecting the entities. A relationship has **cardinality**, or the *number* of the relationship. Cardinalities may be one-to-one, one-to-many, or many-to-many (see Figure 9-3). Cardinality is shown on a diagram by crows' feet to indicate a 'many' relationship and a single line to indicate a singular relationship.

Refinement of the ERD has two activities: attributes are defined, and the ERD is normalized. **Attributes** are named properties or characteristics of an entity which take on values. We use the terms attribute, field, or data item, as synonyms. An **instance** is one occurrence of an attribute or relation. For example, an instance of the attribute customer-ID is the number 2922951.

Normalization is the refinement of data relationships to remove repeating information, partial key dependencies, and nonkey dependencies. Normalization can be directly applied to the ERD or can use a tabular method of data analysis. The **direct method** proceeds by examination of the relationship cardinalities and the attributes of entities. For 1:*n* relationships, and for entities with repetitive information in the entity, we create (or validate) attributive entities. For an *m*:*n* relationship, the relationship is promoted to create an associative entity. A synonym for associative entity is **relationship entity**. The cardinalities of *m*:*n* are reversed to create two 1:*m* relationships (see Figure 9-4).

The **tabular method** is recommended when data and relationships are not clearly specified. The tabular method forces explicit definition of all attributes and their relationships. When these dependencies are removed, each relation's data are fully, functionally dependent on the primary keys. An example is shown in Figure 9-5. By removing repeating information (first normal form), we create attributive entities (for 1:*n* relationships) and associative entities (for *m*:*n* relationships). In Figure 9-5, we create the items from a purchase order as an attributive entity. By removing partial key (second normal form), and nonkey (third normal form) dependencies, we create new fundamental entities. In the example, the new fundamental entities relate to items and vendors.

Upon completion of data modeling, **entity structure analysis** is performed to determine whether a class structure applies. This analysis evaluates each entity to determine if the same processes and

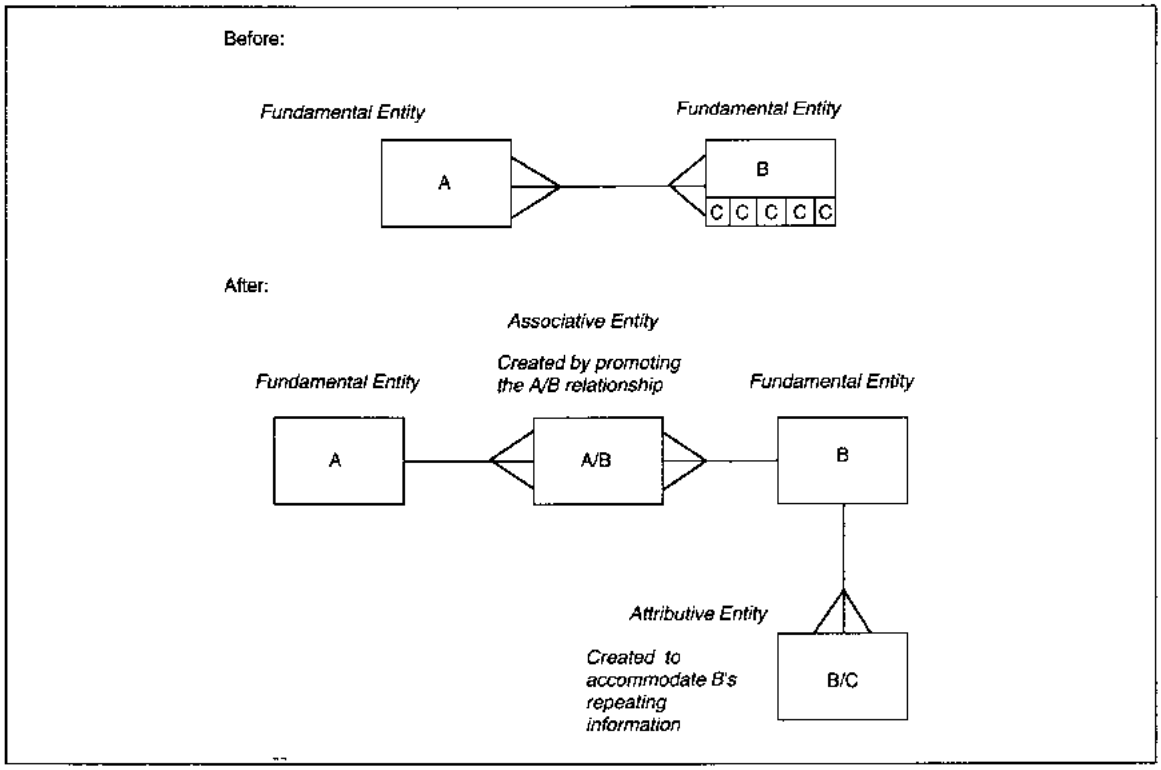


FIGURE 9-4 Direct Normalization of ERD

attributes apply to all entities of a given type. If contingent data usage applies, then classes are defined and a data hierarchy depicting the structure is developed.

Next, business functions are identified as a prelude to process modeling. A **business function** is a group of activities that accomplish some complete job that is within the mission of the enterprise. Business functions are ongoing and are not related to organization structure. Functions describe what is done in the organization from a high level of abstraction. Business function analysis is usually performed at the enterprise level, but can be the first activity of process modeling, if required. Representative or generic functions that may be present in a business are listed below. Some of the functions are specializations, for instance, public protection is usually a government function. Specialized functions included are for banking, retail, governments, schools, and manufacturing. Other functions are general, like Finance, which every organization has.

Accounting
Alumni Affairs
Audit
Community Programs
Control and Measurement
Customer Relations
Data Administration
Distribution
Engineering Support
Facilities, Equipment, and Supplies Administration
Finance
Funds Management
Funds Transfer
Health and Hospitals Services
Human Resources Administration
Information Systems
Judicial Management
Legal Services
Management
Manufacturing
Marketing
Material Acquisition (Purchasing)

Unnormalized	First Normal Form	Second Normal Form	Third Normal Form	Relation Name*
Purchase Order				Purchase Order
(PO) *Number*	*PO Number*		*PO Number*	
PO Date	PO Date		PO Date	
PO Vendor ID	PO Vendor ID		PO Vendor ID	
PO Vendor Name	PO Vendor Name			
PO Vendor Address	PO Vendor Address	→		
PO Ship Terms	PO Ship Terms			PO Vendor
			PO Vendor ID	
PO Payment Terms	PO Payment Terms		PO Vendor Name	
*PO Item Number			PO Vendor Address	
POI Description		*PO Number*	PO Ship Terms	
POI Quantity	*PO Number*	*PO Item Number*	PO Payment Terms	
POI Price	*PO Item Number*	POI Quantity		
POI Extended Price	POI Description	POI Price		
	POI Quantity	POI Extended Price		
	POI Price			PO Item
			PO Number	
	POI Extended Price		*PO Item Number*	
			POI Quantity	
		Item Number	POI Price	
		Description	POI Extended Price	X
		Price		
			Item Number	Inventory Item
			Description	
			Price	

*X indicates deleted items or relations. Relations are deleted if they are duplicates, are consolidated if they have identical keys or are proper subsets, or are named. Attributes are deleted if they are derived by the application. POI Extended Price is derived by multiplying POI Quantity by POI Price.

FIGURE 9-5 Tabular Normalization Example

Operations
Planning
Product Management
Product/Customer Service
Public Aid
Public Facilities Management
Public Protection Management
Public Relations
Public Service
Research and Development
Research
Sales
Scheduling
Service Offering, e.g., Instruction in a school
Student Management

Sample business functions for ABC Video are shown in Figure 9-6.

When the functions applicable to application development are identified, functional decomposition is performed. **Functional decomposition** starts at the business function level to identify the major activities of the function, and progresses to identify the processes and subprocesses for each function (see Figure 9-6). An **activity** is some procedure within a business function that can be identified by its input data and output data, which differ. The

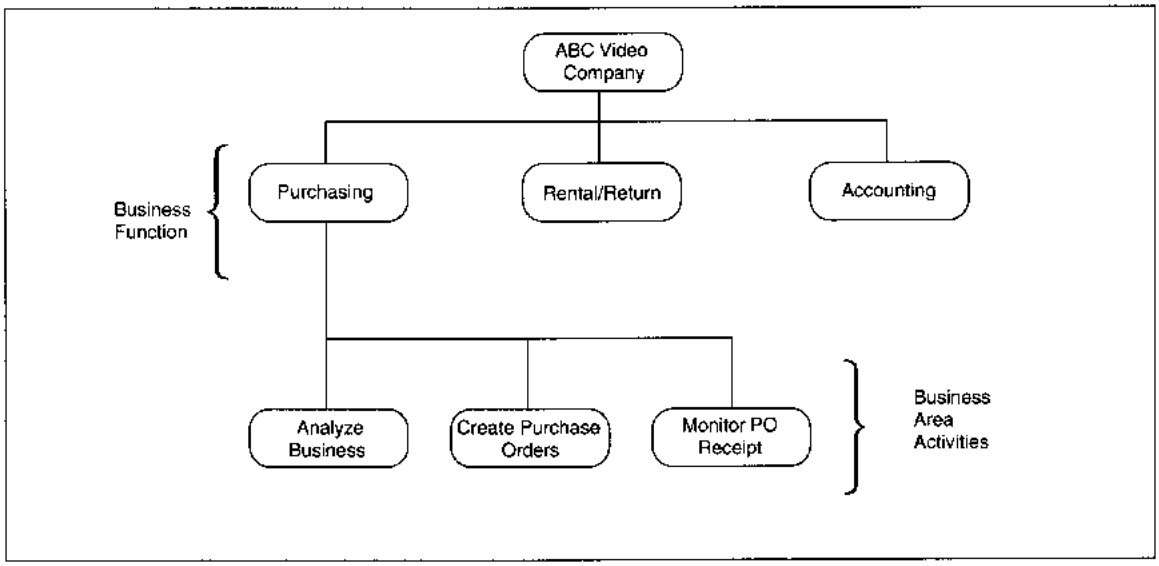


FIGURE 9-6 ABC Video Business Functions and Activities

activity level must *fully define* the function. That is, the activity level is complete when all possible procedures performed within the scope of the function are present in the diagram. Full definition is required to ensure complete data, process, impact, and organization design analysis.

Activity names are usually of the form *verb-object*, where the verb identifies the major transformation and the object identifies what is transformed. Exceptions to this rule are accepted when a name is conventionally called by a different form, for instance, *Cash Management* is more common usage than *Manage Cash*.

Activities are decomposed into their processes. A **business process** identifies the details of an activity, *fully defining* the steps taken to accomplish the activity. Again, full definition is required to ensure completeness of the ensuing analysis. Procedural steps named by processes are repeated and have definable beginnings and endings. Decomposition continues until the elementary, or atomic, level of each process is identified. An **elementary process** is a procedure that cannot be decomposed further without making the procedure lose its identity. Thus, an elementary process is the smallest unit of work users identify.

Figure 9-7 is a sample decomposition showing processes that define the two purchasing activities within ABC Video. Don't forget that the business activities and processes in a decomposition *fully* define the scope of the parent business function.

Decomposition results are used to develop a process dependency diagram. A **process dependency diagram**, like an ERD for data, identifies the sequence and types of relationships between processes. **Process relationships** describe logical connections that include cardinality, sequence, iteration, and selection components (see Figure 9-8). Thus, the process dependency diagram shows the logic of sequence, iteration, and selection for each process. The process dependency diagram is then expanded to include entities and data stores to emulate a data flow diagram from process analysis. The result is a **process data flow diagram (PDFD)**.

Connections between procedural steps in a PDFD are due to data passing from one step to the next and causing it to activate. This type of connection is called a process data trigger. A **trigger** identifies the

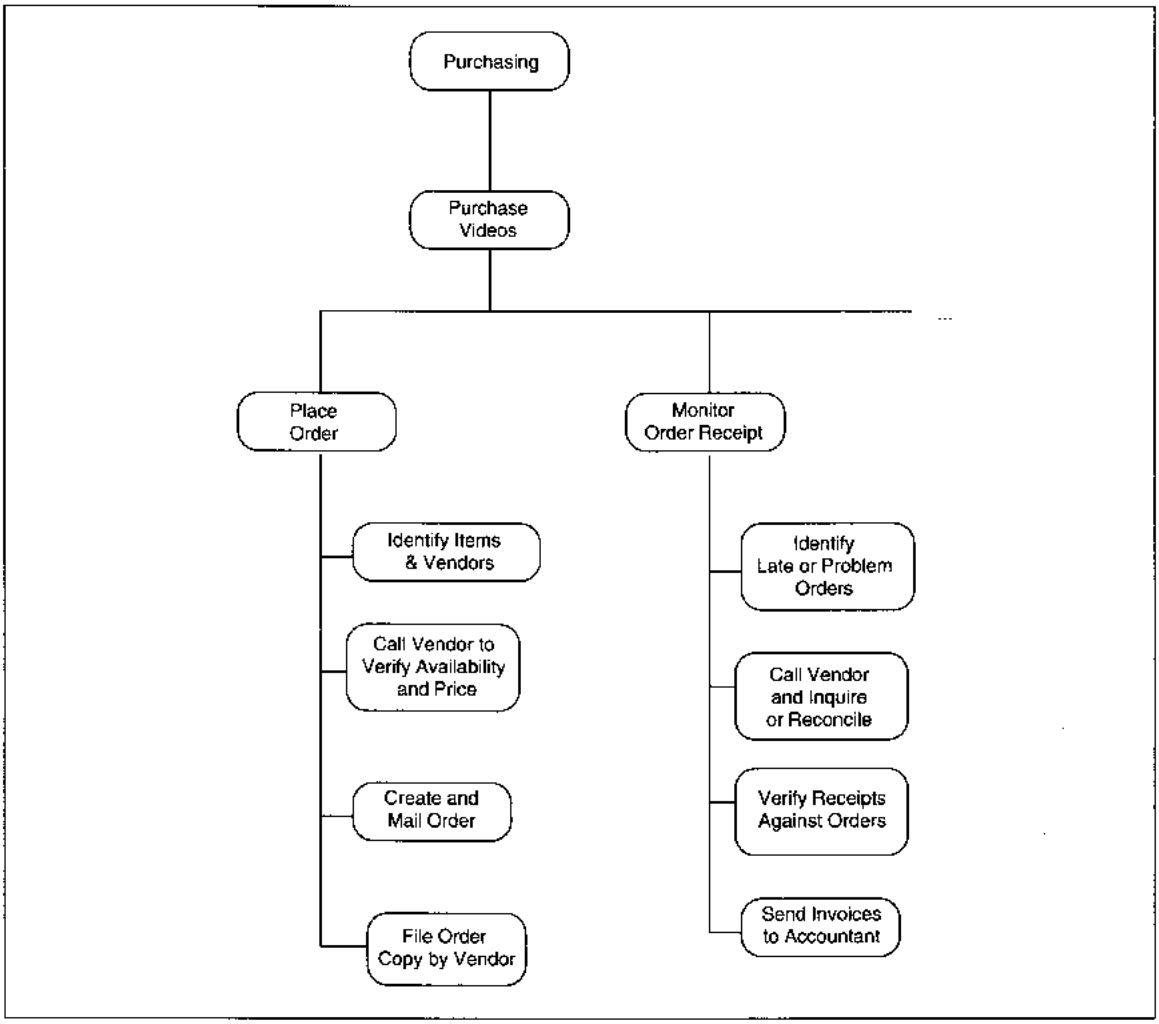


FIGURE 9-7 ABC Video Partial Functional Decomposition of Purchasing

arrival of some data that causes a business process to execute. **Process data triggers** (or just **data triggers**) identify data that flow from one process to another to start execution of the receiving process. In a PDFD, the directed lines between processes signify a data trigger. In addition, external events can cause a process to activate. An **event trigger** signifies data from some business transaction that causes processing to take place. Event triggers are drawn on the PDFD by large arrows with words inside the icons to name the events. For instance, the arrival of a new video releases list (see Figure 9-9) is an event that triggers the purchasing process.

Because the components of the process dependency diagram are different from those of a DFD, the PDFD that results from process dependency analysis is also different. Several key differences are important. First, there is a *sequence* to the process data flow. The directed arrows on Figure 9-9 indicate that some output from a process causes

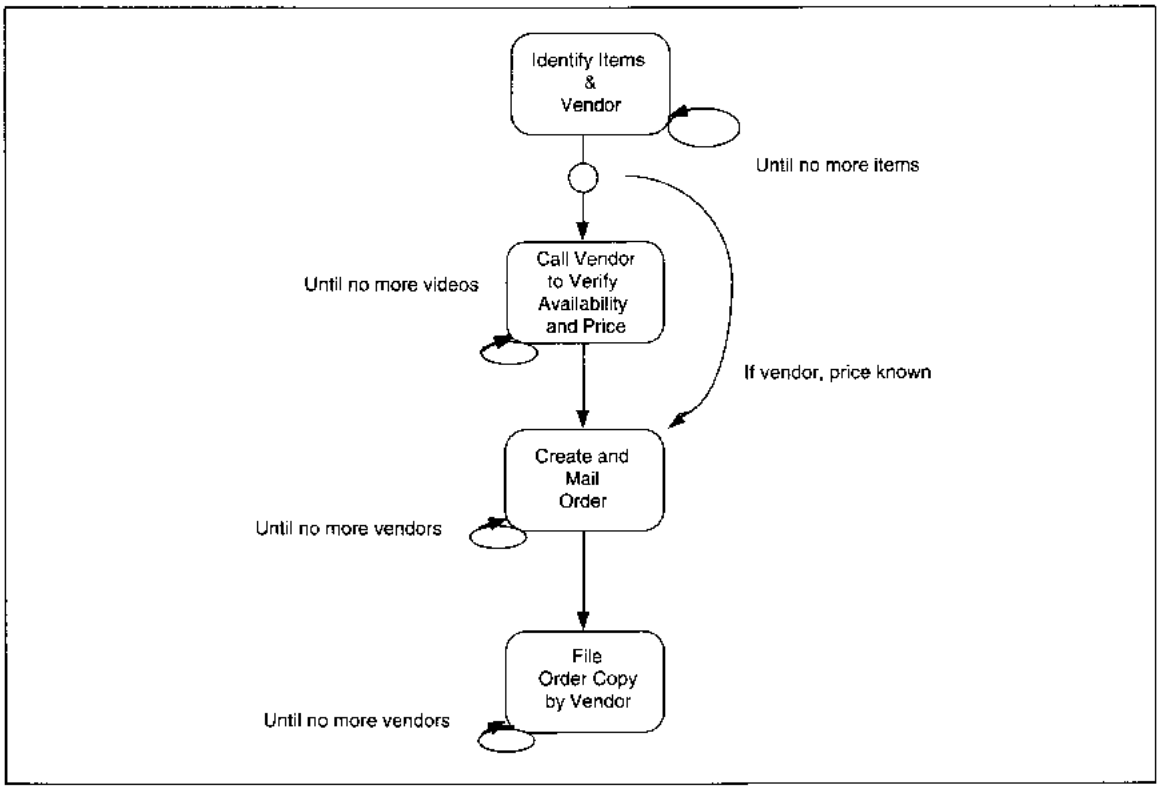


FIGURE 9-8 ABC Create Order Process Dependency Diagram

the execution of the next process. Variations in the directed arrow lines define variations in the sequence. Second, the *media* that connect processes are *not* implied as in a DFD.[3] The information that passes between processes *is* identified, but the form of the data is not. For example, the *Identify Items and Vendors* process in Figure 9-9 generates data that passes to later processes. The shared data might be mental, paper, an automated data flow, or a file. The decision of media, or *stored form*, of data is deferred until design unless it is fixed. Data files, such as *Vendor* and *Order* files on Figure 9-9, are identified because they are known. Third, data and event triggers *identify* the cause of execution of each process. In a DFD, this information either is characterized as a data flow or is hidden within process logic.

The last step of BAA is the development and analysis of an entity/process matrix, also known as a **CRUD matrix**. If no enterprise level ERD exists first, then an ERD is created. The **entity/process matrix** lists entities across the top and business processes down the side (see Figure 9-10). Each cell of the matrix, then, points to a process-entity combination. For each cell, the systems engineers define Create (**C**), Retrieve (**R**), Update (**U**), Delete (**D**), or no (blank) responsibility of each process for each entity. Subject area databases are defined by analyzing logical groupings of processes and entities based on their affinity. **Affinity** means 'attraction' or 'closeness.' **Affinity analysis** clusters processes which share data creation authority for an entity.

3 Remember, DFDs require identification of either a *data flow* or a *data store* as the data linkages between processes.

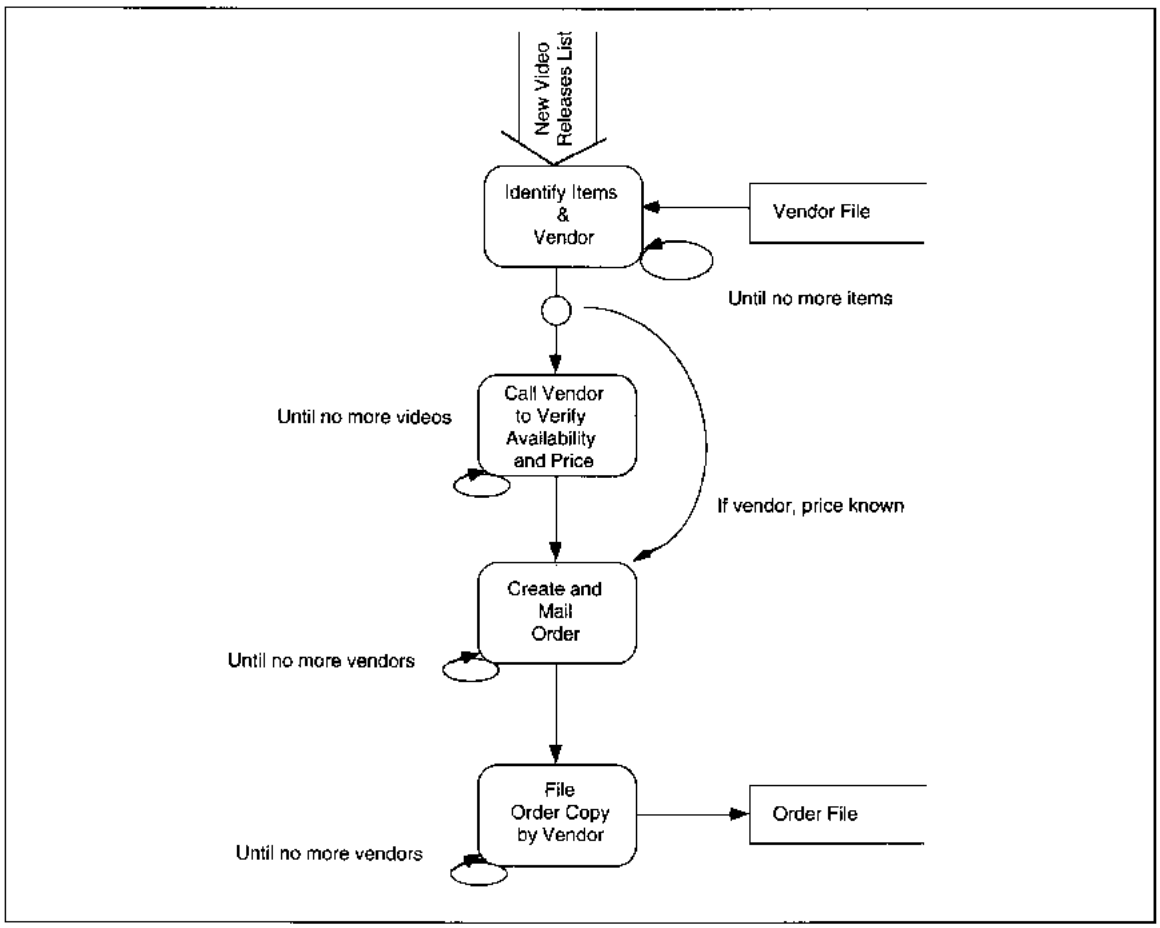


FIGURE 9-9 ABC Create Order Process Data Flow Diagram

These logical groupings become the basis for database design. In Figure 9-10, a partial example of *Create Order* and *Monitor Order Receipt* processes, and the entities they use, are analyzed in an entity/process matrix. The matrix shown is clustered by entity affinity and is ready for analysis. After analysis, the processes and entities are sorted to show affinity based on the actions taken on the same entities (see Figure 9-11).

Two sets of analysis are performed on the results of affinity analysis. The first analysis is to determine the adequacy of organization design based on who creates and has responsibility for data. Each cluster of processes is related back to the organization (in a similar matrix). Ideally, processes that share data responsibility should be in the same organization and report to the same manager. For instance, the ABC Purchasing processes show three potential groupings. If each process is evaluated with all of the data it uses, the three groupings meld into one based on the criteria that 70% or more of the data are commonly shared. If all of these processes report to one manager, the organization is probably adequate. If the three possible groupings all report to different managers, the organization should probably be redesigned.

Entities = Processes	Purchase Order	PO Item	Inventory Item	Vendor
Identity Items & Vendors			R	CRU
Call Vendor to Verify Avail/Price				RU
Create & Mail Order	CRUD	CRUD	R	R
File Order Copy by Vendor	R	R		
Identify Late & Problem Orders	R	R	R	RU
Call Vendor & Inquire on Order	RU	RU	R	R
Verify Receipts against Order	RU	RU		
Send Invoices to Accountant	RD	RD		

FIGURE 9-10 Create and Monitor Order Receipt Entity/Process Matrix

The second analysis looks at the data entities by process cluster to define subject area databases. A **subject area database** is normalized across the organization and provides shared support for one or more business functions. At the application level, one subject database is assumed. In the ex-ample in Figure 9-11, one database would support the purchasing function; the database would have at least two user views to package *Purchase Order* with *Purchase Order Item* and *Inventory Item* with *Vendor*. A fourth user view linking all entities might be used for retrieval processing.

At the organization level, if the process groupings are logical and useful, they are the basis for reaffirming the scope of applications. At the business area level, the groupings of processes should be consistent with the scope of the activities defined for the application. If they are not consistent, then management review and rescoping of the project are required.

The last step of BAA is to finalize all information found during the analysis in a data dictionary or CASE repository. Since dictionaries were discussed in detail in Chapter 7, in this chapter we will document the information found using the same format as in Chapter 7, but will not comment again on the format of entries.

To summarize, business area analysis begins with an entity-relationship diagram that is fully identified, normalized, and analyzed for class structure. Business functions are identified and decomposed to create process hierarchy, process dependency, and process data flow diagrams. The business processes from the decomposition are coupled to entities from the ERD. Data-related responsibilities are described for each process. Affinity analysis of the CRUD matrix is used to decide organizational and database groupings for further design and management action. Next, we turn to a detailed description of how to perform each activity, exemplified by the ABC Video Rental Processing application.

Entities = Processes	Purchase Order	PO Item	Inventory Item	Vendor
Create & Mail Order	CRUD	CRUD	R	R
Call Vendor & Inquire on Order	RU	RU	R	R
Verify Receipts against Order	RU	RU		R
Send Invoices to Accountant	RD	RD		
File Order Copy by Vendor	R	R		
Identify Late & Problem Orders	R	R	R	RU
Identity Items & Vendors			R	CRU
Call Vendor to Verify Avail/Price				RU

Subject Area 1

Subject Area 2

FIGURE 9-11 ABC Purchasing Process Affinity Analysis

BUSINESS AREA ANALYSIS ACTIVITIES

Develop Entity-Relationship Diagram

Rules for Entity-Relationship Diagram

The steps to building an entity relationship diagram (ERD) are as follows:

1. Define fundamental entities and their primary keys.
2. Define the relationships between the fundamental entities.
3. Identify all attributes of entities, including primary keys.
4. Add attributive entities, where needed, to simplify one-to-many relationships.
5. Promote all many-to-many relationships to define associative entities.
6. Normalize the fundamental entities, analyzing if there are other entities which are hidden in the current definitions. Place new entities in the ERD. Define the new entities' attributes and primary keys.
7. Analyze the entities and their relationships to determine if a class structure is needed. If some instances of entities have identifiable differences in processing, data stored, or relationship participation, classes probably are needed.

The first step is to define fundamental entities and their primary keys. Identifying entities is a difficult process until you have done it several times. It is easy to talk about entities, but less easy to define them. Part of the difficulty is that entities are context related. An entity for one company/application may not be an entity in another company/

EXAMPLE 9-1

ENTITY DEFINITION IN XYZ ANNUITY

In Example 7-1, we discussed how at the annual meeting of the XYZ board of directors in 1991, the marketing director said that she had four different, irreconcilable counts of the number of institutions the company serviced. What was worse was that there was a defendable definition of each number.

The board thought that was terrible and ordered a redevelopment of the Institutional Processing application to resolve the problem. When Diane Smith, the software engineer, began work on the application, her first task was to develop an ERD for the information, without regard to the current files (12), applications (6), interfaces (4), procedures (28), or time relationships currently used in the organization. Just in sheer numbers, this was a significant amount of information to be ignored.

Twenty-two different people were interviewed, resulting in 22 different definitions of an institution. They included such descriptions as:

- an organization that pays in to a pension plan for its employees
- an organization that requires counseling about products and services provided by us
- an organization we target for marketing campaigns
- an organization that defines a pension plan
- an organization that is subject to a pension plan that may or may not be of its own definition
- an organization that is subject to legally defined pension plans by the state government in which it resides
- an organization that receives information about pension plans of the suborganizations for which it administers plans

Working with a data administrator, Diane and the key users unraveled the spaghetti of definitions to uniquely define major entities for

application. When in doubt, define more entities rather than less. You can always eliminate unnecessary entities when the information for deciding becomes clear.

It is important to define each entity using terms that apply for all of its uses in the company. Such definitions may not match current definitions of the entity in use in the organization. An example in defining the terms from an educational pension firm (see Example 9-1) shows the difficulty dealing with current thinking about entities and their definition. Current thinking is frequently imprecise, muddled, and even inconsistent as the example shows. Unraveling the spaghetti of definitions imbedded in the various terms used to describe institutions, colleges, campuses, plans, and their relationships took three people much of six months, working with 10 user departments for the information.

ERDs depict the *big picture*, capturing the organization and its constituent activities. For this diagram, we must constantly remember to ask: What processes and activities are legal *in the context of the business*? Not: What is legal based on *today's procedures* in our company?

In general, entities define some*thing* about which the business keeps information. An entity can be a person, object, application, concept, or event *about which the application maintains information*. For example, customer, order, and inventory are all entities. Entity names are usually nouns, however, NOT all nouns are entities. First, define a list of possible entities. Then, examine each entry and ask yourself the following:

1. Is this a noun? If yes, continue. If not, either rename it or strike it from the list.

EXAMPLE 9-1 *(Continued)*

the organization. The following definitions, which took six months to attain, fully explain all variations of XYZ Annuity's institutional processing.

XYZ Annuity Entity Definitions

State Optional Pension Plan (SOPP)—An optional pension plan(s) defined by law, governing institution(s) specified in the law. SOPP institutions must adhere only and completely to the legal requirements of the SOPP.

Institution—A legal entity that is governed by an SOPP or, if not, may define its own pension plan(s) subject to Internal Revenue Service limitations.

Campus—A legal entity that is a subunit of an institution. If an institution defines its own plan, one of the plan items specifies whether or not campuses are bound by its definition. If a campus is not bound by the institution plan, it may own its own plan(s).

Plan—A legal description of the product(s) offered, eligibility and waiting period requirements, and other pension plan provisions. Usually, and always after 1992, each plan defines the offering for one product.

Product—A pension service offered by Educational Pension Trust, including individual annuity, group annuity, supplemental retirement, or group supplemental retirement accounts. Each product defined requires definition of one or more investment types allowed.

Investment type—Annuity, Money Market, Educational Pension Trust Stock fund, and Educational Pension Trust Bond fund.

These six definitions sound simple enough to be obvious, but they began with a total of 120 different interpretations.

2. Is this *potential entity* (replace with the name of the *potential entity*) unique with a clearly defined purpose? If yes, continue. If no, either define the item uniquely from the context that led to its being on the list, or strike it from the list.
3. Can this *potential entity* take a value? If yes, it is an attribute; strike it from the list. If no, continue.
4. Does the business area need to keep information about this *potential entity*? If yes, continue. If no, ask why it is on the list. If it triggers processes, continue. If it is a different *form* of some other entity (for instance, an order report is a paper version of an order), strike it from the list. If it is unique but does not fit the other criteria, leave it on the list for now.
5. Give a formal name to the entity and define its primary key.
6. Draw one rectangle for each entity to begin developing the ERD.

Once you are comfortable with the entities, begin defining their relationships. Relationship names, describing entity associations, are usually verbs, however, NOT all verbs describe relationships. The goal is for all rules of association to be unambiguous. First, define possible relationships. In general, ask yourself how entities relate to each other. If I have *Entity A*, do I also have *Entity B*s? If so, how many are legal? Ask the question without regard to each entity's current usage in the company. As with entities, relationships should define what is legal within a business context. Sometimes, ignoring current definition is extremely difficult because we

and users internalize such definitions and use them to narrow our focus and simplify the world.

Examine each possible relationship and ask yourself the following:

1. Is this a verb? If yes, continue. If not, either rename it or strike it from the list.
2. Is this verb an action? If no, continue. If yes, remind yourself that relationships do NOT describe processes or processing. If the verb is a process, strike it from the list.
3. Is this *potential relationship* (replace with the name of the *potential relationship*, e.g., place as in customers *place* orders) unique with a clearly defined purpose? If yes, continue. If no, either define the item uniquely from the context that led to its entry on the list, or strike it from the list.
4. Is this *potential relationship* needed to fully describe the business area's data? If yes, continue. If no, ask yourself why it is on the list. If it is not clear what the reason is, continue, leaving the relationship to be reevaluated when more information is known. If the reason is not related to the business area, strike it from the list.

Once you define a relationship, draw a line(s) to connect the entities participating in the relationship. Mark the diagram with a verb to describe each direction of the relationship. The convention is to read the relationship above the line from left-to-right, and the relationship below the line right-to-left. For instance, customer *places* order and order *is placed by* customer (see Figure 9-12). The words are placed differently depending on the placement of the entities on the diagram. By convention, the active verb (in this example, 'places') is positioned on top of the line with the acting entity ('customer' here) on the left of the diagram.

Next determine the number, or cardinality, of the relationship. The number of the relationship is one of three possibilities: one-to-one, one-to-many, or many-to-many. A **one-to-one relationship** defines a situation in which every entity *A* relates to one and only one entity *B*. In a **one-to-many relationship** every entity *A* relates to zero to *n*, that is, any number of entity *B*s. In a **many-to-many relationship** all *A*s can relate to any number of *B*s. Decide cardinality by asking the same questions of each side of the relationship: If I have one entity *A*, how many entity *B*s can I have associated with it *at any point in time*? Conversely, if I have one entity *B*, how many entity *A*s can I have associated with it at any point in time?

An example of the time issue relates to student registration and tracking. A student may take many classes in one semester; this describes a 1:*n* relationship. Over time, students take many courses and courses contain many students; this is an *m*:*n* relationship. Which is correct? The *m*:*n* relation that

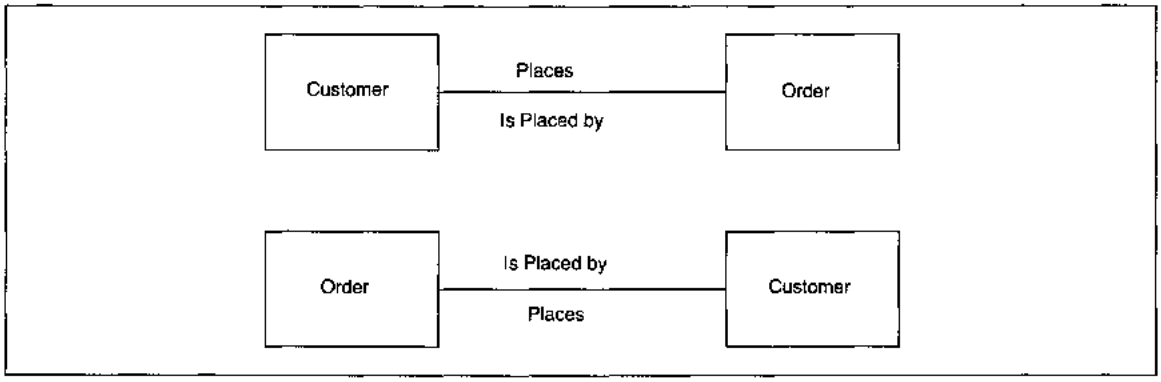


FIGURE 9-12 Placement of Words on Entity-Relationship Diagrams

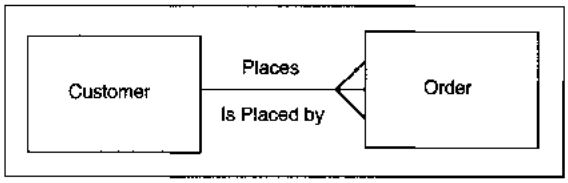


FIGURE 9-13 Relationship Cardinality and ERD Representation

describes the student-course relationship without *regard to time* is correct within the context of a student registration application.

Draw crows' feet to show cardinality of each relationship. Crows' feet are reverse arrow heads that indicate a 'many' numbered relationship.[4] The example in Figure 9-13 shows the relationship of customer to orders as one-to-many. That is, for any one customer, one to many orders may be associated with it. Conversely, the relationship of orders to customers is one-to-one. That is, for any given order, it is associated with one, and only one, customer.

Lastly, for each entity in a relationship, we decide whether the entity is required or optional in the relationship. In a **required relationship**, the entity must be present for the other entity to exist. In an **optional relationship**, the entity described may or may not exist when the other entity exists. Either an 'o' or a vertical bar, '|', shows each side of a relationship as optional (o) or required (|).

Returning to the order example, customers place orders at their discretion, so orders are optional. Customers are required to have been identified as customers to place orders, so customer is required (see Figure 9-14).

In the order-item relationship, orders do not exist if there are no items ordered, so order is required. The vertical bar ('|') bisects the relationship line close to the order entity. Examining the items, we have a similar relationship. For an order to exist, there must be at least one item, so item is also required. Both sides of the relationship line have a vertical bar (see Figure 9-14). 'Read' this entire relationship as follows:

> Orders *contain* items. For each order, there are one or more order-items. For an order to exist, at least one order-item is required. An order-item *is contained in* order. For an order-item to exist, an order is required. For each order-item, there is one, and only one, order.

4 This is the IE convention. There are other techniques for drawing ERDs, such as Chen's [1976]. Chen uses multiple arrow heads for 'many' relationships and uses diamonds to identify relationships with only one verb. The logic of both approaches is identical. Martin's notation is used here because it is automated in CASE tools.

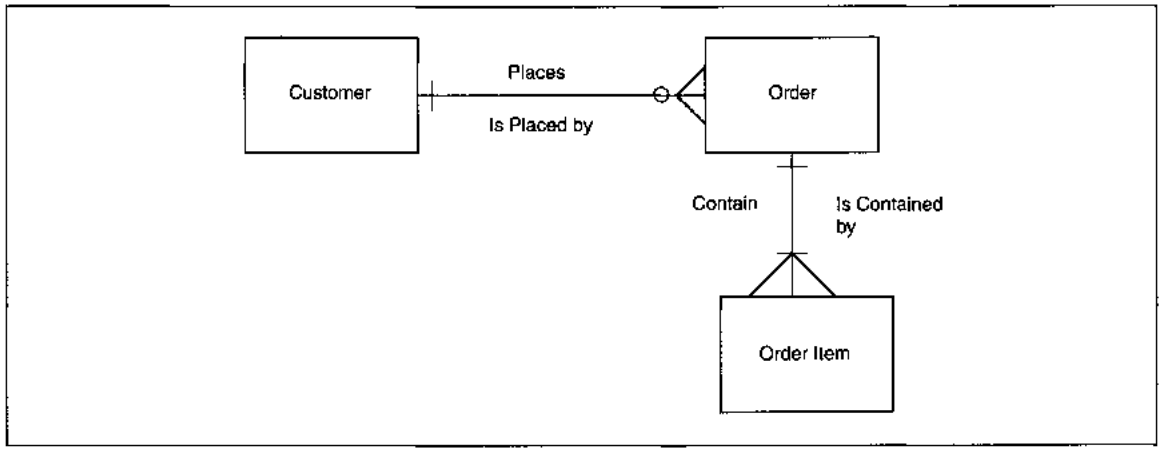


FIGURE 9-14 Required/Optional Relationship Representation

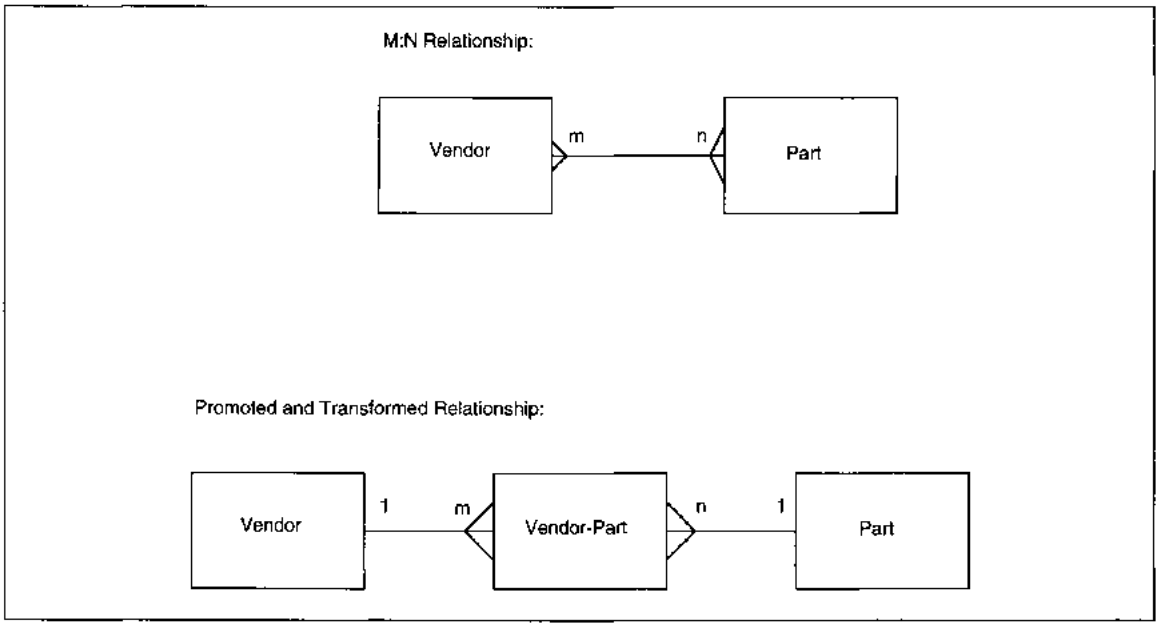


FIGURE 9-15 Many-to-Many Relationship Promotion and Transformation

Similarly, **associative entities** are created by promoting *m*:*n* relationships, joining the primary keys of each participating entity to identify the association. Other fields might also be needed to provide unique identification. The *m*:*n* relationship is converted into two 1:*m* relationships in the promotion process (see Figure 9-15).

After all known relationships are defined and entered on the diagram, we define attributes for the entities and normalize them (Steps 3–6 of list on p. 339). The goal of this part of the exercise is to define hidden attributive and associative entities. In the example above (see Figure 9-14), *Order* and *Customer* are fundamental entities. *Order-Item* is an attributive entity. If it had not already been identified, either normalization would identify it, or it would be identified by answering the question: Can any of the attributes relating to entity *Order* occur more than once? If the answer to this question is yes, there are attributive entities to be identified.

Direct normalization of ERDs is possible but requires detailed understanding of data. When you have an ERD but are less comfortable about your understanding of the data and their relationships, tabular normalization can be used to complement, validate, or replace direct normalization. Tabular normalization requires complete definition of data and relationships, and results in *exactly the same* entities as direct normalization.

To use tabular normalization, first describe each entity and all entity attributes. Cluster attributes depending on whether they are singular or multiple occurrences. (Tabular normalization rules are summarized in Table 9-1.) Then, proceed to remove repeating groups. For each repeating group create a new relation. The key of the new relation is the key of the repeating group and the original key. To remove partial key dependencies, create new relations of any attributes and the part of the key to which they relate. The key of the new relation is the part of the original primary key that functionally defines the relationship. Finally, remove nonkey dependencies by creating new relations from the nonkey attributes that are related. The key to the new relation is the attribute(s) that define the functional relationship. In the tabular method, multivalued dependencies are treated as single attribute, repeating groups in the nonnormalized set-up stage.

TABLE 9-1 Normalization Rules

For Unnormalized Data

1. Identify all attributes that relate to an entity. Keep in mind that there are several types of attributes.
   - Nonrepeating, primary key attributes(s). A nonrepeating attribute is a single fact about an entity type. A **primary key** is a unique identifier for all attributes associated with an entity type.
   - Nonrepeating, nonkey attributes(s) are single facts about an entity type.
   - **Repeating attribute**(s) are facts that may have more than one occurrence for a specific value of an entity's primary key. Repeating attributes may be single repeating facts, such as the date of birth of offspring; or may be groups of repeating facts, such as date of birth and name of offspring. Repeating attributes are either repeating key attributes or repeating nonkey attributes. Repeating nonkey attributes are listed with their primary key identifier.
2. List all attributes that relate to an entity together. Indent repeating information. Skip a line or leave a space between entities and between repeating groups. Repeating groups might have only one attribute that repeats; this is also called a multivalued dependency. Place an asterisk at the first attribute of each repeating group to show its beginning.
3. Underline the primary key field(s) of the unnormalized relations, including keys of both singular groups and repeating groups.
4. Proceed to first normal form.

First Normal Form (1NF)—The Goal of 1NF Is to Remove Repeating Groups

1.1. Examine each relation. If the relation has no repeating groups, it is in 1NF. Draw an arrow from the unnormalized column to the normalized column to show that the analysis is complete, and continue.

1.2. If the relation has repeating groups, build a relation from the single nonrepeating fields. The key of the relation is the key of the original relation. Continue.

1.3. Next, for each repeating group, build a new relation of the repeating information. Append the key of the original relation to the repeating information. The key of this relation is the key of the original relation *plus* the key of the repeating group.

Second Normal Form (2NF)—The Goal of 2NF Is to Remove Partial Key Dependencies

2.1. Examine each relation independently. If the 1NF relation does *not* have a compound key, it is in 2NF. Draw an arrow from the relation through the 2NF column to show that it is complete, and continue.

2.2 If the 1NF relation has a compound key for each nonkey field, ask the following question: Do the data field relate to the *whole key*? In other words, do you need to know the whole key to know the values of the attribute, or do you only need part of the key to know the value of the attribute? If the answer is that you need the whole key for all fields, the relation is in 2NF. Draw an arrow from the relation through the 2NF column to show that it is complete, and continue.

2.3 If by knowing a *part* of the key we know the value of one or more data fields, then we will build two new types of relations. First, build a relation with the nonkey data field(s) that are wholly dependent on the compound key. The key of this relation is the key of the 1NF relation.

2.4 Second, build one new relation for each partial key identified. The new relation(s) include the nonkey data field(s) and the part of the original key on which they are fully dependent.

(Continued on next page)

TABLE 9-1 Normalization Rules (*Continued*)

Third Normal Form (3NF)—The Goal of 3NF Is to Remove Nonkey Dependencies

3.1 If the 2NF relation(s) have only one nonkey data field, it is in 3NF, go to optimization.

3.2 If all data fields in the relation(s) are dependent upon the key and nothing but the key, then the relation is in 3NF. The question here, is "Do nonkey fields relate to the key or do they really relate to each other?"

3.3 If a nonkey dependency exists, build one relation of the nonkey data field(s) that are dependent on the 2NF key (this include the nonkey field that is the key in the step below).

3.4 Build one new relation for each nonkey dependency identified. The new relation(s) include the nonkey data field(s) and the nonkey field on which they are dependent. The key of this relation is the nonkey field from the original relation on which the other field(s) is dependent.

Now, check for anomalies . . . conditions that still will cause errors. This is one way of double-checking that your original relationships were correctly defined. Ask two questions.

1. Given a value for a key(s) of a 3NF relation, is there *just one* possible value for the data? If the answer is NO, then multivalued dependencies exist. Check that the correct data relationships are defined, then treat the multivalued single fact as a single-attribute repeating group and renormalize the data.
2. All are attributes directly dependent upon their related key(s)? If the answer is NO, then transitive dependencies exist. Treat the transitive dependency like a nonkey dependency and renormalize the data.

Finally, synthesize and integrate the relations.

1. Remove any fields that are computed in the application. This does not mean that these attributes are not stored in the physical database; it means that they are not logically required to define the entity.
2. If two or more relations have *exactly the same primary key*, combine them into one relation. Make sure that each attribute occurs only once.

At third normal form, synthesis of the resulting relations is performed to

- combine relations that have identical primary keys but different nonkey attributes
- eliminate relations which are exact duplicates, or proper subsets, of other relations
- combine relations for which the primary key of one is a proper subset of the primary key of another

After normalization and synthesis are complete, new entities (or relations) and their relationship to the fundamental entities are added to the diagram as needed to fully depict the information.

Next, the entities and relationships are analyzed to determine if a class structure is needed. The reasoning process is as follows:

1. Ask if this entity occurs in this, and only this, form (i.e., with all attributes) for *every* legal occurrence of the relationship being examined? If the answer is yes, continue. If the answer is no, you must define subclasses that describe the contingencies of existence for

the entity. This procedure is described in the next section.

2. Does this relationship hold for all occurrences of the entity? If yes, continue. If no, follow the reasoning below to determine the subclasses of the entity and their relationships.
3. Is this entity ever optional? If no, continue. If yes, follow the reasoning below to determine the subclasses of the entity and their relationships.
4. Can only a *subset* of occurrences of an entity participate in a given relationship? If no, continue. If yes, follow the reasoning below to determine the subclasses of the entity and their relationships.
5. Have several types, or kinds, or categories of an entity been identified? If no, continue. If yes, follow the reasoning below to determine the subclasses of the entity and their relationships.
6. Are words like "either, or, sometimes, usually, generally, in certain cases" ever used in describing entity behavior? If no, continue. If yes, follow the reasoning below to determine the subclasses of the entity and their relationships.

To determine subclasses, you must determine which information is kept (or which processing is done) for which type (or subclass) of orders. Ask questions about every possible variation of information and processing until 'if-then-else' logic surfaces. Use the alternative situations to define the subclasses. That is, one subclass for the *if* logic, another subclass for every other *else if* logic. Ask questions of each type of information about every entity. Don't stop just because you find one subclass; there may be others. When you have found all subclasses, verify them with the user and modify the diagram accordingly.

For instance, in an order fulfillment application, a legal entity relationship describes 'customers place orders' (see Figure 9-16), but that information may not be the same for all customers' orders. Does the time of day or time of month affect the relationship? Do the shipping address differences affect the relationship? Does the sold-to/ship-to arrangement affect the relationship? Does the type of goods ordered affect the relationship? Does the type of payment affect the relationship? In this example, we will say that *Cash Order* information kept includes *Customer ID* and *Total Amount*, where *Credit Order* information kept includes *Customer ID, Name, Sold-to/Ship-to addresses, Order Date, Shipping Terms,*

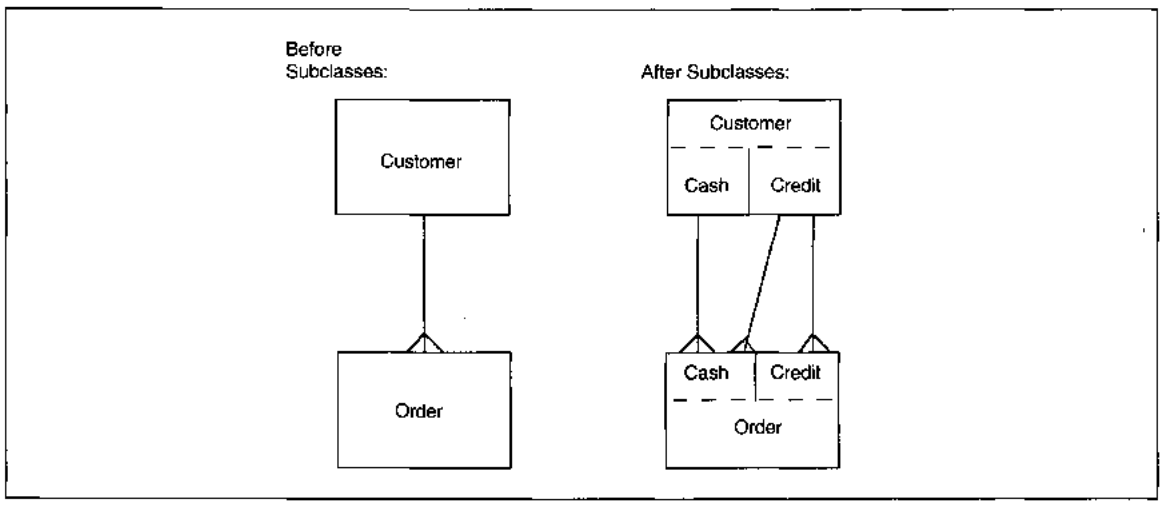


FIGURE 9-16 Examples of Subclasses in Customer-Order Relationship

for each item (*Item Number, Item Description, Quantity, Price, Extended Price*), *Sales Tax*, and *Total*. Here, we know there are subclasses because different sets of data are kept. The next issue is to decide the entity(s) to which the subclasses relate.

To decide which subclasses apply to orders, we ask if the entity *Order* is affected differently by cash and credit sales. If the answer is yes, different information is kept for each. In this example, there would be subclasses for *Cash Order* and *Credit Order*. Then, we ask if the entity *Customer* is affected differently for cash and credit customers. Are all customers either cash or credit? What are the rules for buying on credit? The common answer is applied here. Some customers are only cash, thus, creating a cash customer subclass. Some customers are qualified to buy on credit, but they are not *required* to buy on credit. That is, credit customers can pay either by cash or by credit. Therefore, knowing which type of order a customer will create is only possible if the customer is a cash customer. Depending on the application, these customer subclasses might be important. Here, we will say they are. The ERD is altered to show the subclasses of each entity class and how they now relate to each other. Notice that the simple before diagram in Figure 9-16 is more complex with subclass additions.

To summarize, first define entities, then relationships, then attributes. Promote the many-to-many relationships to associative entity status and modify the diagram to reflect the new entities. Add attributive entities as required for repeating information relating to entities. Identify all new attributes of all entities. If necessary, do tabular normalization of the relations. Analyze each entity to determine if subclasses are required and modify the diagram to describe them. These activities are best documented in a CASE tool with repository (or dictionary or encyclopedia) entries made as the work progresses. At the end of ERD creation, you have not only the ERD, but also the repository definitions for all items in the ERD.

ABC Video Example Entity-Relationship Diagram

The first step (refer to the list on p. 339) in developing the ERD is to identify fundamental entities. A first-cut definition of the potential fundamental entities in ABC rental processing includes: customer, video, rental, printed rental, clerk, and system. These entities are identified from the ABC Rental Processing requirements in Chapter 2. Next we analyze each potential entity to see if it really is in the business area and application.

Customer is a noun. It uniquely defines the people who rent and return videos. By itself, customer does not take on a value; rather, each customer is described by a set of attributes. The business must keep information about customers renting videos to do business with them. The formal name is *Customer*.

Video is a noun. It uniquely defines an item from inventory that is available for rent. By itself, video does not take on a value; it has descriptive attributes. The business must keep information about videos to conduct its business. The formal name is *Video*.

Rental is a noun that uniquely describes videos rented by customers for a specific period. By itself, *rental* does not take on a value; it combines attributes of *Customer* and *Video* with attributes of its own. The business must keep information about rentals to provide an audit trail for tax purposes. The formal name is *Rental*.

A Printed Rental is a noun that describes videos rented by customers for a specific time period. A printed rental is not unique since its definition mirrors that of rental. However, it is unique in that it shows the customer signature. If there is a legal dispute over charges, the business is required legally to provide documentation that rental took place, and the customer knowingly rented. The business does not keep information *about* a printed rental, though; the information is about a rental.[5] Printed rentals are another medium or form of *Rental*. Printed Rental is stricken from the list.

Clerk is a noun uniquely describing the person who initiates processing for the application. By itself, clerk does not take on a value. The business does not need to know who did the entry of information unless Vic changes the requirements of the

5 If customer signature was kept, or if we just left printed rental on the list, when the data were normalized we would find that the primary keys to the printed rental and the rental were identical. That would lead us to combine the fields in one relation called *Rental*.

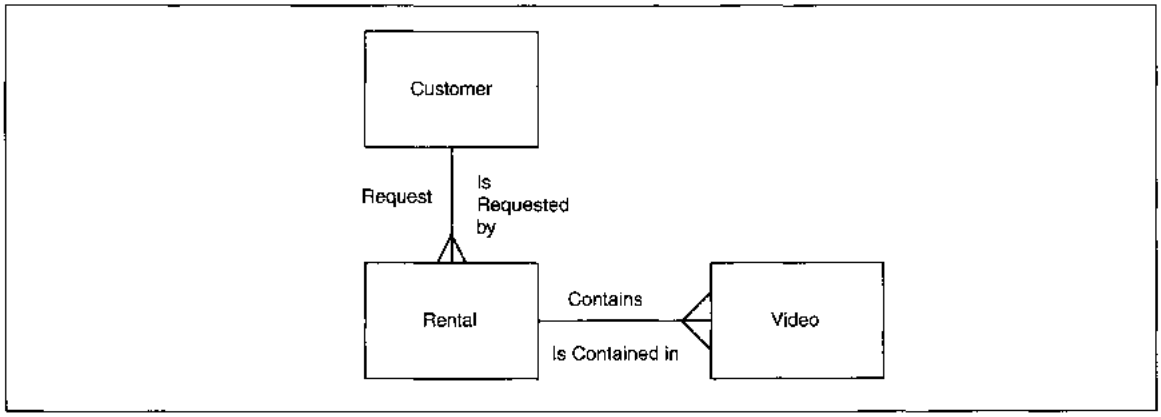


FIGURE 9-17 ABC Rental Processing—First-Cut Entity-Relationship Diagram

application. Since clerk is not required, we strike it from the list.

Similarly, system is a noun that uniquely describes the hardware/software environment that will do rental processing. The system has no personal values, and neither do we maintain information about the system. System is stricken from the list.

Next we draw a rectangle for each of the three entities that remain: *Customer*, *Video*, and *Rental*. Figure 9-17 shows the entities and relationship(s) between the fundamental entities. Customers *request* Rentals. Rentals *contain* Videos. The relationship names are unique verbs describing the interactions. The line connecting *Customer* and *Rental* contains crows' feet at the *Rental* side to show a one-to-many relationship. That is, each *Customer* may place one or more *Rentals*. Each *Rental* is placed by one and only one *Customer*.

Similarly, each *Rental* contains one or more *Videos*; each *Video* can be rented by one and only one *Rental* at a time. We have a problem with the clause *at a time* in this definition. Relationships are supposed to be defined without regard to time. How do we account for this problem? We might defer a decision on how to deal with this until some later time, making a note of the need for 'date' as an identifier for the video-rental relationship. Or, we might remove time from the definition, creating the ERD in Figure 9-18 which shows a many-to-many relationship with rental and video. That is, each video may

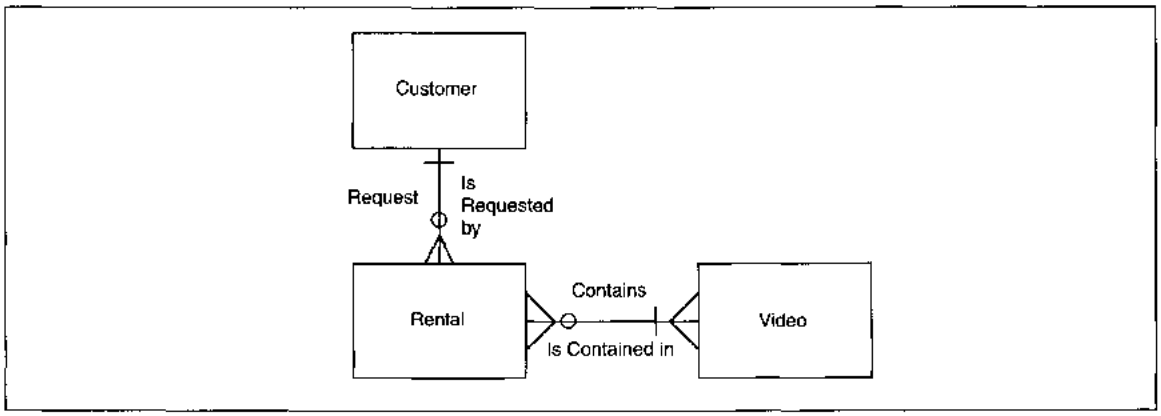


FIGURE 9-18 ABC Rental Processing—Second-Cut Entity-Relationship Diagram

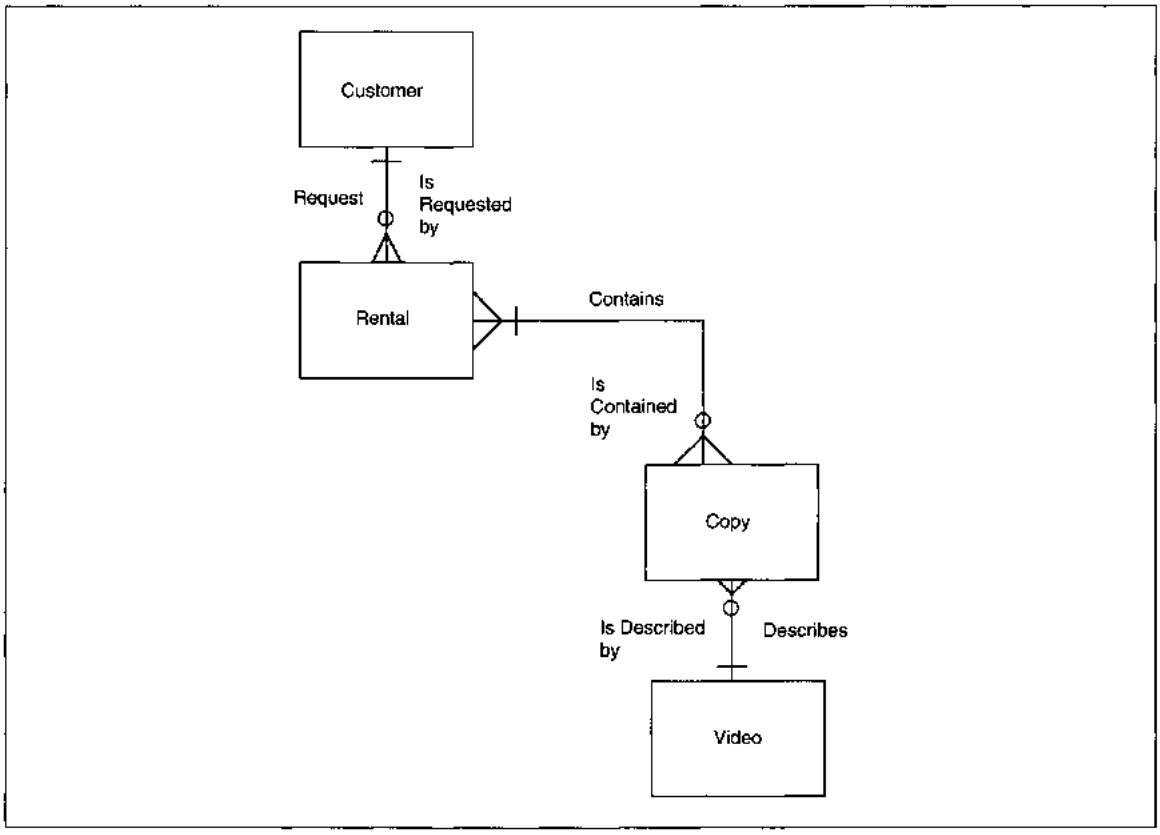


FIGURE 9-19 ABC Rental Processing—Third-Cut Entity-Relationship Diagram

be rented more than once, and each rental may contain more than one video. We take this option at the moment, knowing that it is an incomplete definition of the relationships which need to be refined.

Next we decide the nature of the relationships, whether they are required or optional. A *Customer* must exist to place *Rentals*. A *Video* must exist to be contained in a *Rental*. Does this make sense; must you have both a *Customer* and a *Video* to do a *Rental*? Yes, this makes sense. Now analyze the other side of the relationships. Are *Rentals* required for *Customers* to exist? No, *Customers* do not necessarily have rentals every day. Are *Rentals* required for a *Video* to exist? No, *Videos* can exist without being related to a *Rental*. Both relationships of *Rental* to the other entities are optional.

Identify attributes and associative entities. The *m:n* relationship of *Videos* and *Rentals* should be promoted to make an associative entity. The new entity relates to each physical tape being rented. Thus, we have a *Video* entity and a *Copy* entity. We reason through this creation in another way. Video information is not detailed enough to keep track of every physical tape in inventory because each video may have many copies. This leads us to add information about copies. Referring to the case in Chapter 2, we find that Vic wants to be able to track the status of any tape. The minimum copy information needed is *Video ID*, *Copy ID*, *Date Received*, and *Status*. Other information might be considered, for instance, current month rental counts, but we defer this for the moment. Figure 9-19 shows the ERD to this point.

Does the insertion of *Copy* take care of the many-to-many relationship? We can still have a *Copy* on many rentals over time and *Rentals* can contain many items, so the answer is no. Next we look at the Rental to further examine its details. Rentals are similar to orders. Just as an order has one or more items, each rental can have one or more rental items. There is a one-to-many relationship of *Rental* to *Rental-Item* which we add to the diagram. By itself, this does simplify the many-to-many relationship; a *Rental-Item* belongs to a specific *Rental* and relates to a specific inventory *Copy*. Now the entities and relationships look clean with all many-to-many relationships promoted, and all apparent one-to-many relationships explained (see Figure 9-20).

To confirm the original and promoted entities, we will normalize the data using the tabular method. For tabular normalization to proceed, first define the attributes of each entity. From the functional requirements in Chapter 2, list all attributes of each entity. The list is shown in Table 9-2, in unnormalized form, with the copy information identified as repeating within video information. Make a separate list for each entity. For each entity, list together attributes that occur only once. Indent repeating groups under the related entity, making sure that all information for each *group* is together. Underline primary keys for *both* nonrepeating and repeating information. Remember, the primary key *uniquely* identifies its information.

Next apply the rules in Table 9-1 to remove repeating groups, partial key dependencies, and nonkey dependencies. Synthesize the 3NF results to ensure minimal redundancy. The result of ABC's

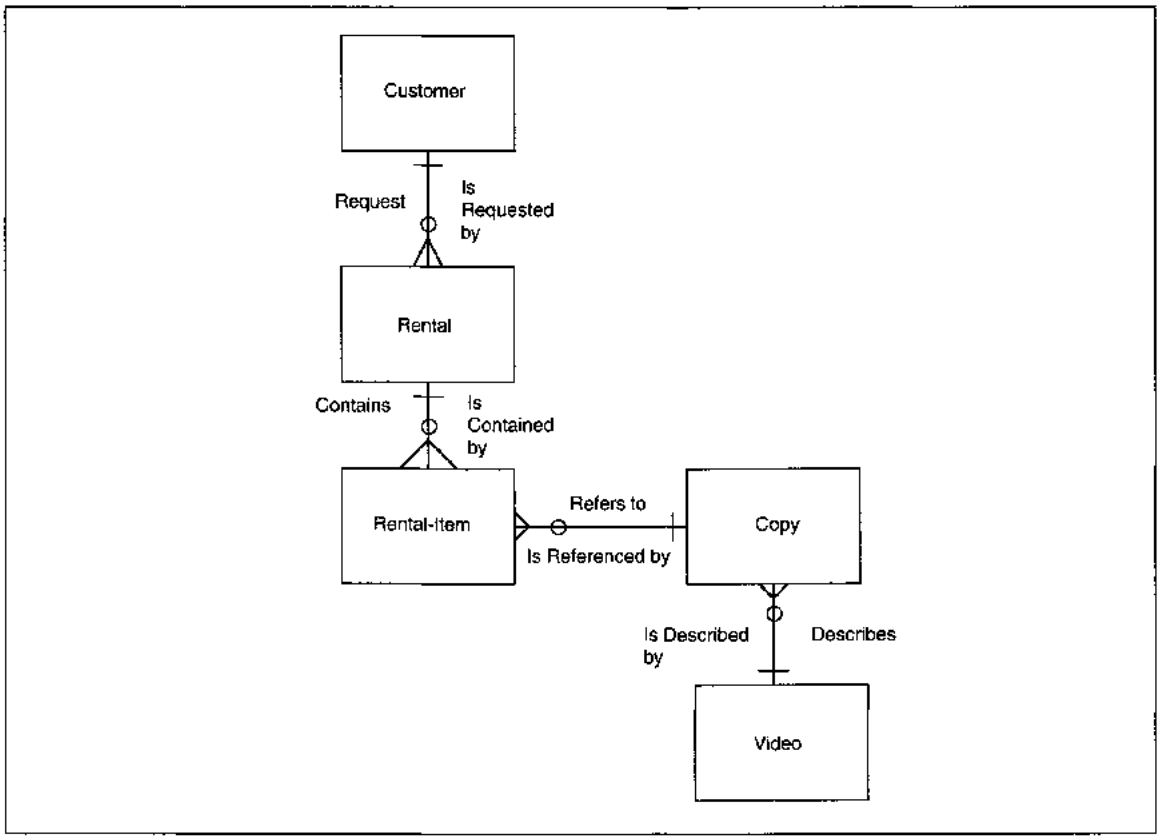


FIGURE 9-20 ABC Rental Processing Fourth-Cut Entity-Relationship Diagram

TABLE 9-2 List of ABC Video Entity Attributes

Unnormalized Form—Repeating Groups, All Primary Keys Identified	First Normal Form (1NF)—Repeating Groups	Second Normal Form (2NF)—Partial Key Dependencies	Third Normal Form (3NF)—Nonkey Dependencies	Delete
Customer Phone Customer Name Customer Address Customer City Customer State Customer Zip Customer Credit Card Number Credit Card Type Credit Card Expiration Date				
Customer Phone All Customer Info from Above Rental Date Total Rental Fees *Video ID Copy ID Video Name Rental Date Return Date Rental Rate Late Fee Due Fees Due				
Video ID Video Name Entry Date Rental Rate Copy ID Date Received Status				

normalization is shown in Table 9-3. Now we have six relations to be synthesized and evaluated.

In the synthesis step, several pieces of information are deleted. 'All Customer information' is not required to maintain rental information; only a *Customer Phone* or *ID* is required as a cross reference, or foreign key, to the *Customer* relation. *Total Rental Fees* are calculated and, therefore, not required. The entire relation containing *Video ID*, *Video Name*, and *Rental Rate* is deleted because it *exactly* duplicates information already in the next relation which has more attributes.

To reconcile the 3NF results to the ERD, we look at the *Rental* relationships again, and use some 'out of the box' thinking. The relationship we identified for *Rental* to *Rental-Item* is similar for many busi-

TABLE 9-3 ABC Video Normalization Results

Unnormalized Form—Repeating Groups, All Primary Keys Identified	First Normal Form (1NF)—Repeating Groups	Second Normal Form (2NF)—Partial Key Dependencies	Third Normal Form (3NF)—Nonkey Dependencies	D e l e
Customer Phone Customer Name Customer Address Customer City Customer State Customer Zip Customer Credit Card Number Credit Card Type Credit Card Expiration Date	→	→	→	
Customer Phone All Customer Info from Above Rental Date Total Rental Fees *Video ID Copy ID Video Name Rental Date Return Date Rental Rate Late Fee Due Fees Due	Customer Phone All Customer Info from Above Rental Date Total Rental Fees Customer Phone *Video ID* *Copy ID* Video Name Rental Date Return Date Rental Rate Late Fee Due Fees Due	→ *Customer Phone* *Video ID* *Copy ID* Rental Date Return Date Late Fee Due Fees Due Video ID Video Name Rental Date	→ →	X X
Video ID Video Name Entry Date Rental Rate	Video ID Video Name Entry Date Rental Rate	→	→	
Copy ID Date Received Status	Video ID Copy ID Date Received Status	→	→	

ness transactions: orders, confirmations, shipping papers, back-orders, and invoices. The question here is: Do we *need* both entities? We require the *Rental-Item* entity information because it documents the business transaction. The question then is: Do we need the *Rental* information separated? Is it uniquely different? *Customer Phone* is also in *Rental-Item*; *Rental Date* is also related to each video the

Customer: xxxxxxxxxxxxxxxxxx, xxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxx, xx xxxxx
(xxx) xxx-xxxx

Open Rentals:

Video	Copy	Description	Rental	Pd	Late	Pd	Other	Pd
xxxxx	xx	xxxxxxxxxxxxxxxxxxxxxxxxx	9.99	x	9.99	x	999.99	x
xxxxx	xx	xxxxxxxxxxxxxxxxxxxxxxxxx	9.99	x	9.99	x	999.99	x
xxxxx	xx	xxxxxxxxxxxxxxxxxxxxxxxxx	9.99	x	9.99	x	999.99	x
xxxxx	xx	xxxxxxxxxxxxxxxxxxxxxxxxx	9.99	x	9.99	x	999.99	x

New Rentals

xxxxx	xx	xxxxxxxxxxxxxxxxxxxxxxxxx	9.99	x				
xxxxx	xx	xxxxxxxxxxxxxxxxxxxxxxxxx	9.99	x				
xxxxx	xx	xxxxxxxxxxxxxxxxxxxxxxxxx	9.99	x				
xxxxx	xx	xxxxxxxxxxxxxxxxxxxxxxxxx	9.99	x				
xxxxx	xx	xxxxxxxxxxxxxxxxxxxxxxxxx	9.99	x				

Total	99.99	99.99	9,999.99
Total Amount Due			9,999.99
Amount Paid			9,999.99
Balance			9,999.99

FIGURE 9-21 Partial Rental Screen

customer has. We eliminate *Total Rental Fees* as a computed field but we need to redecide if we will ever need this information stored in a file.

Continuing this reasoning, think of the processing to be done. When a customer requests a video, the system should display all open rentals regardless of when they were rented. The *Rental-Item* information will be listed down the screen in rows, one row per video (see Figure 9-21).[6] A total of all open rental fees plus any new fees will be near the bottom. From where will customer information come? If we *keep* the *Rental* relation/entity, we either *choose one* from potentially several for display, or, if Customer Phone is entered first, we *ignore them all*. This sounds like a kludge, that is, a mess! If we delete this entity/relation, can we get the information another way? That is, can we recreate this relationship if we need to for any reason? The answer *in this case* is, yes. *Rental-Items* all have *Customer Phone* and the first one accessed can be used to retrieve customer information. We conclude that we can eliminate the 'Rental' entity entirely. The completed, revised ERD is shown as Figure 9-22. *Rental-Item* is renamed to *Open Rental* to avoid confusion about its contents.

After removing the *Rental* entity, the relationships and entities now appear minimal. That is, we must keep all of these entities. If we remove any one of these entities, we *cannot* recreate the desired information for the removed entity, nor can we completely describe all data relationships. With these entities, we can represent the entire problem data

6 This is another example of the jumping between levels of detail required to complete each logical step in the process.

space, and we can accommodate all the processing required in the application. The ERD now appears complete. Keep in mind that complete does not mean cast in concrete; the ERD can be modified as required to accommodate new information.

The reasoning process we used to eliminate the rental entity can be used on any similar entities, for instance, orders. In other applications the higher level entity analogous to the *Rental* entity here might be required. You *cannot* eliminate an entity when any of the following conditions is true:

- The entity has unique information of its own.
- The entity, or its attributes, cannot be recreated through combining other entities.
- The entity is required for legal purposes.

An accurate and complete ERD is crucial to developing an application that solves a real-world problem. During development of the ERD, pay particular attention to entity definitions, making sure they are distinct, simple, and precise. Analyze each entity for selectivity in processing, data, or timing to determine if a class structure is warranted. Also analyze each entity for its actual need. If an entity can be recreated from other information, has no unique attributes of its own, and is not required for legal purposes, omit it. Analyze every possible relationship to determine relationship existence. When defining relationship cardinality and required/optional status, make sure time and current procedures are ignored. Do pay attention to legal requirements and business requirements in defining cardinality and required status.

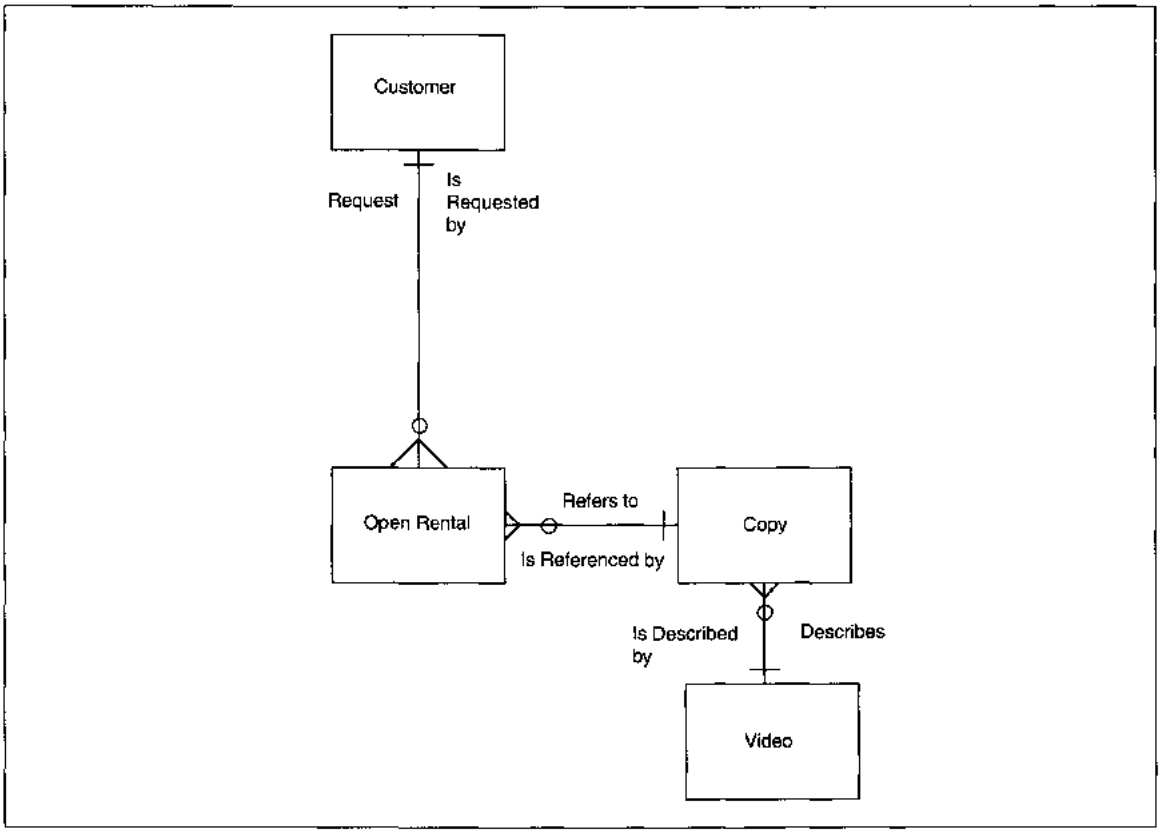


FIGURE 9-22 ABC Revised, Complete Entity-Relationship Diagram

Decompose Business Functions

Rules for Decomposing Business Functions

If a functional decomposition was not yet developed at the enterprise level, it is created now. If a decomposition was developed at the organization level, it is further decomposed here to define details of processes. In either case, decomposition is independent of the ERD; it can be done before, during, or after the ERD. IE recommends the ERD first, but while you gather data for the ERD, you invariably get process information. Many practitioners concentrate on data first, but begin to build the decomposition simultaneously. The steps to functional decomposition are:

1. Define the enterprise for which the diagram is being developed. Place the enterprise name in a rounded rectangle at the top of the diagram.
2. Define business functions of the enterprise. Using consistent parts of speech for each name, place the functions in rounded rectangles on the second row of the diagram. Do not pay attention to current organization, policy, or procedures in defining functions. Use current business practices in the industry to guide the definition and placement of functions (and activities and processes).
3. Define the activities that fully define each function. Name them using consistent parts of speech, usually of the form verb-noun. For each function, create a separate diagram with a row depicting the activities of the function.
4. For each activity, fully define the processes that describe work performed for each activity. Name each process using the form verb-noun. Add processes under their respective activities on the diagram in the sequence in which they are performed.
5. Continue to decompose the processes and add them to diagrams depicting successive levels of detail until the definitions are atomic.
6. Verify all diagrams with the user.
7. Define the detailed procedures for accomplishing each process and document functions, activities, processes, and procedures in the repository.

First, identify (or verify) the functions applicable to the BAA activity. An easy way to check functions is to review the list of generic functions on p. 332 and, for each, determine its applicability to the situation. Name each function so it relates to the business context of the BAA. For example, if the function deals with finance, but in the client context finance includes both Finance and Accounting, use the latter function name. Name each function with a noun, preferably a nonqualified noun. For example, Finance is preferred to *Corporate* Finance. If users have not participated in this activity, verify the list with a user. Place each function on the decomposition under the enterprise identifier in rounded rectangles (see Figure 9-23).

Next, for each function, define the major activities and place them under the function they describe. The diagram resembles an organization chart. When complete, the activities should *fully describe* each function. *Do not* pay attention to organizational boundaries or current organization policies and procedures. *Do* pay attention to legally required actions, actions that specifically relate to goals of the organization, and industry practices that are required. *Do* identify timing, cardinality, or current business practices for each activity.

Activity names do not have a specific form, but should be consistent in the part of speech used for all names. In the example above, the function *Finance* might include several activities, such as Corporate Finance, Regional (or *Subsidiary-name*) Finance, International Finance, Analysis and Reporting, Planning, Budgeting, and Funds Management. In this example, Funds Management might have been called Manage Funds, but the inconsistent part of speech makes this a weak name.

Next, for each activity, decompose the activity to define the processes that *fully describe* the activity. Processes may have their own subprocesses. Continue decomposing until the elementary, or atomic level, of process is identified. Recall that an elementary process is the smallest *unit of work* users can

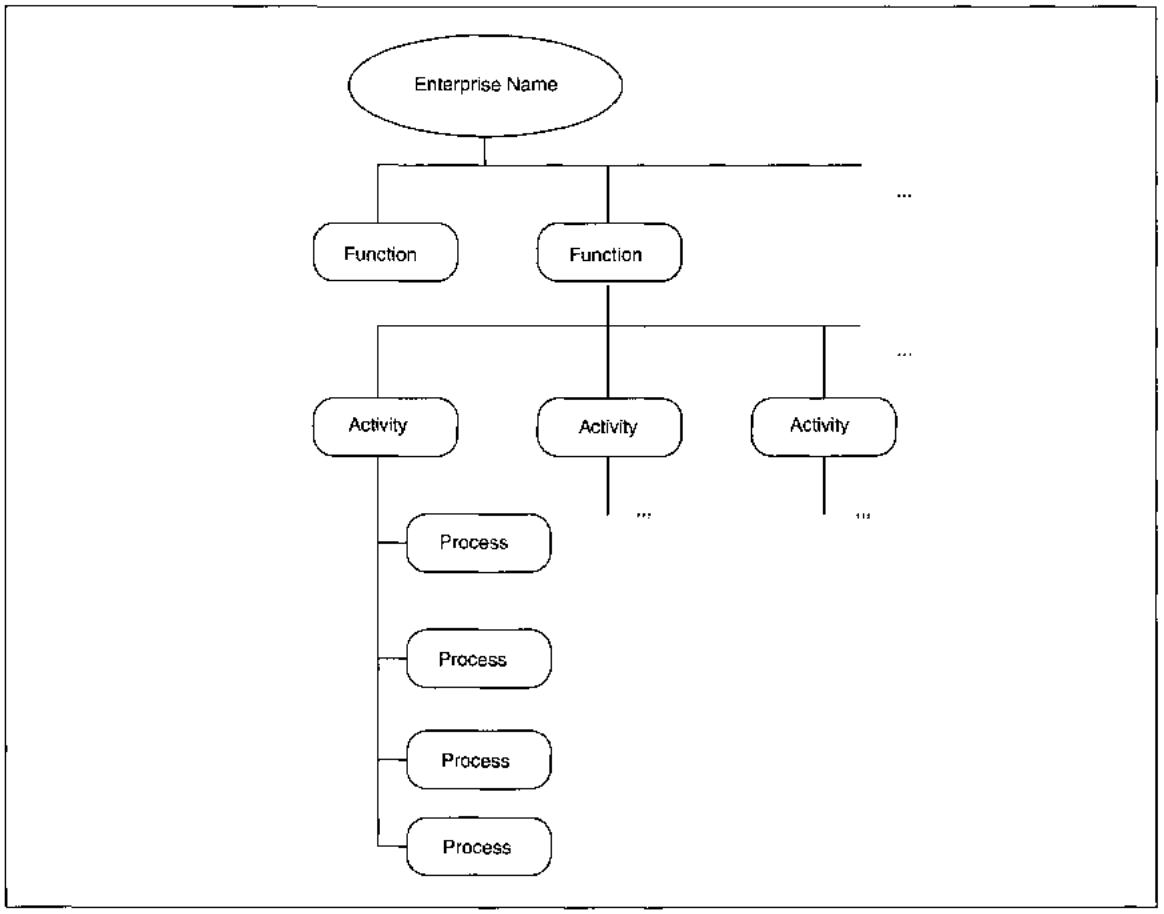


FIGURE 9-23 Placement of Functions, Activities, and Processes on a Functional Decomposition Diagram

identify. Name each process with a verb-object name. Within Funds Management, for example, processes might include: Manage Overnight Funds, Manage Cash, Manage Payroll Accounts, and Manage Savings Accounts. Each of these processes can be further decomposed to identify the details of the procedure used to perform this process. Continuing with this example, Manage Overnight Funds might include: Identify Funds, Identify Options, Analyze Options, Place Funds, Complete Accounting Entries. Each of these is a process, too, but these processes cannot be further decomposed without requiring interrogation of multiple processes to locate all of its component parts. Therefore, these processes are atomic, or elementary. That is, each can be performed as a unit, but cannot be further decomposed without losing its unit identity.

The difficulties of process decomposition lie in achieving parallel levels of abstraction and completeness. The goal is to maintain consistency within a level of process decomposition. The SE and user *must* work together during this definition because the levels of detail are beyond IS knowledge. Only job incumbents know exactly what they do and how they do it. The user is the main person defining the decomposition, but the SE is the person who actually

abstracts the diagram and systems information from the user-supplied information. The user relates each process to all of the other processes, describing each in detail.

Some clues to consistency of abstraction are amount of work, user comfort, same type of inputs and outputs, and timing. If all processes appear to do similar amounts of work, they are probably at a similar level of abstraction. If the user feels comfortable that the information is similar, it probably is.

If the processes have similar types of inputs and outputs, that is, they have no error processing and no exception processing *at the same level*, then they are probably at a comparable level of abstraction. Similarly, if one process has error and exception processing, the others also should have error and exception processing *at the same level.*

For concurrent processes, each process must be performed completely independently of all other concurrent processes. If concurrent processes are independent, then the abstraction level is probably okay. If concurrent processes have dependencies, then determine the relationship between the processes. Either the dependent process is, in fact, a subprocess, or the processes are not concurrent.

During process identification and definition, mark the diagram for processes that are used in more than one place. This identifies both potential reusable processes for the design activity and possible job consolidation for organizational analysis. Make sure that the names assigned to reusable processes are exactly the same and actually perform the same work.

The larger the organization, the more likely you will need more than one level of process decomposition to describe fully the processes of each activity. Continue to decompose levels of subprocesses until you reach processes that can no longer be described as performing some *whole action.*

ABC Video Example Process Decomposition

To begin, we ask ourselves what are the functions of ABC that relate to this BAA. The functions of ABC are Purchasing, Rental Processing, Accounting and Personnel/Payroll as shown in Figure 9-6. This application is concerned only with Rental Processing, so we decompose only the Rental Processing function.

First, we define the activities of Rental Processing and place them on the diagram in rounded rectangles. Return to the case in Chapter 2 and outline the major activities. If you have difficulty finding activities, look at the entities and define the actions taken for each entity. Obvious activities relate to customer and video maintenance and actual rent/return processing. Can you identify any others? If not, add these to the diagram and decompose them. Activity identification is not a one-time activity; it is ongoing and other activities might become obvious as you work through the processes. Keep in mind that when you identify activities with a user, it is from their experience and not from written text, so it is somewhat more direct.

Both maintenance activities are decomposed into create, read, update, and delete processes (CRUD). Notice that the activity names are of the form *verb-object*. The resulting additions to the decomposition are shown in Figure 9-24.

Next, we must decide if rent and return are one activity or two. This is the same issue we dealt with in *process design* (in Chapter 8); here we will have slightly different results because the reasoning process is different. The questions here are: Can we define rental without reference to return? Can we also define return without reference to rental? And, does this completely define rent/return processing? The first two answers are yes, the third is no. Both rentals and returns must accommodate the other process as a subprocess for completeness. Therefore, rent and return processing must be combined as one activity.

An easy way to decompose these processes and be reasonably sure we are complete and correct is to decompose the four options separately. The options are rent without return, return without rent, rent with return, and return with rent. A table listing the four options and their subprocesses is shown as Table 9-4. Several issues can be identified for discussion. First, is *Check for Late Fees* the same level of abstraction as the other processes? Second, is *Print Receipt* the same type of process and does it belong on the table? Third, does this look complete? For

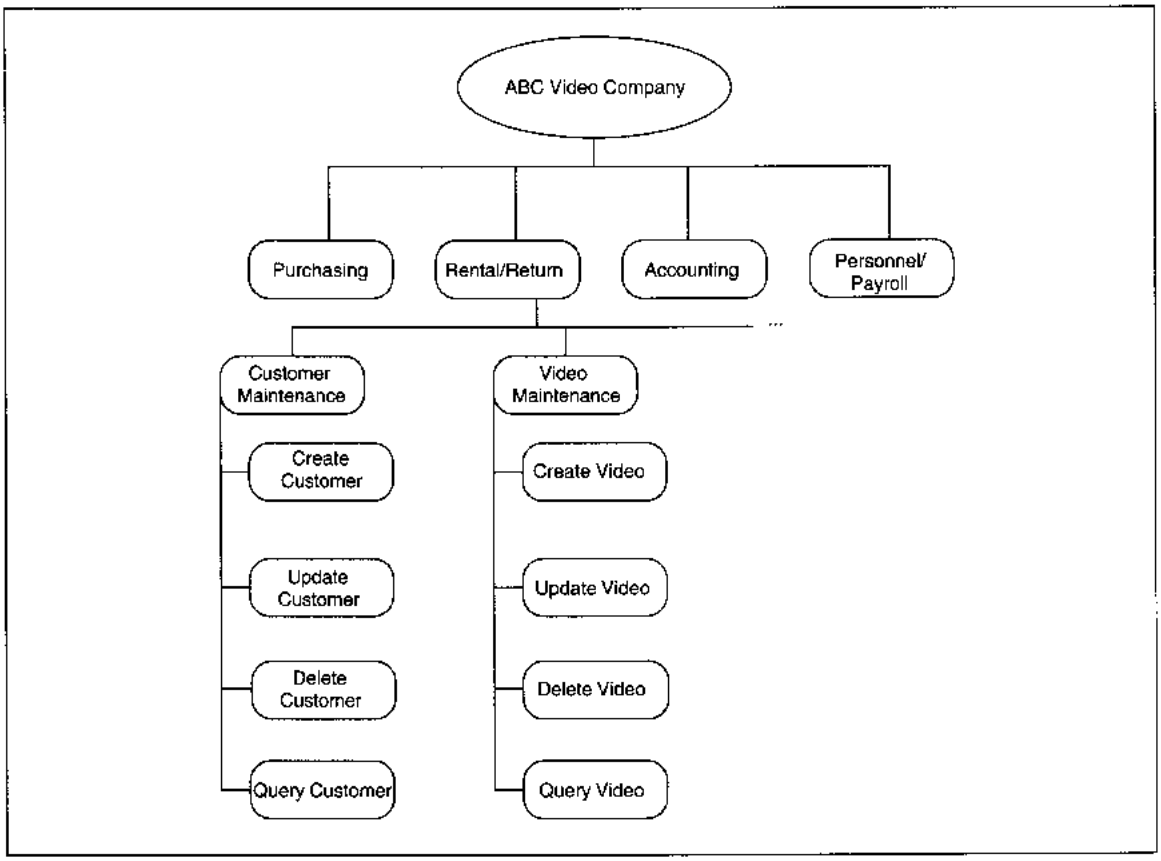


FIGURE 9-24 Decomposition for Customer and Video Maintenance

instance, where are *Create Customer* and *Create Video* when the items are not found in a database? Last, can we consolidate these four lists to develop one list for the decomposition diagram?

First, *Check for Late Fees* appears to be at a lower level of detail than the other processes. To check this, walk-through the process. To check for late fees, data from an open rental must be in memory.

> If the *Return Date* is not equal to zero, subtract *Rental Date* from *Return Date* to get *Number Of Days Rented.*
>
> If *Number Of Days Rented* is greater than the allowed amount (here we use two), multiply (*Number Of Days Rented* – 2) by $2.00 (the late charge) to get *Late Fees.*
>
> If *Late Fees* are greater than zero, display *Late Fees* and add *Late Fees* to *Total Amount Due.*

This is all logic; there is no reading or writing to files. Thus, this is a simple process that borders on being too small to be called a process. This logic could be included in another process if and only if the other process has the same execution pattern for each pass of the logic. This means we next look at how often Check for Late Fees is executed. Check for Late Fees is in every list, but is it executed for every rental and return? The answer is that for all open rentals, this process would execute to check for fees owed whether there are current returns or not. Also, for all returns, after the return date is added, the process Check for Late Fees should be executed.

Next we review the logic to see if exactly the same procedure is followed in both cases. The answer to this issue depends on *when* late fees are considered. So far, we have talked about late fees for

TABLE 9-4 Decomposition of Rental Options

Rental Without Return	Return Without Rental	Rental With Return	Return With Rental
Get Customer ID	Get Return Video IDs	Get Customer ID	Get Return Video IDs
Get Valid Customer	Get Open Rentals	Get Valid Customer	Get Open Rentals
Get Open Rentals	Get Valid Customer	Get Open Rentals	Get Valid Customer
Check for Late Fees	Add Return Date	Get Return Video IDs	Add Return Date
Get Valid Videos	Check Late Fees	Add Return Date	Check Late Fees
Process Payment and Make Change	Update Open Rentals	Check for Late Fees	Get Valid Videos
Create Open Rental	Update/Create History	Get Valid Videos	Process Payment and Make Change
Print Receipt	Process Payment and Make Change	Process Payment and Make Change	Create Open Rental
	Print Receipt	Create Open Rental	Update Open Rental
		Update Open Rental	Update/Create History
		Update/Create History	Print Receipt
		Print Receipt	

tapes with return dates only. You may be tempted to charge fees every day as they accrue, whether the tape is returned or not. If you do this, you need *very* complex logic to identify what fees are accrued, what fees are paid, and what fees are still owed. Complex logic is frequently wrong and is always error prone. If possible, use the KISS (Keep It Simple, Stupid) method and charge fees only when a return date is present. To continue this thinking, what rental attributes do we need to deal with late fees? Do we need a late fees field? A flag when late fees have been paid? The case does not tell us what Vic wants; so we need to talk to him about this.

In this case, Vic and the accountant decide that, for accounting purposes, they want to know all charges applied to a rental. Information to be kept includes: regular fees, regular fees payment, late fees, late fee payment, any extraordinary fees, and extraordinary fee payment. Notice they do not care about payment *dates*. We have two choices for dealing with late fee data. First, we can compute fees and add them to the file when paid or second, keep two sets of fields, one for the fee and a flag for fee payment. The data and processing for the first option are simpler, but this now makes the processes creating and updating open rentals dependent on successful *Process Payment and Make Change*. This is not only an acceptable tradeoff, but a better business practice since we do not want to update with unsuccessful payment processing. We note the new attributes and add them to the repository.

The second issue deals with *Print Receipt*. Is *Print Receipt* the same type of process and does it belong on the table? The printed rental orders could be considered an output data flow of *Process Payment and Make Change* rather than requiring its own process. Since ABC defines printing of orders as a separate process required of the application, we could leave it on the list. Unfortunately, the methodology does not give guidance in the issue of whether to include or omit data printing processes. In general, if the printing is incidental to another process, that is, it is a record of the processing, then it is not separate. A print process should be distinct if it fulfills legal

obligations, or is independent of all other processes, or is contingent on other processing. On the job, the SE, with the analysis team, decides which method of defining inputs and outputs will be used, then is consistent in their definitions. A related issue is the relationship of *Print Receipt* to the other processes. Does it follow payment processing, does it follow and confirm file creation and updating, or is it independent? At the moment, printing appears related to payment processing only, but here again is something we need to ask Vic. A similar problem arises with data entry procedures that we will discuss later.

Here is a sample dialogue between Mary and Vic to resolve the relationship issue.

Mary: "We are trying to decide about when to print receipts and how receipts relate to the rest of the process. Can you tell me the legal requirements and if you have any other business requirements?"

Vic: "Hm, now, we write down all the customer numbers, video numbers, amount paid, and reasons for each transaction. We don't really give customers a receipt in the manual system. I'm not sure what the legal requirements are; I'll get the accountant in here, too."

The accountant comes in and is asked the same question. She says, "It would be nice to have a paper copy of each transaction in which money is processed so I can locate errors when I do the bookkeeping. Trying to find an error by querying the computer might be longer than just adding up the days' receipts in different categories. It would also provide IRS documentation if you don't plan to do that on the computer. Do you?"

This leads to a discussion of the tax processing possible and the potential costs to the project, which are negligible at this stage. The final decision is to require receipts not only for transactions in which money is processed, but to offer a receipt as an option to the customer for nonmoney transactions.

The discussion then digresses into the issue of how long records must be kept on the rental file. If all money-related transactions are printed, records of paid transactions could be deleted. Vic wants access to transaction data for historical analysis but thinks the history files will answer most of his questions. In his manual system, Vic purges the files once a year at tax time, but he says there is too much paper to look at any paper records unless a customer actually disputes a charge. In any case, Vic, the accountant, and Mary jointly decide to purge the transaction files monthly and move deleted records to an off-line archive file. This discussion causes a new activity to be added to the decomposition under *Periodic Processing*.

Next, Mary broaches the subject of keeping track of file updates and printing the receipt only when the file updates (or creates) are successful. Vic has two concerns. He needs the ability to fix a file problem if one occurs, and he wants the ability independently of the *rental* process. Second, he is leery about using the receipt as notification of a problem. "If users think there is a problem with the computer system, they might not trust the information we give them about late fees and other charges." Vic decides that printing is independent of file updates and that an operator message should be displayed for errors in writing to files.

The third issue is to evaluate the completeness of the processing defined. In particular, where are *Create Customer* and *Create Video* when the 'valid' items are not found in a database? From a simple evaluation of process names, the processing appears complete. To resolve the issue about the two create processes, we look specifically at those processes. Again, there are two options for dealing with the need to create customers and videos: It can be a separate process or it can be a subprocess to the associated *Get Valid . . .* process. The question to answer is: How important, in the rent/return activity, are create customer and video? The answer is that they are not very important. They are performed on an exception basis to allow processing continuity. Both processes are important to the related file maintenance activity. A related issue is the name given to the processes—*Get Valid Customer* and *Get Valid Video*. The implication from these names is that both valid and invalid conditions are dealt with *within* the procedure; only valid customers and videos will be passed for further processing. A missing condition would lead to the initiation of the create procedure. The resolution, then, is to leave the process

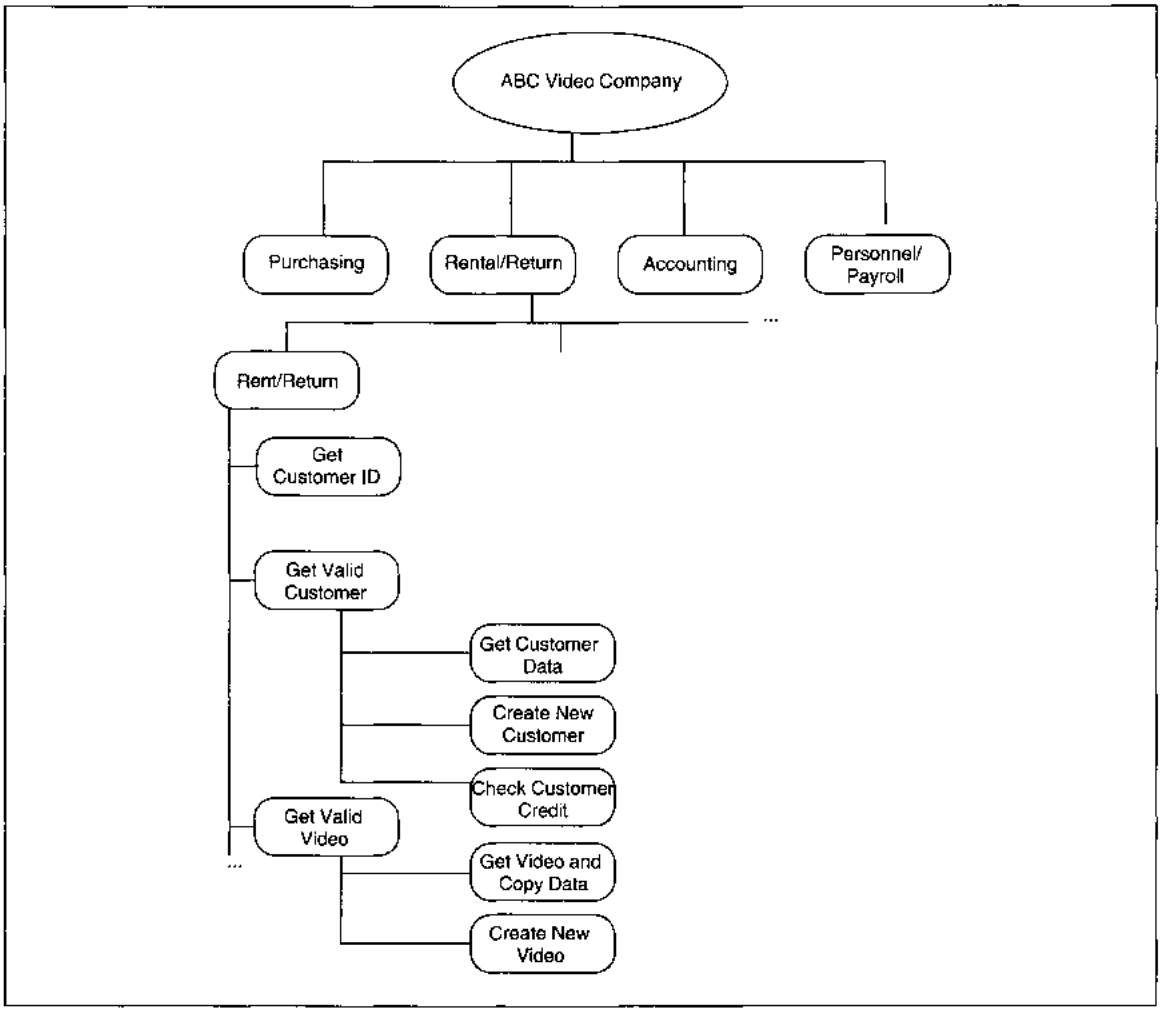


FIGURE 9-25 Partial Decomposition with Details of Get Valid Processing

definitions as they are and to treat the *Creates* as subprocesses under the associated *Get Valid* process. Figure 9-25 shows the details of the two get valid processes for the next level of decomposition.

The final issue is to consolidate these four lists to develop one list, completing the decomposition diagram. The consolidated list is shown, with sequence implied but without selection, in Figure 9-26, the final decomposition diagram. The fourth activity, *Periodic Processing*, has been added. At the moment, this activity includes archival, end-of-day, and query processing. Other processes may be added as we continue through analysis and design. We will use the separate lists of processes again in the next activity, developing the process dependency diagram.

To summarize, process decomposition can be performed independently of ERD development. This step concentrates on activities, processes, and subprocesses of all functions in the BAA. First, all activities are defined, then the processes for each activity are identified and defined. Both activities and processes are defined without regard for current organization, timing of processing, or current procedure. Emphasis is on processes and procedures that are required to fulfill business obligations. The

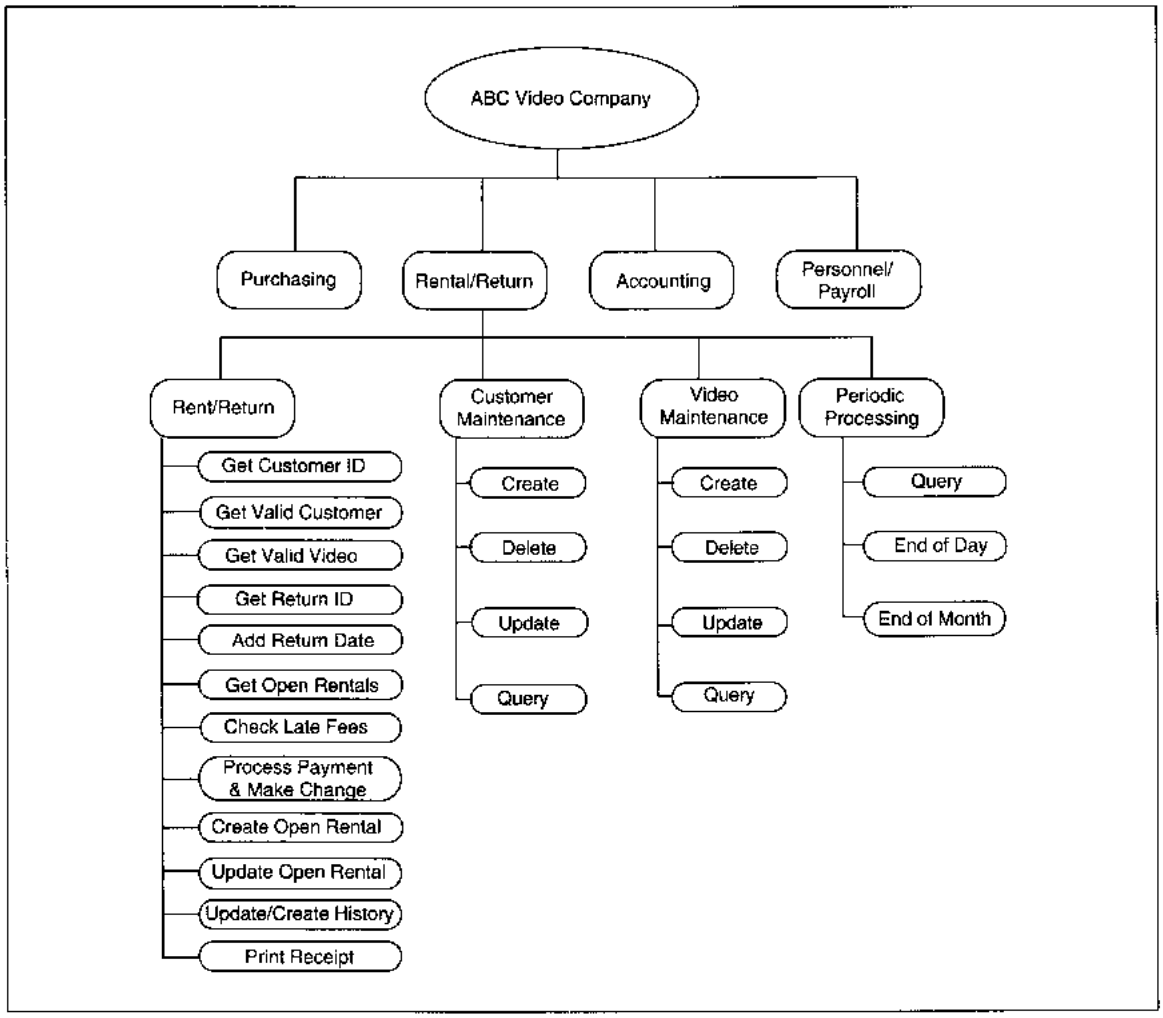


FIGURE 9-26 Completed Decomposition Diagram

final decomposition should be validated through user review.

Develop Process Dependency Diagram

Rules for Developing Process Dependency Diagram

Process dependency relates processes and shows cyclical, logical, and data connections between processes. For each activity and level of processes decomposed, we examine the processes and sequence them by order of occurrence: what happens first, then second, and so on. A diagram using rounded rectangles for each process and arrows to connect them shows the sequencing of the processes. Processes that are independent of other processes are placed on the diagram but not connected to anything. *One diagram is created for each activity.* The steps for creating the process dependency diagram (PDD) are as follows:

1. For each activity, draw the processes on a sheet of paper.

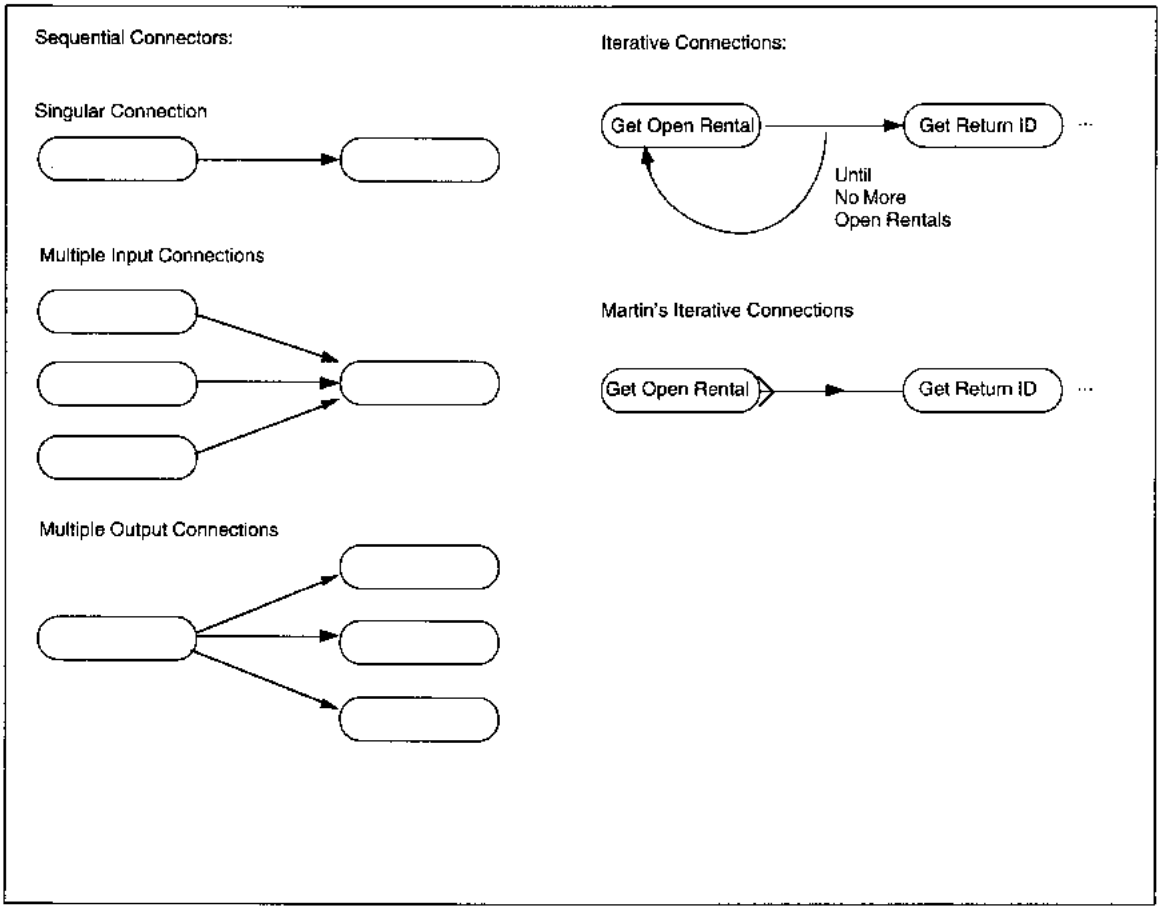


FIGURE 9-27 Types of Process Dependency Connections

2. Examine each process to determine how it is initiated. For processes that pass data to begin work, connect the process to its data receivers. These connections depict the *sequence* of processing.
3. For all connected processes, examine each to determine the cardinality of execution. Define *iterative* processing and document it on the diagram. Be careful to uncouple to the maximum extent possible based on business requirements.
4. For all connected processes, examine each to determine *selection* in processing. For mutually exclusive processes, alter the diagram to depict exclusivity. For all selected processes, add the selection conditions under which processing takes place.
5. For all connected processes, examine each to determine *Boolean* connections. Alter the diagram to include required Boolean logic.
6. Review all connections with the users to verify correctness.

The types of connections between processes in a process dependency diagram differ from those of data flow diagrams discussed in Chapter 7. In process dependency, four types of connections are allowed: sequence, iteration, selection, and Boolean (see Figure 9-27). All connections identify the data

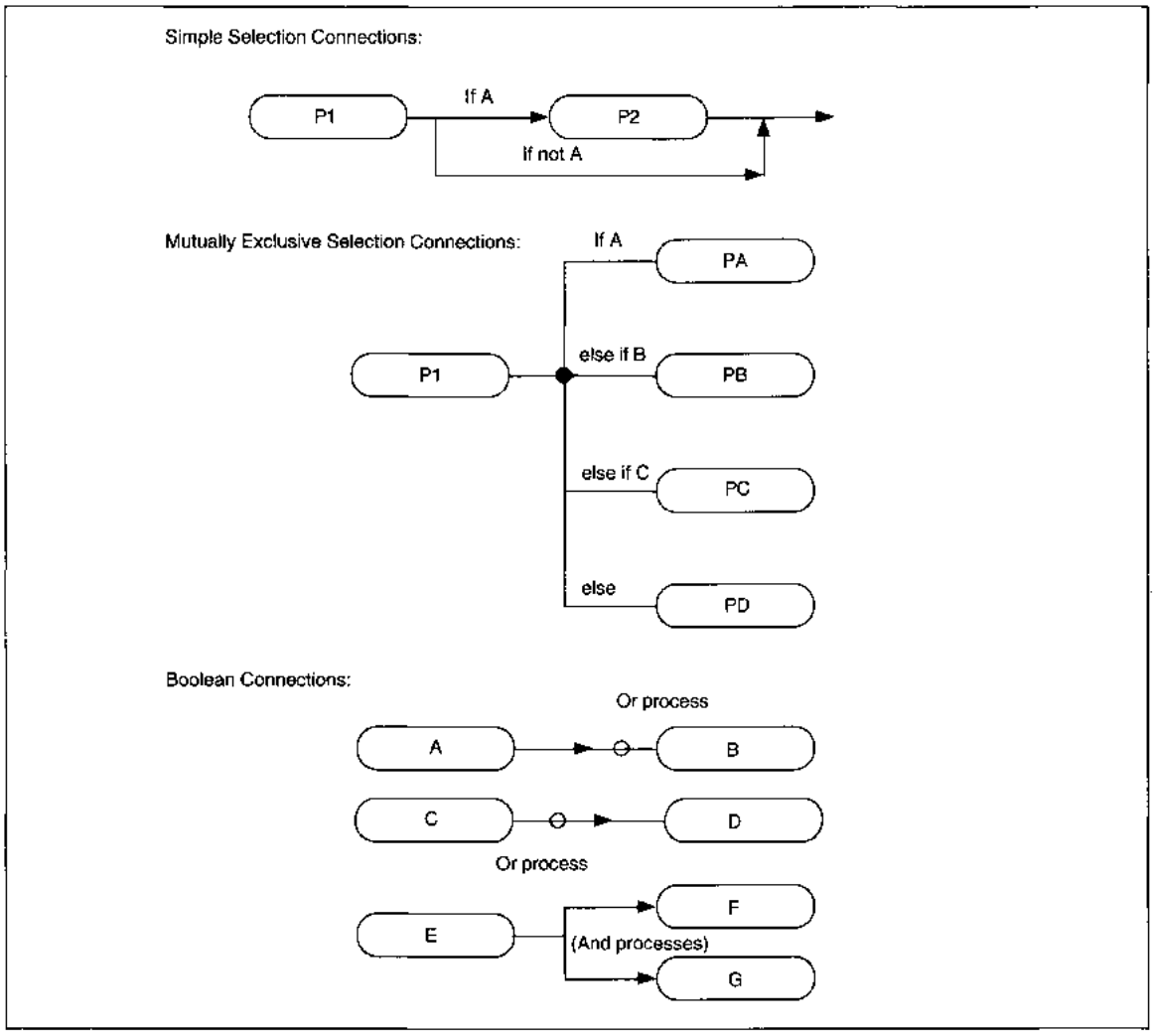


FIGURE 9-27 Types of Process Dependency Connections (*Continued*)

passing between processes by writing its name, when known, above the line.

Sequential connections may be singular or multiple, with many processes feeding another process, possibly feeding the same data (as in reusable processes) (see Figure 9-27). Multiple entries into (or exits from) a single process do not imply any relationship between the multiple processes. That is, no control structure is required to ensure correct order of execution of the processes. In fact, multiple processes could be concurrent, if needed.

Iterative connections between processes are shown with feedback loops, with an indication of how many iterations are performed. A popular alternative is Martin's notation of iteration which uses cardinality indicators, i.e., crows' feet. This notation implies a coupling between processes that may not exist, so the decoupled, more standard iteration loop is used in this text. Both Martin's notation and the decoupled notation are in Figure 9-27.

Selection, or conditional, connections show the alternative choices connected by a solid circle to

differentiate Boolean processes. The *if-then-else* logic conditions are written on each line.

Boolean connections identify 'and' or 'or' types of connections. Boolean connectors use connected lines with an open circle at the junction for optional (or) processes. Simple connected lines which join (or split) to show multiple ('anded') entry (or exit) from processes. That is, any processes not identified as optional or selected are assumed to be executed following the preceding process.

A comment about multiple required process connections is required. Two options for multiple processes, one using multiple directed lines, and one using multiple lines joining or splitting into one line, are discussed. These notations have specifically different connotations. The first, multiple directed lines, shows that any or all of the multiple processes can be executed *and* that control over that execution is imbedded in the processes. The second, multiple lines joining or splitting into one line, specifically identifies 'anded' processes and may require logic to ensure that all are executed.

In addition to showing the logical connections between processes, the lines connecting processes identify process data triggers, that is, data flows from a process to its dependent processes. The last step to the dependency diagram is to identify, as much as possible at this stage, the data that triggers the dependent processes. Attribute names, relation names, or other identifier names are written on the connective lines.

ABC Video Example Process Dependency Diagram

The dependency diagrams for ABC vary in their complexity. The maintenance diagrams are simple because all processes are independent (Figure 9-28). Similarly, the periodic processes are also unrelated (see Figure 9-29). The processes of rental and return are complex and are discussed in detail.

The discussion in the preceding sections identified a dependency of processes in rental/return processing that we need to carry forward: *Print Receipt* is dependent on *Process Payment and Make Change*.

There are other dependencies as well. To show the logic behind the final diagram, we show the process dependencies for each process alternative as we did above. The four choices, again, are rentals with and without returns, and returns with and without rentals.

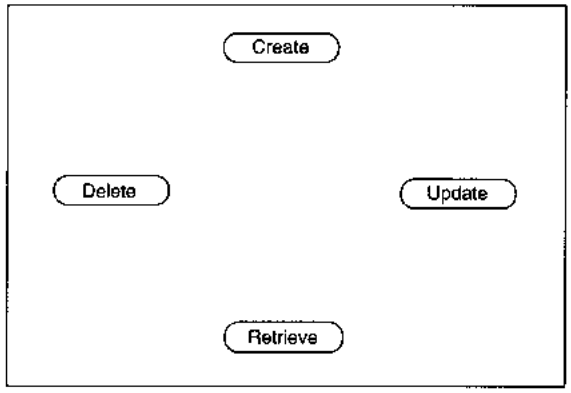


FIGURE 9-28 Maintenance Process Dependency Diagram

The first diagram lists the processes for rentals without returns and, *informally*, draws connections between them. We draw the informal diagram because changes are expected and changing an informal diagram is easier than changing a formal one.

When you start considering the processes, two apparent features are: first, they are not all sequential and second, they are not all done only once. Repetitive processes are *Get Open Rentals* (with *Check for Late Fees* in an iterative loop), and *Get Valid Videos*. Both of these are performed until there are no more

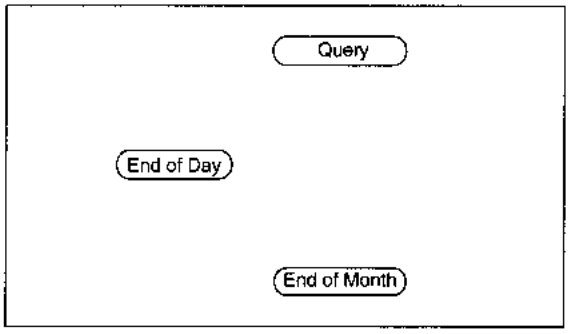


FIGURE 9-29 Periodic Process Dependency Diagram

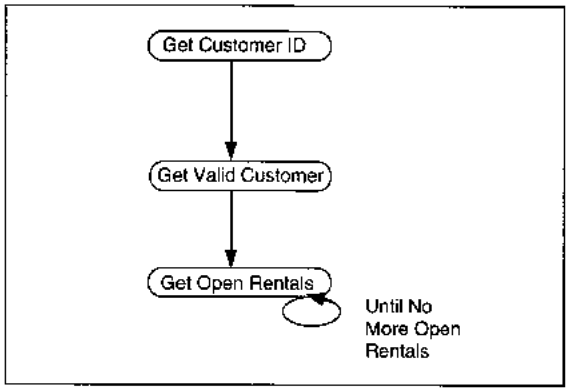


FIGURE 9-30 Get Customer ID and Get Valid Customer Process Dependency Diagram

of the items being got. Draw a circular line from the process to itself to show iteration. Is there any data passed from one iteration to the next? No specific data except an indicator to keep going and, maybe, a memory address at which to store the next data, but this is an implementation detail. We do not identify it now.

Next, walk-through the processing to identify dependencies, drawing the appropriate connections as you proceed. *Get Customer ID* provides the external display and entry processing and passes a *Customer ID* to *Get Valid Customer* (see Figure 9-30). We cannot go directly to any other process because we only want to process valid customers. Consequently, the only dependency is *Get Valid Customer*. *Get Valid Customer* retrieves a customer record, creating a new one if required, checks credit status, and passes a valid customer record to the next process: *Get Open Rentals* (see Figure 9-30). We already said *Get Open Rentals* iterates until there are no more open rentals, and it proceeds to *Get Valid Videos* after it is complete (see Figure 9-31). Could we go directly from *Get Valid Customer* to *Get Valid Videos*? That is, could *Get Valid Videos* and *Get Open Rentals* be concurrent? We need to jump into implementation details again to decide, since there is no business reason why these processes cannot be concurrent. What do we do *in* these processes? The open rentals procedure reads a file, checks late fees, composes and displays a line, and adds to a total field, as required. The video process gets *Video IDs*, reads the Video and Copy relations, composes and displays lines, and adds to a total field. Both processes need access to the screen. So, we need a protocol, or set of rules, to govern where and when a process can display information. They cannot both try to use the same line. The easy method is to force one process to be first and to create an artificial dependency. This is what we will do: *Get Open Rentals* will be first, because we can also have the clerk verify late fees with the customer. So *Get Valid Videos* will be dependent for screen location information (and memory location information, too) on *Get Open Rentals*.

To continue, *Get Valid Videos* iterates until there are no more videos to be rented, then proceeds to *Process Payment and Make Change* (see Figure 9-32). We decided above that both *Create Open Rental* and *Print Receipt* are dependent on payment processing but independent of each other, so we draw two lines from *Process Payment* to each of these processes (see Figure 9-33). Putting all of these dependencies together, we arrive at Figure 9-34.

It is important to practice walking through the diagrams one step at a time to identify dependencies, considering each process alone and as possibly

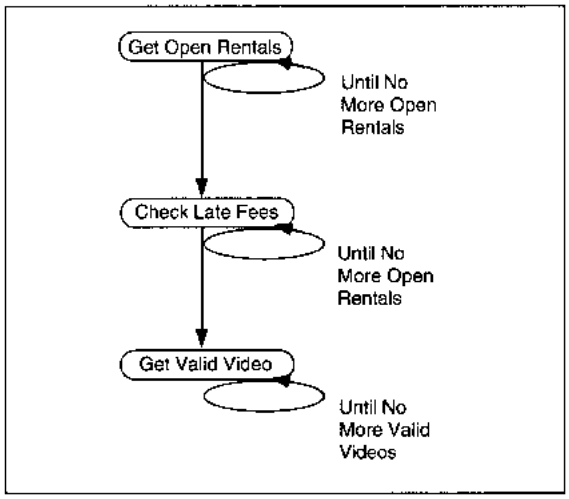


FIGURE 9-31 Get Open Rental and Check for Late Fees Dependent Processes

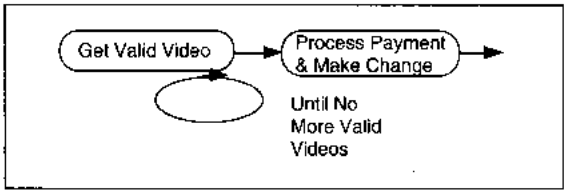


FIGURE 9-32 Get Valid Videos Dependent Processes

connected to all other processes. Ask if all of these processes get done once, or are some iterated? Can groups of processes that iterate together be identified?

Draw each connection as you consider it so it is translated properly. Finally, identify the data triggers when they are known so that you know *what you don't know* when your diagram is 'complete,' meaning it contains all known information.

Figure 9-34 shows the dependency diagram for Rentals Without Returns. Try to develop the process dependency diagram for each of the options on your own, without looking at the answers. The other process option dependency diagrams are shown as Figures 9-35–9-37. Keep in mind that iteration and sequencing required for each alternative way of processing are important because we eventually must consolidate into one diagram—that is, a composite of the individual diagrams. We only discuss the difficult connections and connections that change the way we think about the rent/return process (i.e., may alter our mental model).

In Figure 9-35, we have several differences from the first dependency diagram (Figure 9-34). First, *Get Valid Customer* is only done for the first return, so we need a selection connector. If the Video ID (or Open Rental) is not the first, we proceed to *Add Return Date* and *Check for Late Fees*. We have two choices on the iteration grouping shown (see Figure 9-38). The first strategy shows coupled logic with the loop encompassing *Get Request, Get Open Rentals, Get Valid Customer, Add Return Date,* and *Check for Late Fees*. The second strategy shows several loops that may look more awkward, but reflect *required* coupling for the processes. This logic is uncoupled and shows three iterative loops, all iterating for all returns. The first loop is *Get Open Rentals*. The second loop is *Add Return Date*; the third is *Check for Late Fees*. Both of these alternatives would be acceptable in program specifications and selection for one over the other might be based on the common iteration pattern. At the logical requirement level, however, the preferred method is the more loosely coupled one because the iteration cycles may change when we consolidate the dia-

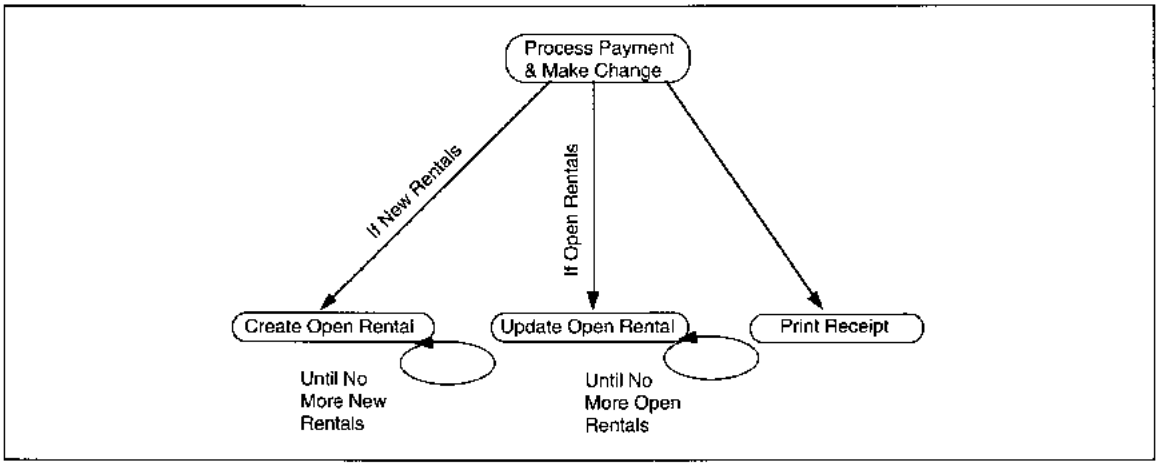


FIGURE 9-33 Process Payment and Make Change Dependent Processes

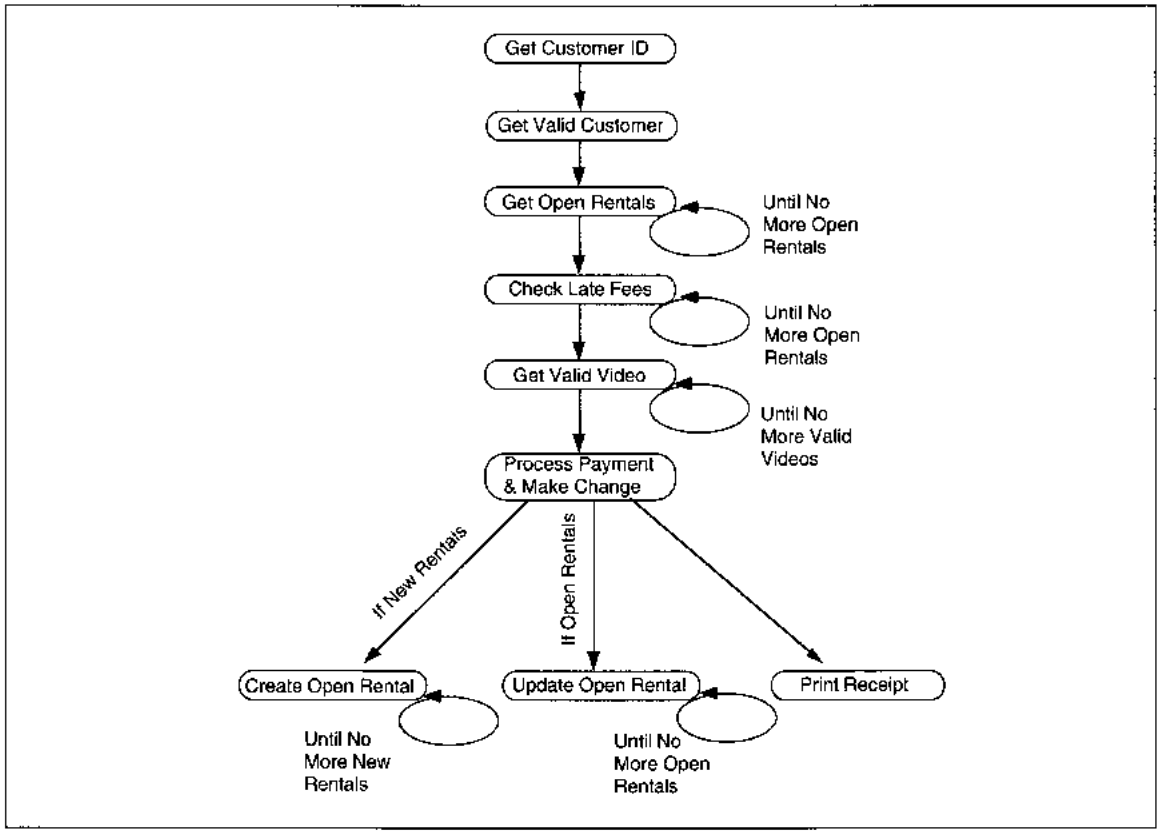


FIGURE 9-34 Rentals without Returns Dependency Diagrams

grams. If the diagrams were already consolidated, we could go to the program design level of detail to choose the iteration grouping. The more uncoupled dependency is shown in the completed diagram, Figure 9-35. The decisions about preferred looping are deferred until design.

The second difference in Figure 9-35 is that a receipt has selection criteria applied to its creation. A receipt *must* be printed whenever a payment is made, and *may* be printed upon request of a customer with returns but no payments. This selection is shown on the diagram.

Figure 9-36 shows rentals with returns. In this procedure, we have two iterative cycles: one for return processing and one for new rental-item processing. These, in effect, consolidate the previous two diagrams. Notice here that the initial input is from *Get Customer ID* so returns do not include the selective execution of *Get Valid Customer*. Also notice that we again have coupling options for return processing (the coupling options for return processing are shown in Figure 9-38). In the selected option we have three iteration cycles. *Get Open Rentals* is performed for *all* open rentals. *Get Return IDs* and *Add Return Date* are performed together for all returns which may be a subset of open rentals. *Check for Late Fees* is performed for all open rentals whether or not returned today. The final difference is for history processing which is selected for open rentals with *Return Date* equal to today's date.

The last procedure is for returns with rentals (see Figure 9-37) which is similar to Figure 9-36 except for the initial entry of information. If a return is first,

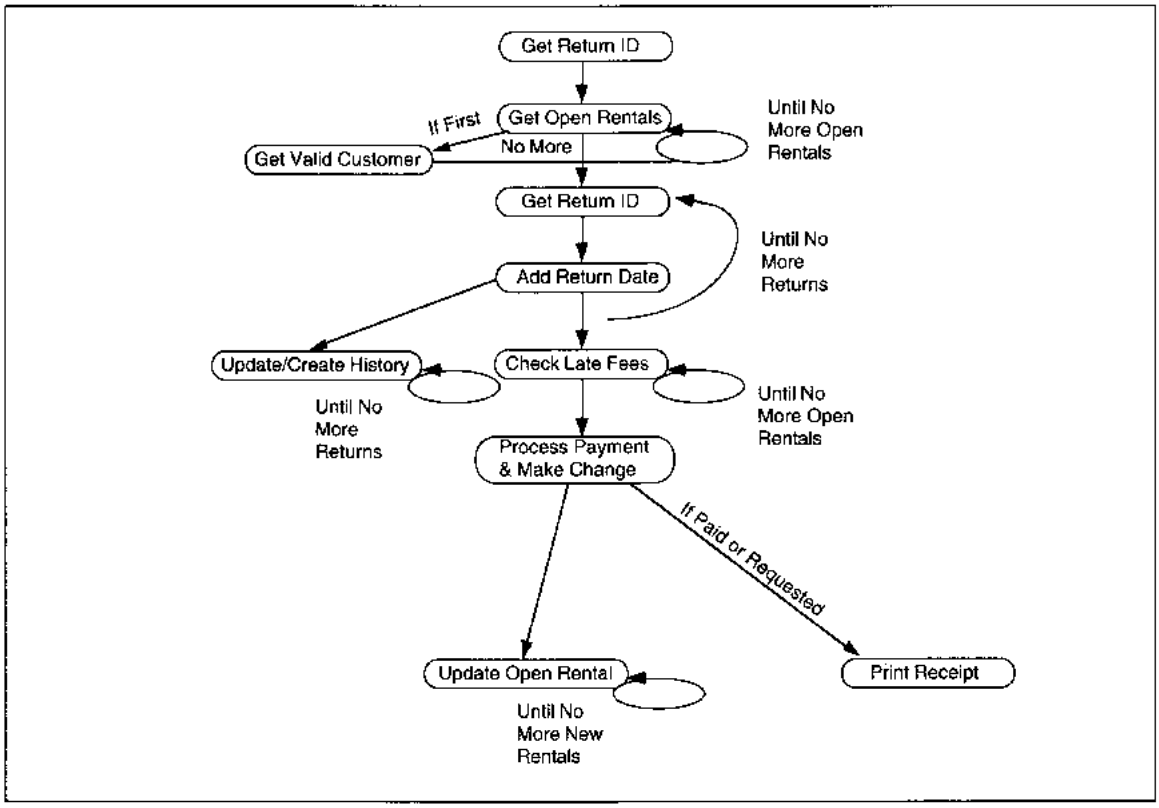


FIGURE 9-35 Returns Without Rentals Dependent Processes

Get Return ID is the first process and we need the selective execution of *Get Valid Customer*.

Now, we are ready to consolidate the diagrams into one (see Figure 9-39). The obvious complexity is in dealing with all of the return options, so they will be done first. If we look at the first step of each procedure, the differences are that for returns, *Video IDs* are entered first and for rentals, *Customer IDs* are entered first. If these are separate processes, we have a problem knowing which is, in fact, being executed. If we consolidate these processes, we can use program logic to figure out which numbers are for videos and which for customers. This means that *Get Return ID* and *Get Customer ID* are replaced with one process we will call *Get Request*. This process will select either *Get Valid Customer* or *Get Open Rentals* depending on the data entered. This change is reflected back to the decomposition diagram also.

Next, for returns, we need selective execution of *Get Valid Customer*. To consolidate, we need to know whether we are processing a rental or a return. Two changes are required. *Get Request* must call either *Get Valid Customer* or *Get Open Rental*, depending on the type of entry. This is indicated by the selection logic in Figure 9-39. Second, *Get Request* has to pass some indicator to *Get Open Rentals* that it is the caller; this means data is triggering the process.

The last return issue is what to do with *Check for Late Fees*. There are three options. First, include it in *both Get Open Rentals* and *Add Return Date* to ensure complete processing of late fees for old and new returns. Second, leave it separate and execute it

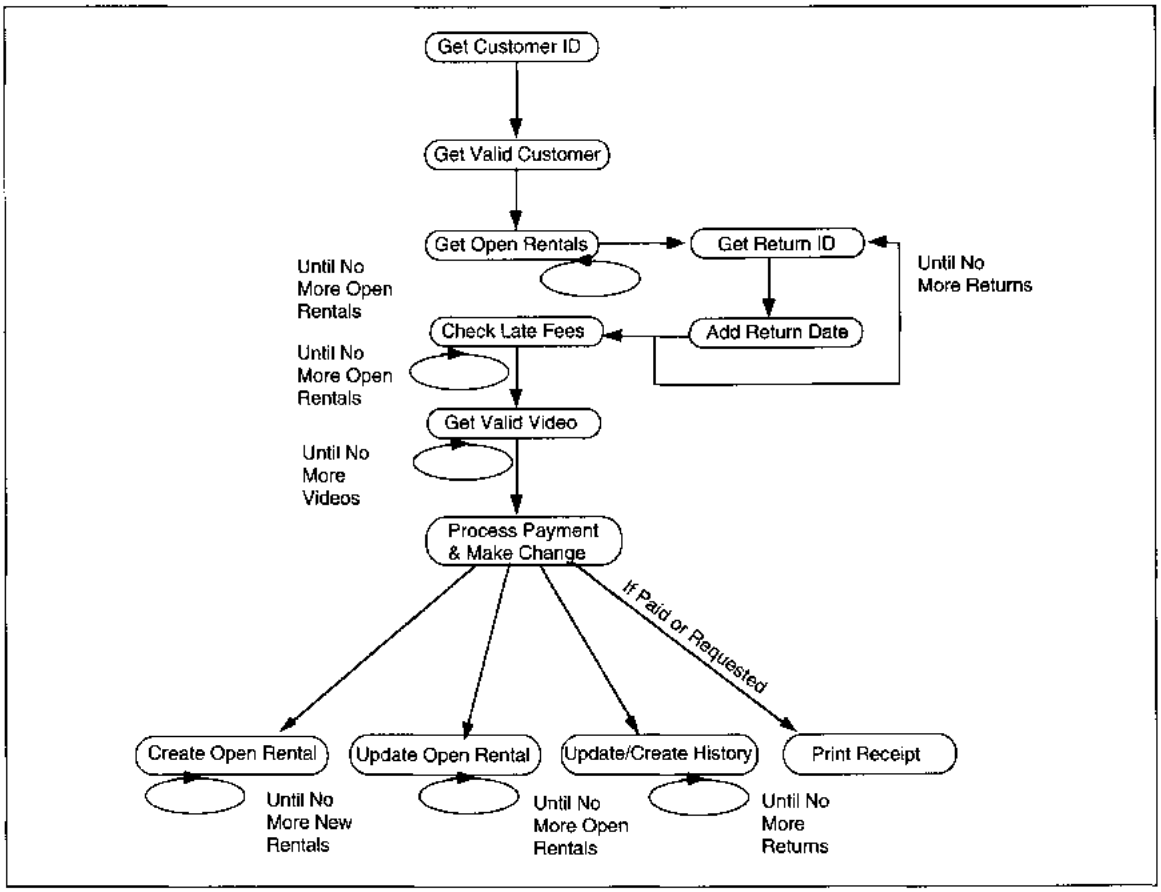


FIGURE 9-36 Rentals with Returns Dependency Diagrams

for all open rentals, including those returned today, as a separately iterated process. The first option guarantees double processing for all returns when rentals are also done. The second option requires somewhat complex logic for memory loop processing. Both options are acceptable technically and from a business perspective. The last option is to defer a decision. Since we have *no business basis* for a decision, we leave the process on the diagram and defer any decision about grouping until design. The final dependency diagram is shown in Figure 9-39 and reflects all of the decisions discussed above.

The dependency diagram for periodic activities is in Figure 9-40. The diagram is somewhat strange because there is no necessary connection between any of the processes. It reflects the independence of the processes, and is the basis for the PDFD which completes the dependency diagram. This type of independence also identifies possible concurrent processes and is considered a normal diagram. Notice that even though these processes would be connected on a menu for processing, no menu selection options are shown at this logical level. The reason is that the *business* requires no menu.

To summarize, to develop the dependency diagram list the processes for an activity, in sequence. Then, examine each process to determine its relation to all other processes. If complex processing is involved, as we have here, separate out the options

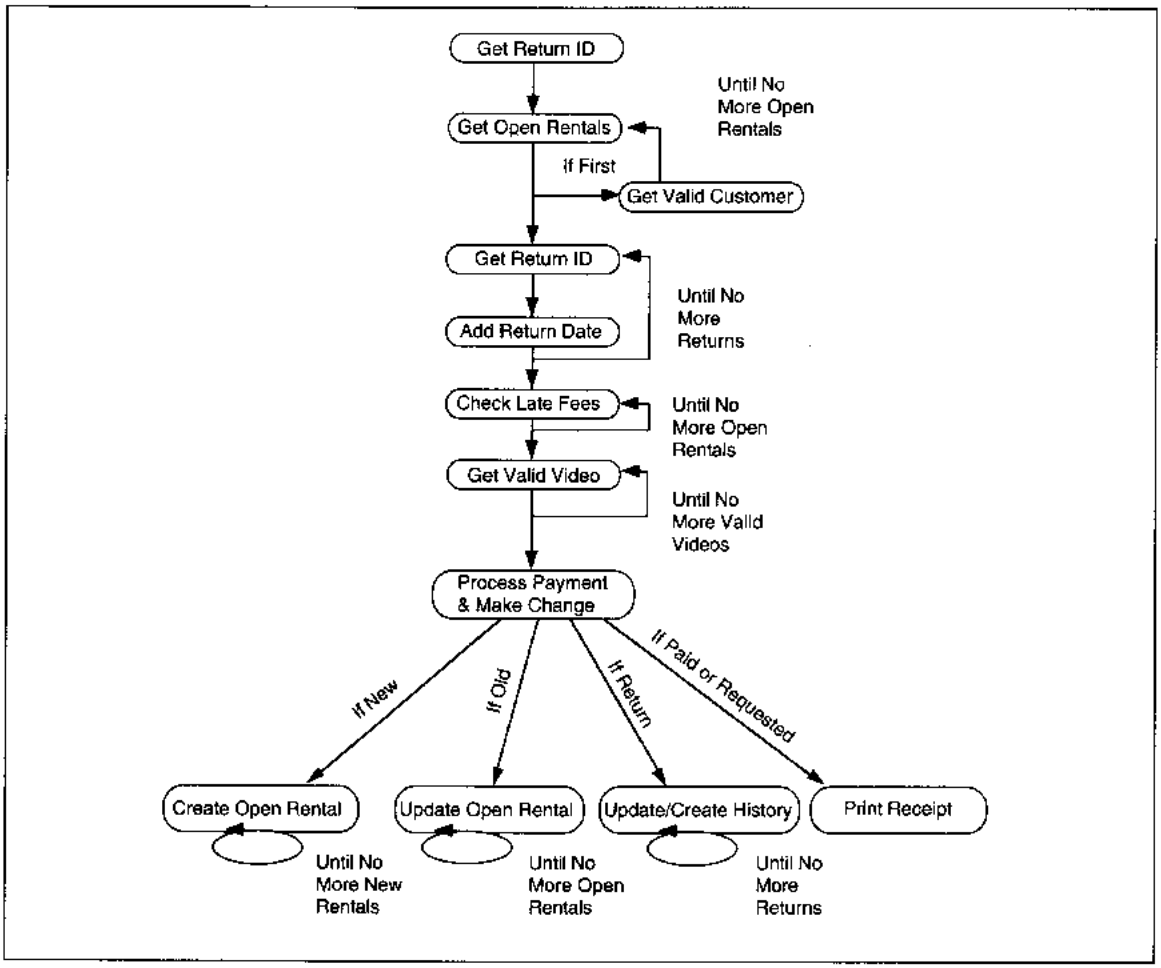


FIGURE 9-37 Returns With Rentals Dependent Processes

and develop dependency diagrams for each option. Be careful to couple processes based on business requirements rather than on convenience. Convenience is decided in design. Consolidate any options and only change processes if required to support integrated process interactions.

Notice also that in going back to the client for information during this procedure, we obtained information we would otherwise not have about the need for periodic purging of the file and requirements for IRS documentation.

Develop Process Data Flow Diagram

Rules for Developing Process Data Flow Diagram

This is a three step process:

1. For each process dependency diagram, examine every process to determine if external events provide information used in the execu-

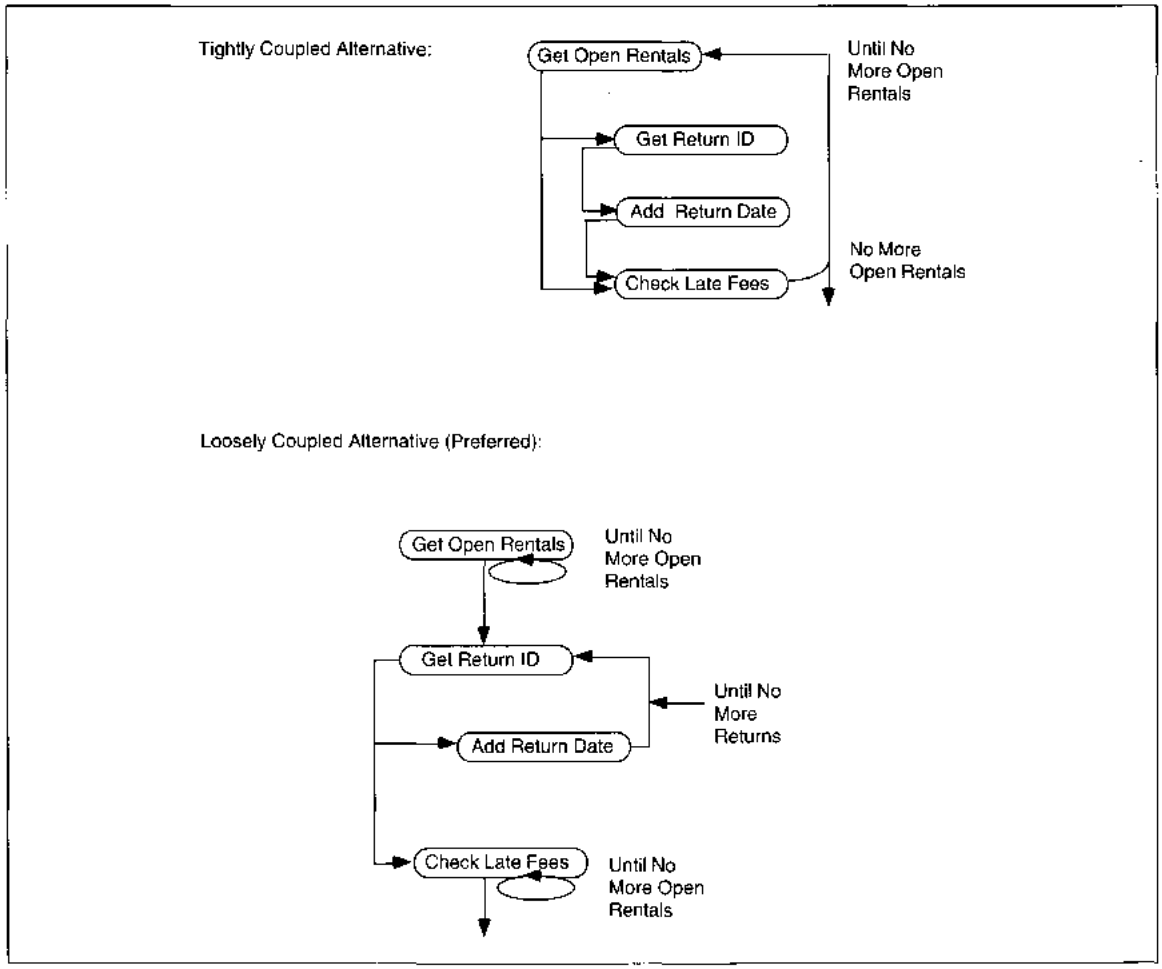


FIGURE 9-38 Alternative Coupling Strategies in Return Processing

tion of the work. For each external event, add an event trigger and identify the event (or the data it provides).

2. For entities from the ERD, examine their use by processes in each diagram. For known connections, add one file for each entity to the diagram and connect them to processes with arrows depicting the direction of data flow. For all files, when a relation is not the unit of data retrieved, list the attributes that make up the data flow.
3. Review the triggers and files with the user to verify correctness.

Using the process dependency diagram, first add the information about triggers, that is, the data or events that trigger each process. If arrival of information from another process is the trigger, identify the data on the lines connecting the rectangles. Use large arrow outlines for event triggers. Use single-directed lines for data triggers (see Figure 9-41).

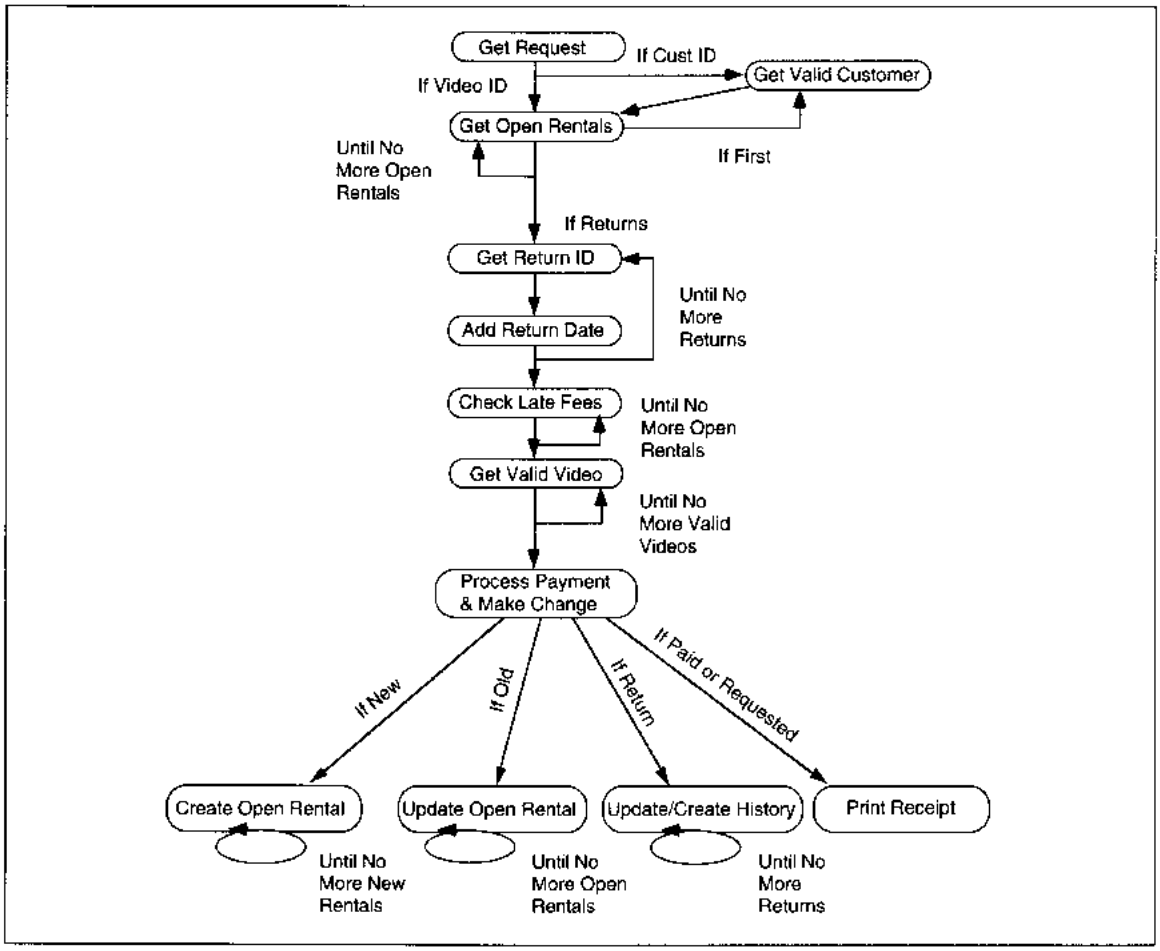


FIGURE 9-39 Consolidated Process Dependency Diagram

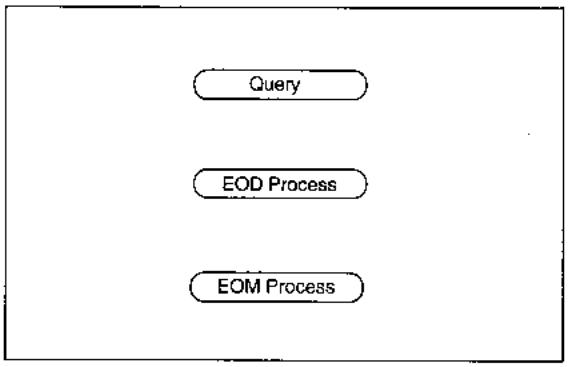


FIGURE 9-40 Periodic Activitied Dependency Diagram

Each process *must* be triggered, or initiated, by either an event or arrival of data. If you have a process without either data or event as input, then you have missed information during data gathering and should return to the user to obtain the information.

Identifiers for both data and events should link directly to some entity. The trigger may be the arrival of some entity or may be some partial data from an entity. If the identified data does not map directly to an entity from the ERD, you are also missing information and should return to the user to obtain the information.

Next, data files are identified, if they are known. Not all files are necessarily identified at this point of

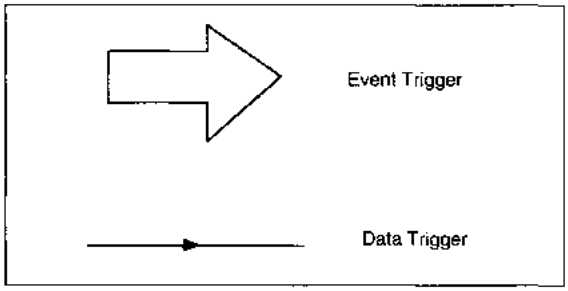


FIGURE 9-41 Trigger Identification on Process Data Flow Diagram

the analysis. However, most information that is required in persistent files will have been identified as entities on the ERD. The files are connected to processes with the appropriate arrow signifying the direction of data flow. If a unit of data other than a logical relation is required, the lines connecting files should be labeled with their contents.

PDFD validation is performed last to guarantee that all functional requirements are satisfied by the processes depicted. The validation walk-through uses both the original text or functional specification, plus the decomposition from the specification, plus any additional user requirements or information obtained throughout the analysis activities.

ABC Video Example Process Data Flow Diagram

To complete the ABC PDFD, begin with the final process dependency diagram. We examine each process sequentially, adding events and data files as needed to complete the logical processing. For each process, ask: How does this process know to execute? What information does it need? Where does the information come from? Ask these questions without paying attention to current connections from other processes. For each process, when you have the answers, look at the current connections and decide if they completely define the required list of information. If not, define the external 'triggers'—either data or events—that initiate the processing. The individual chunks of the diagram on which we are working are shown with the discussion. They are integrated into one diagram at the end of the discussion.

The first process, *Get Request*, requires input of either a phone number or a video ID to begin execution. The information is provided by the customer and entered into the computer (either by scanning or typing) by the clerk. Since the information is externally generated by a rental or return request, the data being entered is an event. That is, arrival of a *Customer ID* or *Video ID* into the computer triggers the *Get Request* process which begins the sequence of processes for rental/return processing. The hollow arrow is added to the diagram to show the arrival of the *Request* event (see Figure 9-42).

After the request is entered, the process determines which data were entered and passes control to the appropriate process. If the *Customer ID* was entered, the *Get Valid Customer* process would be triggered. For that process, customer information from storage is required for validation and credit checking. A file symbol for a customer information file with a line indicating data *into* the process is added to the diagram (see Figure 9-42). Since there is a possibility that the customer is new, an arrow going *to* the *Customer File* is also shown.

Next in rental processing, the *Open Rentals File* should be read to retrieve all information about *Open Rentals* relating to the present customer. These are formatted and displayed. The file is added as input to the *Get Open Rentals* process (see Figure 9-42).

If the request entered had been a *Video ID*, the *Get Open Rental* process followed by the *Get Valid Customer* process would have been triggered. The control of these processes is shown by the selection arrows from the dependency diagram. The data and trigger requirements do not change. We might make a note that, in this execution of *Get Valid Customer*, we do not allow new customers to be added. That is, it should be logically impossible for a new customer to have a return.

Return processing takes place next with two possible variations. First, if the first *Request* was a *Video ID*, there already *is* a return *Video ID* in memory and no *Get Return ID* is triggered. Instead, *Add Return Date/Late Fees* is triggered. The second variation is in the rental process; returns are entered after open rentals are displayed. The *Get Return ID* process is

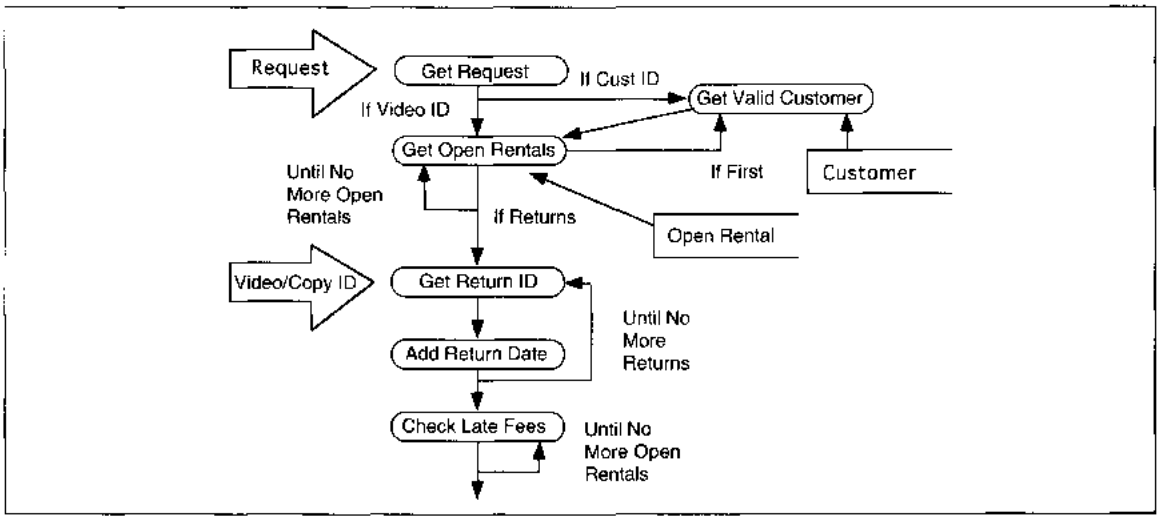


FIGURE 9-42 Process Request 'Chunk' of PDFD

triggered but it now needs the *Video ID*, external information, to process. The event trigger added to the process, then, contains *Video IDs* (see Figure 9-42). The data is made available to the *Add Return Date* process. *Get Return ID* and *Add Return Date* iterate until the *Return ID* is ended (exactly how is decided during design). Then all *Open Rentals* (and returns) are *Checked for Late Fees*.

Next, the *Get Valid Video* process executes to identify videos requested for rental. The information needed for this process comes from an external trigger, the customer-supplied *Video ID*. The ID is validated by reading the *Video File* and a *Copy File*. For rental/return processing, the *Video* and *Copy* files are always used together, so they are shown in one file symbol. By doing this, we are reminded that we need a user view that connects the two relations for rental processing. Customers are to be reminded when they have already rented a particular video, therefore, *Customer History File* is also read during this process. Its file symbol is added to the diagram.

The *Total Amount Due* is passed to trigger *Process Payment and Make Change*. The *Total Amount Due* is displayed from the previous process and awaits the external entry of *Customer Payment Amount* to compute change. This requires an event trigger for *Customer Payment Amount* (see Figure 9-43). The formula used is *Total Amount Due – Customer Payment Amount = Balance*. When the *Balance* is zero, the rental/return process is complete and all files may be updated as required. Each *line* of the rental/return, signifying either an existing *Open Rental* with/without *Return Date* or a new *Open Rental* is processed separately to determine the next process to execute. This represents the *normal* process; now we must also think about exceptions: What if the balance does *not* go to zero? Can a customer ever overpay and leave so fast that they are owed change? Can a customer ever owe money and leave without paying? If the answer to either of these questions is yes, we also need an optional event trigger *End of Payment* that forces completion of the *Process Payment* process and shows that the customer is owed money. For the present, we assume that we iterate through *Process Payment* until *Balance* equals zero. Finally, process payment needs to provide information for *End of Day* totals. A file for EOD data will be created from this process. Notice that this file is not on the ERD, but should it be? It does not represent an entity that the company keeps information about, or does it? When the ERD was developed, we focused on the rent/return processes only and ignored nonrental activities. By ignoring accounting and its needs for rent/return data, we

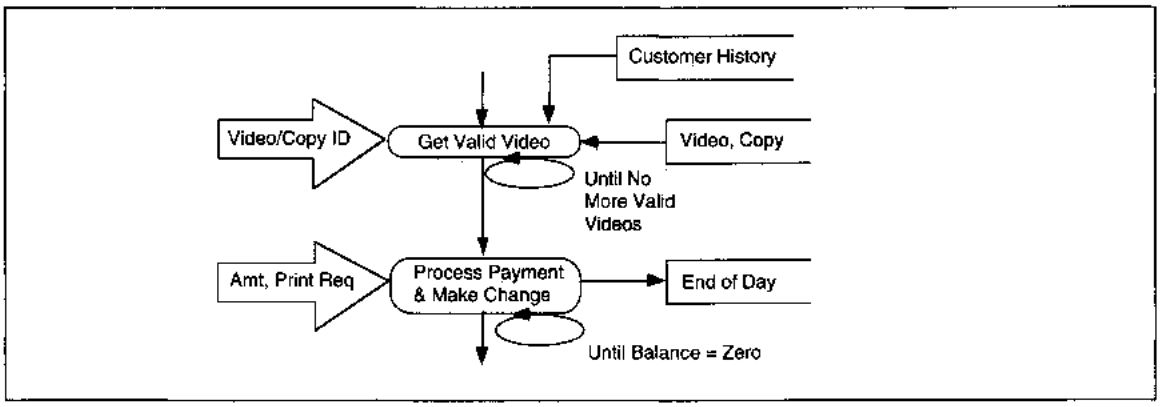


FIGURE 9-43 Rental and Payment 'Chunk' of PDFD

missed the entity for financial information relating to rent/return. We should recheck with the accountant, but it appears that the *EOD Accounting Information* should be an entity that is added to the ERD, connected to the *Open Rental* entity.

The last 'chunk' of the PDFD is for file update and print processes. For each new rental, *Create Open Rental* writes the new rental to the *Open Rental File* (see Figure 9-44). For each unpaid return (i.e., existing *Open Rental* with *Return Date* not null), the *Open Rental* is rewritten to the file (see Figure 9-44). For each paid returned video, some logical delete indicator must be set and the *Open Rental* is then rewritten to the file. If an existing *Open Rental* had no processing, no action is required, but it might be easier, and more consistent, to only look at return and paid criteria to determine correct writing. No business criteria exist for this option, therefore, this is a design decision not made here.

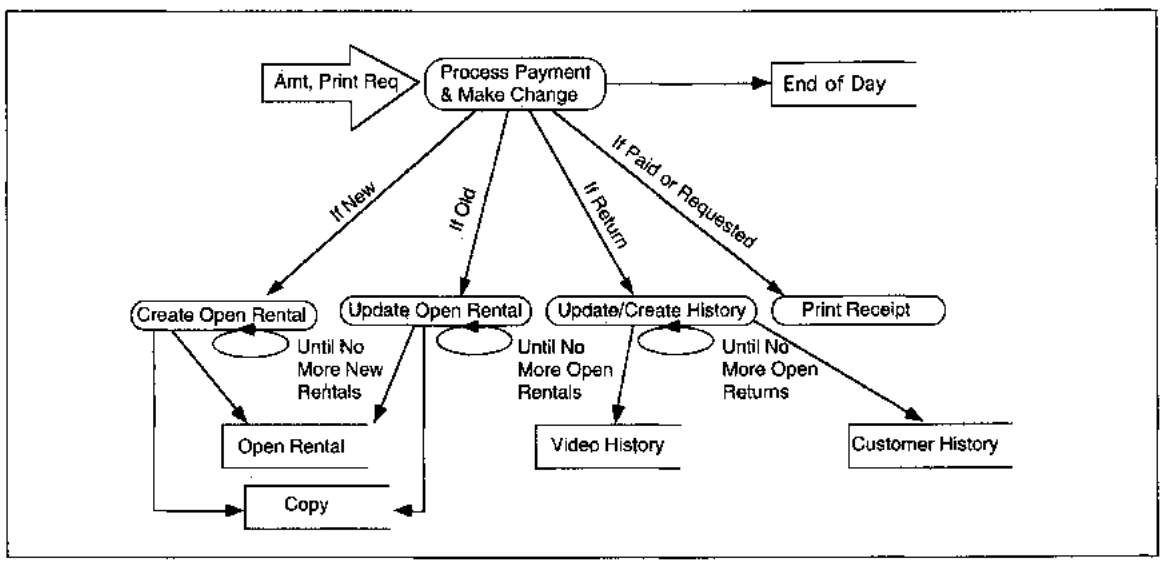


FIGURE 9-44 File and Update 'Chunk' of PDFD

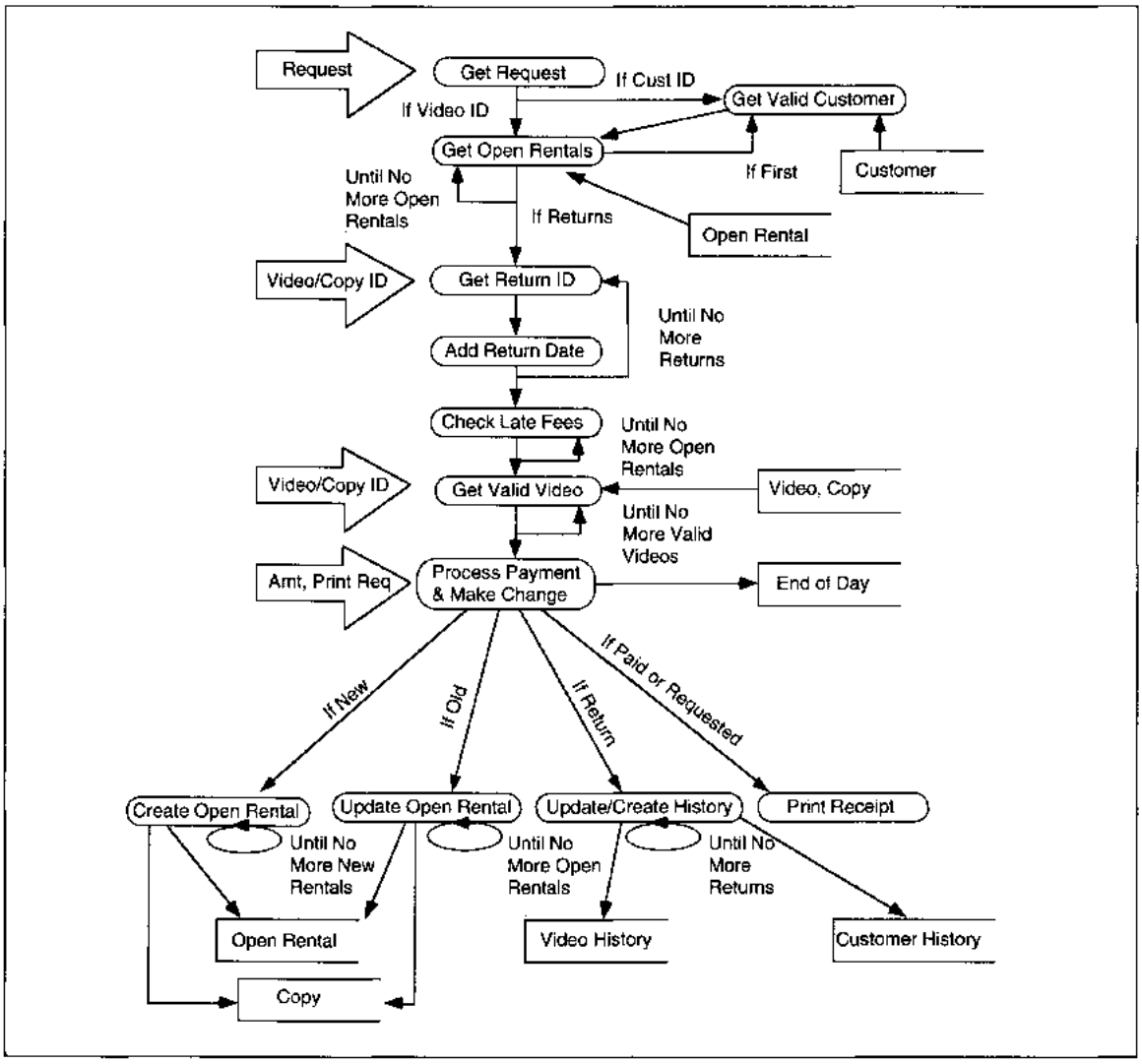


FIGURE 9-45 Consolidated Process Dependency Diagram

Next, history file processing updates both *Video History* and *Customer History*. Notice that the details of history processing require another level of decomposition, which is left as a student exercise. Last, the receipt is printed. The data trigger from *Process Payment* initiates this process; the physical output is not on the PDFD.

The composite PDFD with all of the chunks integrated is shown as Figure 9-45. Before you look at the other diagrams, try to develop one or all of them yourself. The remaining PDFDs are shown as Figure 9-46, for query processing, and Figure 9-47 for customer maintenance processing. The PDFDs for *Video Maintenance* processing are a practice exercise at the end of the chapter.

Next we evaluate the PDFD for completeness based on the decomposition information from the client and the original statement of the problem (Chapter 2).

Errors to watch for are:

1. Processes on the decomposition that are not self-contained are not processes. For instance, 'end process' is a system action, not a business process. You do not need a process to which all other processes feed to show a termination point on a PDFD (see Figure 9-48).
2. Processes on the PDFD that are not *identical* to the processes on the decomposition (see Figure 9-49).
3. If process data trigger contents cannot be identified, there may be no dependency. Reevaluate the relationship, talking to the user if necessary, to determine what data are required for the dependent process.
4. If a process data trigger exists in a different time, the connection is probably wrong. For instance, you might be tempted to connect query processing to rental/return in some way (see Figure 9-50). These are disjoint activities and the only connection is through the database.
5. For query processing, do not try to simulate a menu selection process in the PDFD (see Figure 9-51). Each type of query has its own event trigger requesting information. Each

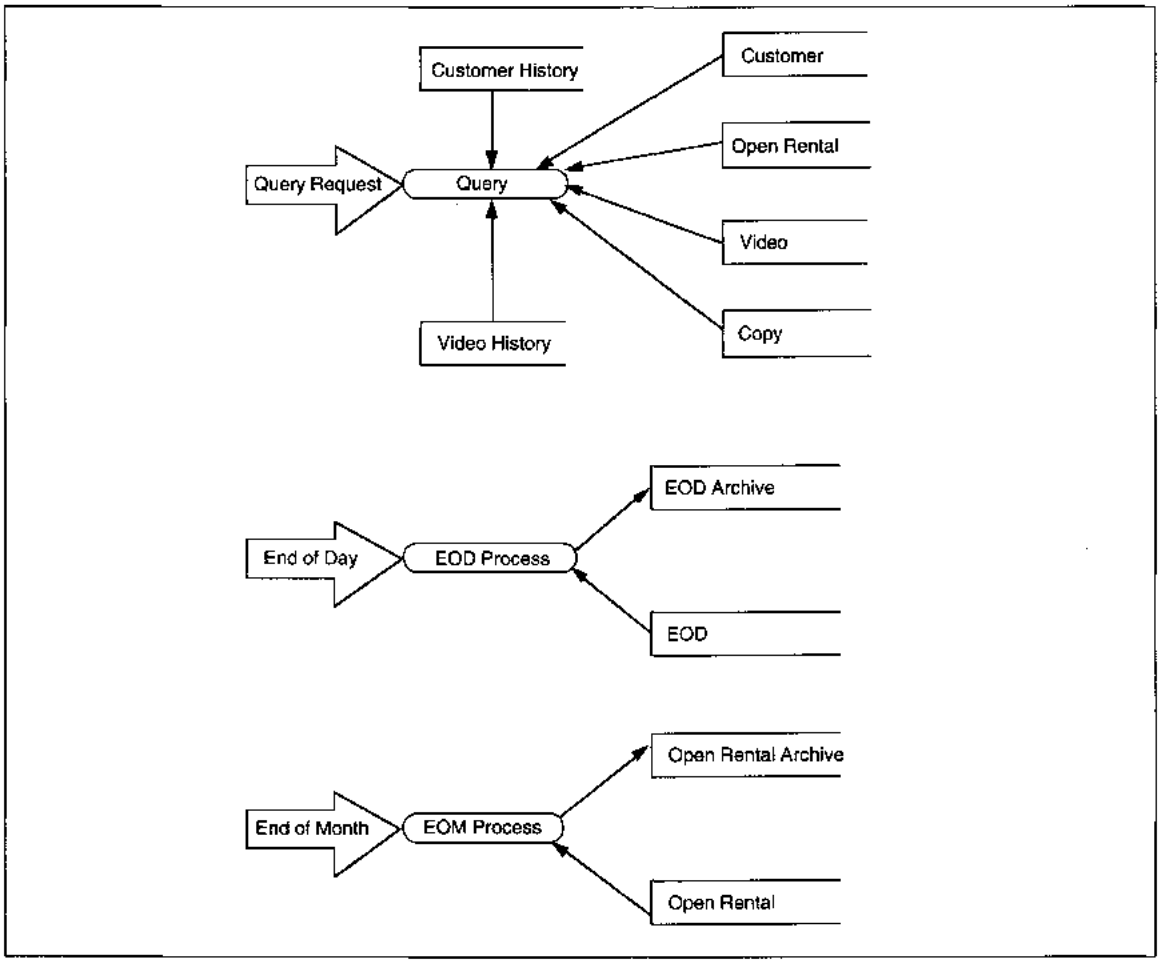


FIGURE 9-46 Periodic Activitied PDFD

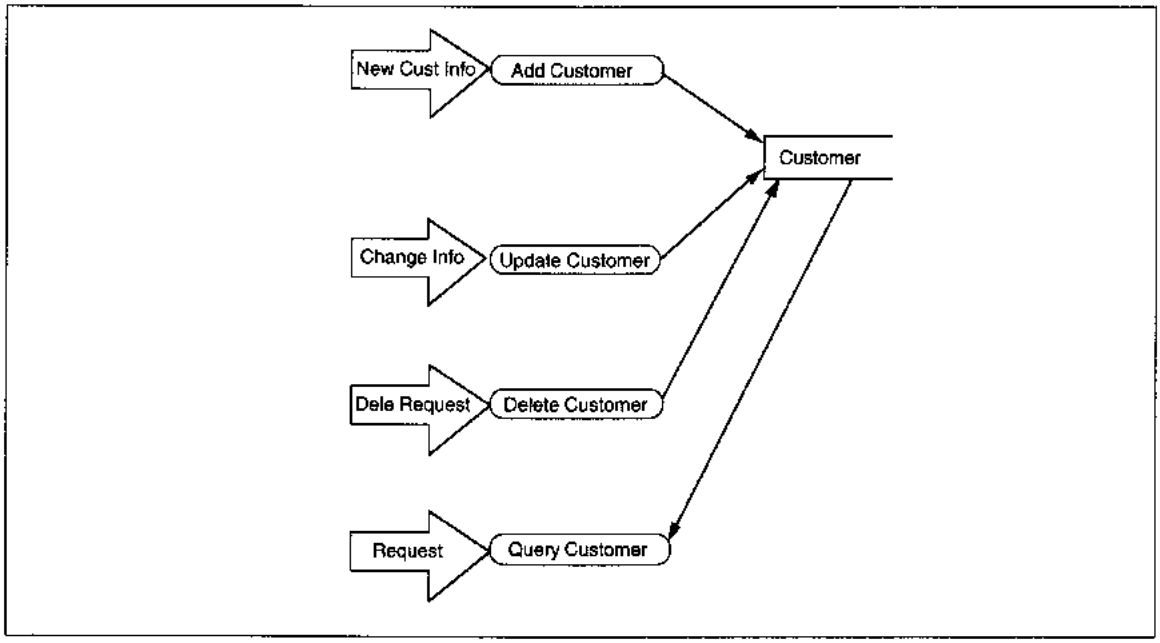


FIGURE 9-47 ABC Customer Maintenance PDFD

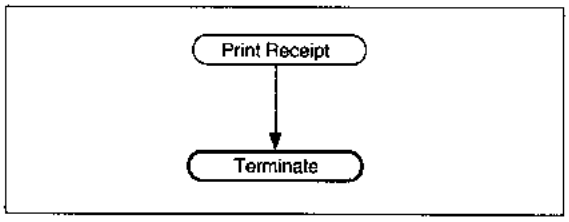


FIGURE 9-48 Nonprocess Problem

type of query is distinct and separate from all other queries. The queries may share files.

6. When more than one activity is shown on a PDFD, problems are encouraged and the diagram no longer clearly delineates any process (see Figure 9-52). Place *at most* one activity decomposition on a page. Use one side of the paper only. Keep in mind that most of the time, the information will be on a CASE tool until printed for documentation, so there is not really much wasted paper.

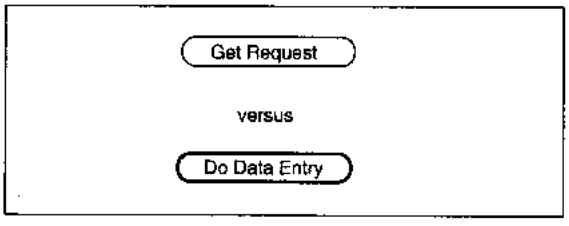


FIGURE 9-49 Name Names Do Not Match Decomposition

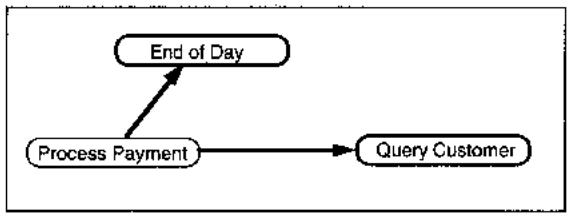


FIGURE 9-50 Data Trigger Timing Problem

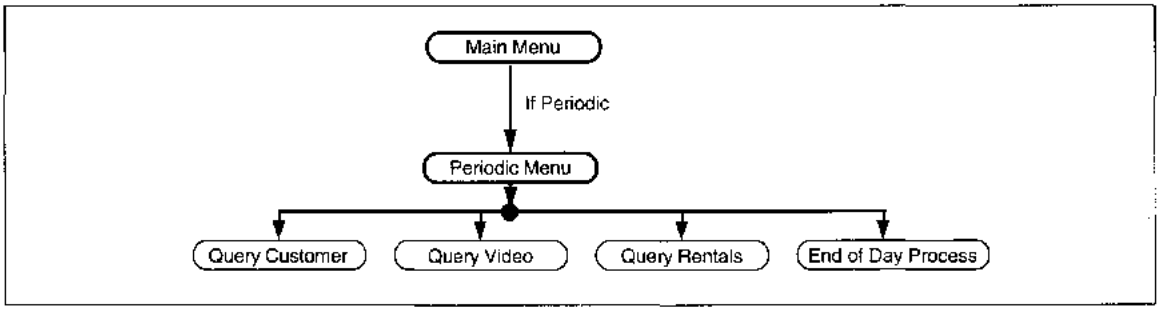


FIGURE 9-51 Simulated Menu Problem

Develop and Analyze Entity/Process Matrix

Rules for Developing and Analyzing an Entity/Process Matrix

This matrix is composed of the results from the ERD and process decompositions; it requires neither the process dependency nor the PDFD for completion. Along the left margin, list each lowest-level process from the process decomposition diagram. Use the lowest-level processes, such that all elemental processes for the organization and application area are present. Along the top, list the normalized entities from the ERD, with one entity in each column.

Completely identify which processes are allowed to Create, Retrieve, Update, and Delete (CRUD) each entity. Enter one or more of the letters as allowed by the current organization's policies and procedures for each entity.

When the matrix is complete, entities are grouped by their affinity, or closeness, in processing entities. If you do this step manually, group processes that share create responsibility first. If the number of clusters is reasonable for the size of the project, stop. When you have analyzed the entire matrix, rearrange the matrix by its clusters. You may have several clusters that overlap. That is normal and not a cause for worry. If you have only one cluster, reanalyze as necessary using first update processing, then delete processing, then retrieval, as the clustering criteria. When you obtain a reasonable number of clusters, go to the next step of the analysis. A reasonable number may be one to five for a small application, such as ABC, or seven or more for a large application.

To perform manual affinity analysis, perform the following procedure (here we do create affinity only). Keep in mind that you are 'normalizing' process-entity relationships. Look at each process and its entities. For an entity/process cell, look down the column and identify other cells in which create processing is done (C). Make an erasable, colored mark in those cells. Do this for all entities for each process.

Even though it is an extra step, and quite a bit of work, create an interim matrix for each potential cluster. This interim matrix makes your visual inspection of relationships easier and actually speeds the affinity analysis. Iterating, build the interim matrix, analyze it as described below, and add the resulting cluster(s) to the new process/entity matrix. Iteration is required because the interim clusters may change as the relationships of each potential cluster are analyzed.

To analyze a potential cluster, start at the first process and look at the data it shares with the next process in the list. Do these two processes share 80%[7] or more of their data creation (or update, delete) responsibility? If yes, mark the original matrix to show they are together and add the processes and their entities to the interim matrix. If the percentage is less than 80%, circle the second

7 80% is not a hard number. You adjust the percentage affinity needed to find multiple clusters. If all processes share all responsibilities, the organizational processes must first be redesigned, then this analysis is repeated.

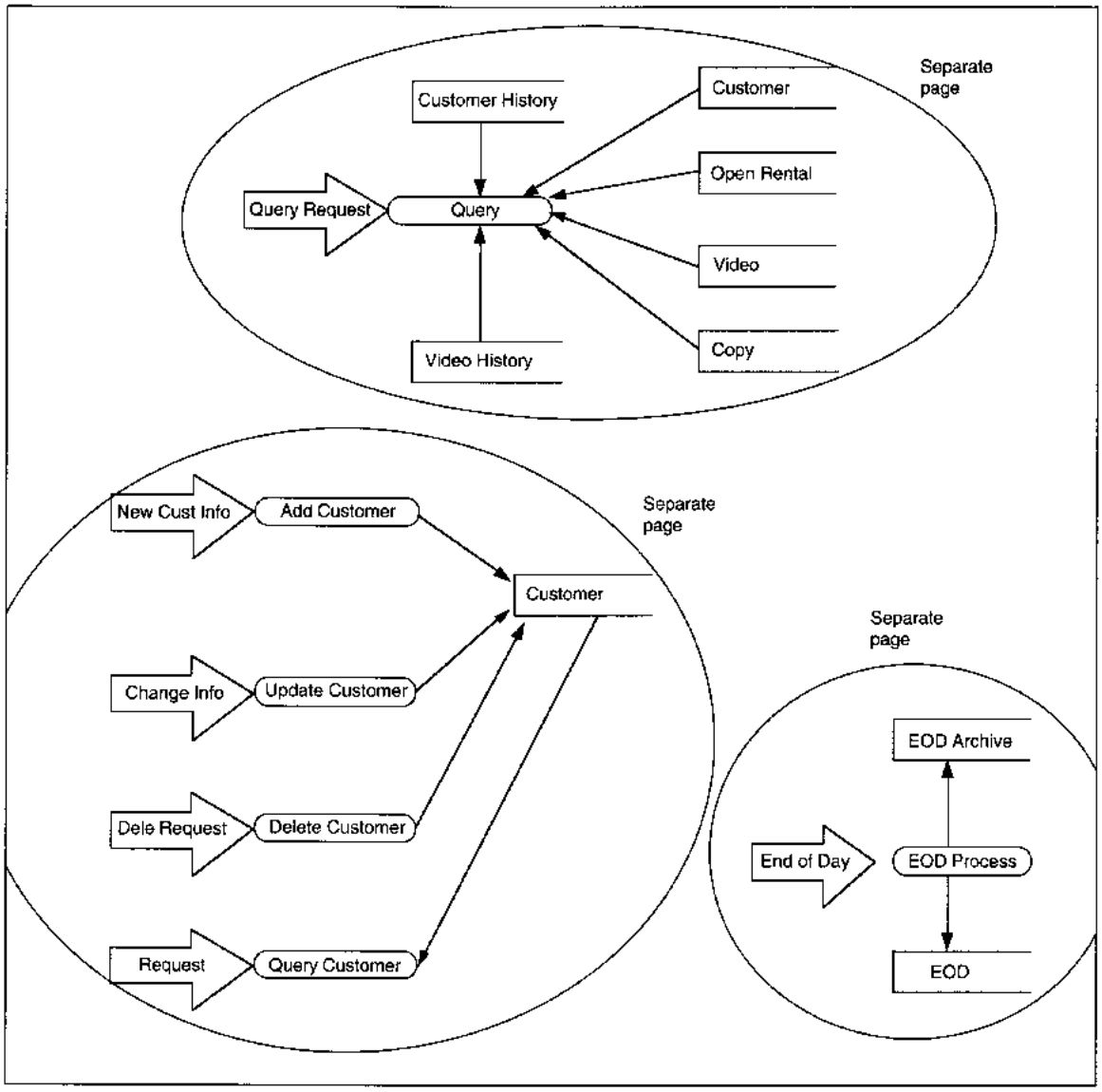


FIGURE 9-52 Combined Process Problem

process for potential deletion from this cluster. If the percentage affinity is between 50–80%, look at the next process that might be related. Do either of the two first processes share more than 80% of their data with the third process? If all three share, cluster them all. If the third strongly relates to the second process but not the first, or the third relates to the first but not the second, still cluster all of them for the moment. Continue to do this type of stepwise comparison of processes using entity affiliation to determine process affinity. Each successive process's functions on the entity are compared to all previous processes, not just the first. If a new process is strongly related to all of them, the cluster remains intact. If the new process strongly relates to some subset of processes, keep it. If the new process

strongly relates to only one process, consider those two processes as a second cluster and set them aside (i.e., create a new interim matrix) with their data for analysis of that cluster. As you complete analysis of a cluster, add it to the new process/entity matrix. Return to the original matrix and draw a line through each process that has been added to a cluster, to ensure that there is no replication of a process and to ensure that processes that are not currently in a cluster are added to a cluster eventually.

As you create the new process/entity matrix, leave several lines and columns of space between each cluster. At the end of the analysis, reanalyze processes that have not been assigned to any cluster. Look at the interim matrices to which each odd process was compared. Find the cluster to which it has the *most* affinity, and add the process to that cluster. If there is no affinity, leave the entity separate.

When affinity analysis is complete, the clustered processes are ready for analysis to determine if they can be in the same execution unit, and the data shared by a cluster should be analyzed to determine the physical design of the database and the needed user views of data. These activities are done during the design phase.

ABC Video Example Entity/Process Matrix

To develop the entity/process matrix, list the lowest level processes from the decomposition diagram down the left column. Then, list the entities across the top, one per column. For each process, add which functions it has for each entity. Possible functions are create, retrieval, update, or delete (CRUD, respectively). The ABC process-entity matrix is shown as first completed in Table 9-5. In completing the table refer back to the PDFD. Use the arrows and names of the processes to identify the type of processing. For entities with arrows only going *into* a process, the correct code is 'R,' for retrieval. For entities with arrows only going *out* of a process, the correct codes are 'C,' 'U,' or 'D,' for create, update, or delete, depending on the processing to take place. For example, *Create Video* has a 'C' under *Copy* and a 'C' under *Video* because it creates both.

First, we check each entity to see if all possible processing is accounted for. *Customer*, *Copy*, and *Video* all have CRUD processing and appear complete. *Open Rental* has CRUD processing during rent/return and R processing as part of *Query*. *Open Rental*, too, looks complete.

EOD Financial Information is created but not ever deleted, which is wrong. It should be archived or deleted at the end of the processing day. We will assume archival at the moment and add both process and data changes back to all other diagrams as needed. We can further assume that the delete process and create process follow. This is an example of what happens with late identified entities. The processing is not as thoroughly analyzed and some information could 'fall through the cracks.' The entity/process matrix helps assure that processing is completely defined.

Finally, history is only created and retrieved. We have not yet defined history, so the decision may not be final. In general, history files *are* only created and retrieved. They are permanent records of past transactions or business states, so they can not be updated or deleted and still be known as 'history.'

When the matrix is complete, we cluster the processes by their entity affinity (see Table 9-6). There are five possible clusters, but most of the data are used by many of the processes. So, the clustering shown is to give an example of how, at the application level, affinity analysis can be used to determine user views for DBMS access. For ABC, because of the extensive retrieval activities, one subject database would be defined. *EOD* data are kept separate from all other data. *Customer*, *Video*, *Copy*, and *Open Rental* data are all used to create individual relations and joint user views.

First we analyze organizational sufficiency as it relates to processes. Is each entity created by one process only? The answer here is yes. Do all processes creating, updating, and deleting an entity report to the same manager? Here the answer is again, yes. So the organization is sufficient.

Next we look at the entities and their usage to determine subject area databases. Again, ABC is a small company, so the data are mostly in one

(Text continues on page 386)

TABLE 9-5 ABC Video Process/Entity Matrix

Entities = Processes	Customer	Video	Copy	Open Rental	Customer History	Video History	Rental Archive	EOD
Get Request	R							
Get Valid Customer	R							
Get Open Rentals		R	R	R				
Add Return Date								
Check for Late Fees								
Get Valid Videos		R	R			R		
Process Payment and Make Change								U
Create Open Rental				C				
Update Open Rental				U				
Update/Create						CU		
History								
Print Receipt								
Query Customer	R			R	R			
Query Rental	R	R	R	R			R	
Query Video		R	R	R		R		
Query History	R	R	R	R	R	R	R	
Query EOD								R
EOD Processing								CRD
Rental Archive Processing				D			C	
Create Customer	C							
Update Customer	U							
Delete Customer	D							
Create Video/Copy		C	C					
Update Video/Copy		U	U					
Delete Video/Copy		D	D					

TABLE 9-6 ABC Video Process/Entity Matrix Affinity Clusters

Entities = Processes	Customer	Video	Copy	Open Rental	Customer History	Video History	Rental Archive	EOD
Create Customer	C							
Delete Customer	D							
Get Valid Customer	R							
Query Customer	R			R	R			
Update Customer	U							
Create Video/Copy		C	C					
Update Video/Copy		U	U					
Delete Video/Copy		D	D					
Get Open Rentals		R	R	R				
Get Valid Videos		R	R			R		
Query History	R	R	R	R	R	R	R	
Query Video		R	R	R		R		
Create Open Rental				C				
Update Open Rental				U				
Query Rental	R	R	R	R			R	
Rental Archive				D			C	
Processing								
Update/Create					CU	CU		
History								
Process Payment and Make Change								U
EOD Processing								CRD
Query EOD								R
Add Return Date								
Check for Late Fees								
Get Request								
Print Receipt								

Data Entities and Final Attributes

Entity	Attributes (Key underlined)
Customer	Customer Phone ID
	Customer Name
	Customer Address
	Customer City
	Customer State
	Customer Zip
	Customer Credit Card Number
	Credit Card Type
	Credit Card Expiration Date
Open Rental	Customer Phone ID
	Video ID
	Copy ID
	Rental Date
	Return Date
	Late Fee Due
	LF Paid
	Rental Fees Due
	RF Pd
	Other Fees Due
	OF Paid
Video	Video ID
	Video Name
	Entry Date
	Rental Rate
Copy	Video ID
	Copy ID
	Date Received
	Status
Customer History	To Be Defined
Video History	To Be Defined
EOD	To Be Defined
Rental Archive	To Be Defined

Activities for Rental/Return and All Processes

Activity	Processes
Rent/Return	Get Request
	Get Valid Customer
	Get Open Rentals
	Add Return Date
	Check for Late Fees
	Get Valid Videos
	Process Payment and Make Change
	Create Open Rental
	Update Open Rental
	Update/Create History
	Print Receipt
Periodic Processing	Query Customer
	Query Rental
	Query Video
	Query History
	Query EOD
	EOD Processing
	Rental Archive Processing
Customer Maintenance	Create Customer
	Update Customer
	Delete Customer
Video Maintenance	Create Video/Copy
	Update Video/Copy
	Delete Video/Copy

Deferred Items for Decision During Design

Item	Decision
Check for Late Fees	Separate or Consolidated with either/or both *Get Open Rentals* *Add Return Date*
History	Is video history updating going to be done to a history file or to the current *Copy* relations? This requires monthly update. The history file requires further decisions about what is on the file.

FIGURE 9-53 Summary Repository Entries

database. *Video*, *Customer*, *Open Rental*, and *End of Day* information will be stored together. *Video* and *Customer* are separate from the other entities because they are only modified by one process. Historical information could be stored in a separate database or set of files. In most companies and applications, history files *are* kept separate from the other databases and files. For applications with huge

amounts of data, history is even on a different storage medium. History is frequently kept on tape and the current databases are on disk. Depending on volume, we would consider tape storage for history here, too.

The entity/process matrix analysis completes the BAA. At this time, all entities, processes, attributes, and their interrelationships should be known based on business requirements. All information is documented in a repository for use in the next phase. The repository entries, without details, are presented in Figure 9-53.

SOFTWARE SUPPORT FOR DATA-ORIENTED ANALYSIS

There are many CASE tools that support data modeling and other data-oriented analysis tasks. Tools that also support Information Engineering are integrated toolsets that cover the complete development life cycle. CASE support for Information Engineering is the best of any methodology. Two CASE environments support the entire IE life cycle from enterprise analysis through maintenance. The two tools are IEF™[8] and ADW. The CASE tools are by no means perfect, however; interphase linkages are weak; numerous bugs plague new releases of ADW; and old releases of IEF were designed so rigidly that all graphical and definitional forms were required to use the tool effectively. The positive aspect of both tools is that they can feed code generators that can automate development of as much as 70% of the necessary program code. Both ADW and IEF support all of the graphical and definitional forms discussed in this chapter. The list in Table 9-7 includes many other CASE tools that support some, but not all, graphical, documentation, or mental models of IE and data-oriented analysis.

8 IEF™ is a trademark of the Texas Instruments Co., Dallas, TX. ADW™ is a trademark of Knowledgeware, Inc., Atlanta, GA.

SUMMARY

Data-oriented methodologies are based on the notion that data are more stable than processes in business. Organizations and procedures change regularly; the data on which they work does not. Data-oriented methodologies, then, concentrate on data as the initial focus of study. The theory underlying data methods applies semantic modeling to data and system theory to business functions.

Information Engineering's business area analysis (BAA) is the example of data-oriented methodology described here. BAAs begin with an entity-relationship diagram that is fully identified and normalized. Business functions are decomposed to create process decomposition, process dependency, and process data flow diagrams.

Business processes from the decomposition are coupled to the entities from the ERD to form an entity/process matrix, also called a CRUD matrix (for create-retrieve-update-delete). The CRUD matrix defines responsibility for actions on each entity for each process. Affinity analysis of the CRUD matrix clusters processes and data into groups. The affinity groupings are used to decide the need for additional project scoping, future applications, and alternatives for subject database design. All information from the BAA is documented in a repository.

REFERENCES

Date, C. J., *An Introduction to Database Systems*, Vol. 1, 5th edition. Reading, MA: Addison-Wesley, 1990.

Finkelstein, Clive, *An Introduction to Information Engineering: From Strategic Planning to Information Systems*. Reading, MA: Addison-Wesley, 1989.

Knowledgeware, Inc., *Information Engineering Workbench™/Analysis Workstation, ESP Release 4.0*. Atlanta, GA: Knowledgeware, Inc., 1987.

Loucopoulos, Pericles, and Roberto Zicari, *Conceptual Modeling, Databases and CASE: An Integrated View of IS Development*. NY: John Wiley & Sons, 1992.

Martin, James, *Information Engineering: Book 2, Planning and Analysis*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1991.

Martin, James, and Carma McClure, *Diagramming Techniques for Analysts and Programmers*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1985.

TABLE 9-7 Data-Oriented Analysis CASE Support

Product	Company	Technique
Analyst/Designer Toolkit	Yourdon, Inc. New York, NY	Entity-Relationship Diagram (ERD)
Anatool	Advanced Logical SW Beverly Hills, CA	ERD
Bachman	Bachman Info Systems Cambridge, MA	Bachman ERD
CorVision	Cortex Corp. Waltham, MA	ERD
Deft	Deft Ontario, Canada	ERD
Design/1	Arthur Anderson, Inc. Chicago, IL	ERD
ER-Designer	Chen & Assoc. Baton Rouge, LA	ERD
Excelerator	Index Tech. Cambridge, MA	ERD
IEF	Texas Instruments Dallas, TX	Functional Decomposition ERD Entity Hierarchy Process Hierarchy Process Dependency Process Data Flow Diagram Entity/Process Matrix
IEW, ADW	Knowledgeware Atlanta, GA	Functional Decomposition ERD Entity/Process Matrix

Texas Instruments, *A Guide to Information Engineering Using the IEF*. Dallas, TX: Texas Instruments, 1988.

KEY TERMS

activity
affinity
affinity analysis
architectures
associative entity
attribute
attributive entity
business area analysis (BAA)
business function
business process
business redesign
cardinality
CRUD matrix
data administration
data trigger
direct method of normalization
elementary process
entity
entity-relationship diagram (ERD)
entity type
entity/process matrix
entity structure analysis
event trigger
functional decomposition
fundamental entity
instance
many-to-many relationship
normalization
one-to-many relationship
one-to-one relationship
optional relationship
process data flow diagram (PDFD)
process data trigger
process dependency diagram
process relationship
relational database theory
relationship
relationship entity
required relationship
subject area database
tabular method of normalization
trigger

TABLE 9-7 Data-Oriented Analysis CASE Support (*Continued*)

Product	Company	Technique
Maestro	SoftLab San Francisco, CA	ERD
Multi-Cam	AGS Mgmt Systems King of Prussia, PA	ERD
ProKit Workbench	McDonnell Douglas St. Louis, MO	ERD
SW Thru Pictures	Interactive Development Environments San Francisco, CA	ERD
System Engineer	LBMS Houston, TX	ERD Entity Life History Diagram
Teamwork	CADRE Tech., Inc. Providence, RI	ERD
Telon	Pansophic Systems, Inc. Lisle, IL	ERD
The Developer	ASYST Technology, Inc. Naperville, IL	ERD Organization Chart Operations Process Diagram Matrix Diagram
vs Designer	Visual Software Inc Santa Clara, CA	Process Flow Diagram

EXERCISES

1. Complete the PDFD for Video Maintenance.
2. The *Get Valid Video* process has as its subprocesses: *Get Video Data*, *Create Video File*, and feeds into the *Check Previous Rental* process. Do a process dependency diagram for these subprocesses. Then add event triggers and data files to complete the PDFD.

STUDY QUESTIONS

1. Define the following terms:
   affinity
   attributive entity
   associative entity
   CRUD matrix
   elementary process
   entity
   entity-relationship diagram
   functional decomposition
   m:n relationship
   normalization
   possible *number* of entity relationships
   possible *nature* of entity relationships
   promoted relationship
   trigger
2. Compare data flow diagrams from Chapter 7 to process data flow diagrams in this chapter. List five similarities and five differences between them.
3. Find a small company and develop an entity-relationship diagram of their data. For each

entity, develop an attribute list and normalize the data. Discuss the problems you have in developing the answer with your class.

4. What is a 'promoted relationship' in an ERD and what is the result of the promotion?
5. Normalization assumes that you know the relationships of data within and between entities. What happens if you do not have the data relationships correctly specified in normalization?
6. What does normalization, as performed during analysis, define? What does it *not* define?
7. What is the purpose of an entity/process matrix?
8. Describe the analysis of an entity/process matrix.
9. What is the significance of subject area databases? What do subject area databases have in common with normalization?
10. What is the importance of an organizational ERD? What problems might arise when you begin the ERD definition for an application during the business area analysis?
11. Describe the relationships between the diagrams developed throughout IE-BAA. That is, how is each diagram used in the creation of successor diagrams?
12. What is the purpose of functional decomposition?
13. What are the three conditions under which you cannot eliminate an entity?
14. On a process dependency diagram, what is the significance of directed lines connecting two processes? Does this meaning change when the processes are connected on a PDFD? If so, how?
15. When should printed items be included on a PDFD?
16. What is a functional decomposition in IE? Define the diagram and the contents at each level of detail. How do you know when the decomposition is complete, i.e., when to stop?
17. What are the steps to developing a PDFD?
18. Define the allowable inputs to a process on a PDFD.
19. What is a CRUD matrix? How is it used?
20. What are the allowable connections on a process dependency diagram?
21. What are the allowable connections on a process data flow diagram?
22. List four problems and their solutions when developing a PDFD.

★ EXTRA-CREDIT QUESTIONS

1. Develop an IE analysis for the accounting (or purchasing) function at ABC Video. Refer to other books to obtain details about accounting applications. One such book is *Online Business Computer Applications*, 3rd edition, Alan L. Eliason, NY: MacMillan, 1991.
2. Compare IE to process analysis. What are the similarities? What are the differences? How are the same terms used differently? Which method has the least ambiguity? Which method results in a more complete analysis?
3. Do an entity-relationship diagram for the AOS Tracking System problem in the Appendix. Normalize the data. Compile a list of issues for future resolution dealing with the data. The issues should relate to how many relations are needed, how the data will be used, and how to minimize the number of relations without having many unused attributes in each relation.
4. Do a process decomposition diagram and a PDFD for the AOS Tracking System described in the Appendix.
5. What do you *not* know after BAA is complete?