

**PROCESS FOR
SYSTEM ARCHITECTURE
AND REQUIREMENTS
ENGINEERING**

Also Available from DORSET HOUSE PUBLISHING Co.

Adaptive Software Development:

A Collaborative Approach to Managing Complex Systems

by James A. Highsmith III foreword by Ken Orr

ISBN: 0-932633-40-4 Copyright ©2000 392 pages, softcover

Complete Systems Analysis: The Workbook, the Textbook, the Answers

by James & Suzanne Robertson foreword by Tom DeMarco

ISBN: 0-932633-50-1 Copyright ©1998,1994 624 pages, softcover

Designing Quality Databases with IDEF1X Information Models

by Thomas A. Bruce foreword by John A. Zachman

ISBN: 0-932633-18-8 Copyright ©1992 584 pages, hardcover

Exploring Requirements: Quality Before Design

by Donald C. Gause and Gerald M. Weinberg

ISBN: 0-932633-13-7 Copyright ©1989 320 pages, hardcover

The Practical Guide to Business Process Reengineering Using IDEF0

by Clarence G. Feldmann foreword by John V. Tieso

ISBN: 0-932633-37-4 Copyright ©1998 240 pages, softcover

The Psychology of Computer Programming: Silver Anniversary Edition

by Gerald M. Weinberg ISBN: 0-932633-42-0 Copyright ©1998 360 pages, softcover

Quality Software Management, Vol. 4: Anticipating Change

by Gerald M. Weinberg

ISBN: 0-932633-32-3 Copyright ©1997 504 pages, hardcover

Surviving the Top Ten Challenges of Software Testing: A People-Oriented Approach

by William E. Perry and Randall W. Rice

ISBN: 0-932633-38-2 Copyright ©1997 216 pages, softcover

Strategies for Real-Time System Specification

by Derek J. Hatley and Imtiaz A. Pirbhai foreword by Tom DeMarco

ISBN: 0-932633-11-0 Copyright ©1988, 1987 408 pages, hardcover

Find Out More about These and Other DH Books:

Contact us to request a Book & Video Catalog and a free issue of *The Dorset House Quarterly*, or to confirm price and shipping information.

DORSET HOUSE PUBLISHING CO., INC.

353 West 12th Street New York, NY 10014 USA

1-800-DH-BOOKS (1-800-342-6657) 212-620-4053 fax: 212-727-1044

info@dorsethouse.com <http://www.dorsethouse.com>

**PROCESS FOR
SYSTEM ARCHITECTURE
AND REQUIREMENTS
ENGINEERING**

Derek Hatley Peter Hruschka Imtiaz Pirbhai



Dorset House Publishing
353 West 12th Street
New York, NY 10014

Library of Congress Cataloging-in-Publication Data

Hatley, Derek J., 1934-

Process for system architecture and requirements engineering / Derek Hatley, Peter Hruschka, Imtiaz Pirbhai.

p. cm.

ISBN 0-932633-41-2 (pbk.)

1. System design. 2. System analysis. I. Hruschka, Peter, 1951- II. Pirbhai, Imtiaz A., 1953- III. Title.

QA76.9.S88 H3735 2000

005.1'2--dc21

00-060997

All product and service names appearing herein are trademarks or registered trademarks or service marks or registered service marks of their respective owners and should be treated as such.

Cover Design: David W. McClintock

Cover Graphic: Detail from Figure 11.2

Cover Author Photographs: Hatley photo: O'Connor-Rice Studios; Hruschka photo: James Robertson; Pirbhai photo: Stewart Auer.

Copyright © 2000 by Derek J. Hatley, Peter Hruschka, and Shams Pirbhai. Published by Dorset House Publishing Co., Inc., 353 West 12th Street, New York, NY 10014.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher.

Distributed in the English language in Singapore, the Philippines, and Southeast Asia by Alkem Company (S) Pte. Ltd., Singapore; in the English language in India, Bangladesh, Sri Lanka, Nepal, and Mauritius by Prism Books Pvt., Ltd., Bangalore, India; and in the English language in Japan by Toppan Co., Ltd., Tokyo, Japan.

Printed in the United States of America

Library of Congress Catalog Number: 00-060997

ISBN: 0-932633-41-2

12 11 10 9 8 7 6 5 4 3 2 1

DEDICATION

A wonderful friend and fine colleague of ours started this book: He laid out the basic ideas and the structure, and developed some of the materials. Then, suddenly and unexpectedly, and in the prime of his life, he passed away. We miss him sorely.

We are grateful to his brother Shams for providing us with the original materials, and honored at being given the opportunity to complete the project. It is not the same book it would have been had he completed it, but we hope it comes somewhere close to his expectations.

It is unusual to dedicate a book to one of its authors, but these are unusual and tragic circumstances. So, we humbly dedicate Process for System Architecture and Requirements Engineering to the memory of a true visionary in the field of system development:

Imtiaz Pirbhai.

ACKNOWLEDGMENTS

We are deeply indebted to Kamal Hammoutene, Alan Hecht, Mark Maier, Vince Peterson, and Tanehiro Tatsuta for their painstaking reviews of the draft of this book and for their excellent comments and suggestions. The book is far better for their efforts.

We thank the editorial staff at Dorset House Publishing—Nuno Andrade, Debbie Carter, David Crohn, Wendy Eakin, Bob Hay, Mike Lumelsky, David McClintock, Matt McDonald, and Mike Richter—for the excruciating detail with which they examined, tore apart, and reconstructed our English. This, too, greatly improved the final product.

Finally, we thank the many participants in our seminars and workshops for their enthusiasm and for teaching us at least as much as we taught them.

Contents

Figures xv

Part I Concepts 1

Chapter 1 Introduction 3

- 1.1 PURPOSE AND SCOPE 3
- 1.2 THE SYSTEM DEVELOPMENT PROCESS 4
- 1.3 UNDERLYING PRINCIPLES 4
- 1.4 WHAT'S IN A NAME? 5
- 1.5 AUDIENCE FOR AND STRUCTURE OF THE BOOK 6
- 1.6 A PARTICIPATIVE CASE STUDY ON THE WEB 7
- 1.7 A CAVEAT 8

Chapter 2 What Is a System? 9

- 2.1 SYSTEM CHARACTERISTICS 9
 - 2.1.1 Introduction to Systems 9
 - 2.1.2 System Hierarchies 11
 - 2.1.3 Multiple Hierarchies 14
 - 2.1.4 System Networks 14
 - 2.1.5 System Life Cycle and Errors 15
 - 2.1.6 Order and Chaos 16
 - 2.1.7 System Predictability 17
 - 2.1.8 Dealing with Complexity 17
- 2.2 VIEWS OF A SYSTEM 18
 - 2.2.1 The Processing View 19
 - 2.2.2 The Processor View 20
 - 2.2.3 The *What/How* View 21

- 2.2.4 The Level of Intelligence View 22
- 2.2.5 The Static/Dynamic View 23
- 2.3 SYSTEM REQUIREMENTS 24
 - 2.3.1 The Sources of Requirements 24
 - 2.3.1.1 Customers 24
 - 2.3.1.2 Users 25
 - 2.3.1.3 Managers 25
 - 2.3.1.4 Industry Standards 25
 - 2.3.1.5 The Development Process 26
 - 2.3.1.6 Others 26
 - 2.3.2 What Exactly Are Requirements, Anyway? 26
 - 2.3.3 A Model of Requirements 28
 - 2.3.4 Quality 36
 - 2.3.5 Requirement Management and Analysis 37
 - 2.3.5.1 Requirement Gathering 37
 - 2.3.5.2 Requirement Integrity Analysis 38
 - 2.3.5.3 Requirement Feasibility Analysis 38
 - 2.3.5.4 Requirement Detailing 38
 - 2.3.5.5 Requirement Deriving 39
 - 2.3.5.6 Requirement Categorizing 39
- 2.4 SYSTEM SUMMARY 40

Chapter 3 A Framework for Modeling Systems 41

- 3.1 A MODEL FRAMEWORK 41
- 3.2 MODELS IN GENERAL 41
 - 3.2.1 Models Are Useful Abstractions 42
 - 3.2.2 Model Representations and Reuse 43
- 3.3 EXPLOITING SYSTEM HIERARCHIES 45
 - 3.3.1 Why Exploit Hierarchies? 46
 - 3.3.2 What Are the Benefits and Pitfalls of Layered Systems? 46
 - 3.3.3 How Many Models? 47
 - 3.3.4 Where Do We Stop? 48
- 3.4 EXPLOITING THE *WHAT/HOW* CLASSIFICATION 49
 - 3.4.1 Separation of *What* and *How* 50
 - 3.4.2 The Architecture Template 51
 - 3.4.3 Using the Architecture Template 53
- 3.5 EXPLOITING THE *INFORMATION/MATERIAL/ENERGY* CLASSIFICATION 55
 - 3.5.1 A Generic Subsystem Structure 55
 - 3.5.2 Categories of a Deliverable System 57
 - 3.5.3 Categories of a People System 60
- 3.6 LAYERED MODELS: THE TRUTH AT LAST! 60
 - 3.6.1 Aggregation/Decomposition Relationship in Models 64
 - 3.6.2 Abstraction/Detailing Relationship in Models 65

- 3.6.3 Supertype/Subtype Relationship in Models 66
- 3.6.4 Controlling/Controlled Relationship in Models 67
- 3.6.5 Layered Models Summary 68

3.7 MODEL FRAMEWORK SUMMARY 70

Chapter 4 System Development Models 72

- 4.1 OVERVIEW 72
- 4.2 ARCHITECTURE MODEL 74
 - 4.2.1 Introduction 74
 - 4.2.2 Basic Modeling Elements 76
 - 4.2.2.1 Architecture Module 77
 - 4.2.2.2 Terminator 78
 - 4.2.2.3 Architecture Flow 79
 - 4.2.2.4 Message 81
 - 4.2.2.5 Flows and Messages 83
 - 4.2.2.6 Inheritance Relationship 85
 - 4.2.2.7 Architecture Interconnect 86
 - 4.2.3 Context Diagram 87
 - 4.2.4 Networks and Hierarchies 90
 - 4.2.5 Architecture Communication Model 92
 - 4.2.5.1 Architecture Flow Diagram 94
 - 4.2.5.2 Architecture Message Diagram—Network Style 97
 - 4.2.5.3 Architecture Message Diagram—Hierarchy Style 99
 - 4.2.5.4 Architecture Module Specification 101
 - 4.2.5.5 Message Specification 102
 - 4.2.6 Architecture Interconnect Model 103
 - 4.2.6.1 Architecture Interconnect Diagram 104
 - 4.2.6.2 Architecture Interconnect Specification 106
 - 4.2.7 Architecture Inheritance Model 106
 - 4.2.7.1 Module Inheritance Diagram 107
 - 4.2.8 Architecture Dictionary 108
 - 4.2.9 Architecture Model Balancing 110
 - 4.2.9.1 Architecture Message Diagram Balancing 111
- 4.3 REQUIREMENTS MODEL 112
 - 4.3.1 Introduction 112
 - 4.3.2 Entity Model 115
 - 4.3.2.1 Basic Modeling Elements 116
 - 4.3.2.2 Attribute 116
 - 4.3.2.3 Entity Class 116
 - 4.3.2.4 Relationship 118
 - 4.3.2.5 Special Relationships 121
 - 4.3.2.6 Class Diagram 125
 - 4.3.2.7 Entity Class Specification 126
 - 4.3.2.8 Relationship Specification 127
 - 4.3.2.9 Attribute Specification 129
 - 4.3.3 Process Model 130

- 4.3.3.1 *Basic Modeling Elements* 131
- 4.3.3.2 *Data Flow* 131
- 4.3.3.3 *Process* 133
- 4.3.3.4 *Store* 134
- 4.3.3.5 *Terminator* 135
- 4.3.3.6 *Data Context Diagram* 135
- 4.3.3.7 *Data Flow Diagram* 137
- 4.3.3.8 *Detailing Diagrams* 139
- 4.3.3.9 *Process Specification* 141
- 4.3.4 *Control Model* 143
 - 4.3.4.1 *Control Flow* 144
 - 4.3.4.2 *Control Specification* 146
 - 4.3.4.3 *Sequential Machines—State Transition Diagrams and State Charts* 147
 - 4.3.4.4 *Other Representations of Sequential Machines* 151
 - 4.3.4.5 *Combinational Machines—Decision Tables and Process Activation Tables* 153
- 4.3.5 *The Control Flow Model: Basic Elements* 155
 - 4.3.5.1 *Control Flow* 155
 - 4.3.5.2 *CSPEC Bar* 156
- 4.3.6 *Control Context Diagram* 156
- 4.3.7 *Control Flow Diagram* 157
 - 4.3.7.1 *Rules and Guidelines for CFDs and CSPECs* 159
- 4.3.8 *Separation of Data and Control* 160
- 4.4 *REQUIREMENTS DICTIONARY* 162
 - 4.4.1 *Timing Requirements* 164
 - 4.4.2 *Requirements Model Balancing* 166
 - 4.4.3 *Architecture Template and Enhanced Requirements Model* 166
 - 4.4.4 *Requirements Model Summary* 169
- 4.5 *REQUIREMENTS/ARCHITECTURE RELATIONSHIPS* 170
 - 4.5.1 *Scope Differences* 171
 - 4.5.2 *Superbubbles* 172
 - 4.5.3 *Traceability* 176
 - 4.5.4 *Architecture Model/Requirements Model Balancing* 177
- 4.6 *A NOTE ON OBJECT ORIENTATION* 178
- 4.7 *SYSTEM MODELS SUMMARY AND FURTHER READING* 180

Chapter 5 The System Development Process 181

- 5.1 *PROCESS, METHODS, AND TOOLS* 181
- 5.2 *THE NATURE OF THE DEVELOPMENT PROCESS* 183
 - 5.2.1 *The Evolution of the Development Process* 183
 - 5.2.2 *The Concurrent Development Process* 184
 - 5.2.3 *The Meaning of Concurrent Development* 187
- 5.3 *THE PROCESS AND THE METHODS* 190

- 5.3.1 System Specification Models 191
- 5.3.2 Requirement Enhancement and Allocation 194
- 5.3.3 Requirement Deriving and Detailing 194
- 5.3.4 Traceability 197
- 5.4 ROLES OF THE SYSTEM ARCHITECT AND SYSTEM ENGINEER 200
 - 5.4.1 Requirement Management 202
 - 5.4.2 Feasibility Analyses, Trade-Off Studies, and Prototypes 202
 - 5.4.3 Project Coordination 203
 - 5.4.4 Manual Procedures and Other Operator Functions 203
 - 5.4.5 Companion IR&D Projects 204
- 5.5 SYSTEM DEVELOPMENT PROCESS SUMMARY 204

Chapter 6 Applying the Models to Development 205

- 6.1 OVERVIEW 205
- 6.2 UNDERSTANDING THE GENERIC DEVELOPMENT STRUCTURE 206
- 6.3 EXAMPLE: A PATIENT-MONITORING SYSTEM 209
 - 6.3.1 Problem Statement: Nurses' Tasks 210
 - 6.3.2 Modeling the Environment 210
 - 6.3.3 Building the Context-Level Model 213
 - 6.3.4 Technology Constraints for the Patient-Monitoring System 219
 - 6.3.5 Creating the Enhanced Requirements Model 220
 - 6.3.6 Building the Architecture Level 1 Model 224
 - 6.3.6.1 *Architecture Model Allocation* 224
 - 6.3.7 Interconnects and Further Enhancements 228
 - 6.3.8 Completing the Architecture 229
 - 6.3.9 Developing the Lower-Level Models 233
- 6.4 CONFIGURING SOFTWARE AND COMPUTER HARDWARE 236
 - 6.4.1 Single-Hardware/Multiple-Software Configuration 239
 - 6.4.2 Multiple-Hardware/Single-Software Configuration 241
- 6.5 MODELING THE NUMEROUS HARDWARE TECHNOLOGIES 244
 - 6.5.1 Electrical 245
 - 6.5.2 Electronic 247
 - 6.5.3 Electromechanical 247
 - 6.5.4 Mechanical 248
 - 6.5.5 Hydraulic and Pneumatic 248
 - 6.5.6 Optical 249
 - 6.5.7 Chemical 249
 - 6.5.8 Manufacturing 250
 - 6.5.9 Detailed Hardware Design 250
 - 6.5.10 Mixed Technologies 250
- 6.6 COMPUTER HARDWARE LAYERS 251
- 6.7 SOFTWARE LAYERS 253
 - 6.7.1 Structured Design 254

- 6.7.2 CODARTS 255
- 6.7.3 Object Orientation 256
- 6.7.4 Lessons from the History of Software 258
- 6.7.5 Software Categories 259
- 6.7.6 Software Architectures 268

6.8 SUMMARIES 273

- 6.8.1 System Summary 273
- 6.8.2 Software Summary 274

Chapter 7 System Development Overview Using a Meta-Model 275

- 7.1 INTRODUCTION 275
- 7.2 A META-MODEL FOR DEVELOPMENT PROJECTS 276
- 7.3 AN ESSENTIAL MODEL OF THE DEVELOPMENT PROCESS 277
- 7.4 THE ENHANCED DEVELOPMENT MODEL 281
- 7.5 THE DEVELOPMENT ARCHITECTURE CONTEXT 284
- 7.6 DEVELOPMENT PROCESS ARCHITECTURE 288
- 7.7 DEVELOPMENT PROCESS TASK ALLOCATION 291
- 7.8 VARIATIONS ON THE ARCHITECTURE TEMPLATE 291

Part II Case Study: Groundwater Analysis System 297

Chapter 8 Initial Problem Statement 299

- 8.1 OVERVIEW 299
- 8.2 REQUIRED CAPABILITIES 300
- 8.3 REQUIRED PERFORMANCE 301
- 8.4 REQUIRED CONSTRAINTS 301
 - 8.4.1 Input/Output Constraints 301
 - 8.4.2 Design Constraints 301
 - 8.4.2.1 *Other Design Constraints* 303

Chapter 9 Modeling the Known Pieces 304

- 9.1 OVERVIEW 304
- 9.2 THE REQUIREMENTS CONTEXT 304
- 9.3 THE SYSTEM TIMING SPECIFICATION 305
- 9.4 THE ENTITY MODEL 307
- 9.5 THE EXISTING SAMPLING MODULE 311

Chapter 10 Building Upon the Known Pieces 316

- 10.1 TOP-LEVEL ESSENTIAL MODEL 316
- 10.2 ENHANCING THE ESSENTIAL MODEL 331
 - 10.2.1 User-Interface Processing 331

- 10.2.2 Input and Output Processing 331
- 10.2.3 Maintenance and Support Functions 334
- 10.2.4 The Enhanced Requirements Models 334

10.3 ARCHITECTURE CONTEXT 341

10.4 BUILDING UP FROM THE EXISTING SAMPLING MODULE 343

- 10.4.1 An Intermediate Module 344
- 10.4.2 Central versus Distributed Processing 344
- 10.4.3 Sample Analyzer Requirements 345

10.5 WHAT DO WE HAVE, AND WHAT IS MISSING? 346

Chapter 11 Filling In the Blanks 348

11.1 INTRODUCTION 348

11.2 ARCHITECTURE MODULES 348

11.3 ALLOCATING THE ENHANCED REQUIREMENTS MODEL 349

11.4 ENHANCING THE ALLOCATED MODELS 357

11.5 ADDING THE ARCHITECTURE FLOWS AND INTERCONNECTS 362

11.6 FLOW-TO-INTERCONNECT ALLOCATIONS 362

11.7 MERGING THE TOP-DOWN AND BOTTOM-UP PIECES 364

Chapter 12 Completing the Models 375

12.1 INTRODUCTION 375

12.2 ACCURACY ALLOCATION 375

12.3 TIMING ALLOCATION—CONCURRENT ARCHITECTURE MODULES 376

12.4 ARCHITECTURE MODULE SPECIFICATIONS 378

12.4.1 Architecture Module Specification: Groundwater Analysis System 378

12.4.2 Architecture Module Specification: Sample Analyzer 380

12.5 ARCHITECTURE INTERCONNECT SPECIFICATIONS 382

12.5.1 Architecture Interconnect Specification: Local Bus 382

12.6 REQUIREMENTS AND ARCHITECTURE DICTIONARIES 383

Chapter 13 Groundwater Analysis System Summary 387

13.1 OVERVIEW 387

Appendix Changes, Improvements, and Misconceptions Since the Methods' Introduction 389

A.1 A LEARNING EXPERIENCE 389

A.2 CHANGES AND IMPROVEMENTS 390

A.2.1 Superbubbles 390

A.2.2 Addition of Object-Oriented Constructs 391

A.2.3 The Total, Multileveled Methods Structure 391

A.2.4 Architecture Interconnect Context Diagram 391

- A.2.5 Entity Model 392
- A.2.6 Hardware/Software Interface 392
- A.2.7 Functions of Time in Data and Control Processes 392
- A.2.8 Extended Traceability 393
- A.2.9 Derived Requirements 393
- A.2.10 Special Cases of Architecture Flows and Interconnects 393
- A.2.11 The Architecture Template as a Meta-Model 394
- A.2.12 PSPEC Guidelines 394
- A.2.13 Extended Guidelines on Separation of Data and Control 394
- A.2.14 Extended Guidelines for Architecture Module and Interconnect Specifications 395
- A.2.15 Data Flows from CSPECs 395
- A.2.16 Use of State Charts in CSPECs 396
- A.2.17 Elimination of Sequential and One-Shot Decision Tables and Process Activation Tables 396
- A.2.18 Manual (Human) Subsystems 396
- A.2.19 Primitive Process Notation 396
- A.2.20 Stores Shown on Multiple Levels 397
- A.2.21 Descriptions of Data Flow Diagrams 397
- A.3 MISCONCEPTIONS ABOUT THE METHODS 397
 - A.3.1 The Methods Are Mostly for Real-Time and Embedded Applications 397
 - A.3.2 The Requirements Method Is More Important Than the Architecture Method 398
 - A.3.3 The Most Significant Feature of the Requirements Model Is the Control Model 398
 - A.3.4 Everything Involving Control Must Use the Control Model 398
 - A.3.5 Everything Involved in Control Must Involve Control Flows 399
 - A.3.6 All Discrete-Valued Flows Must Be Control Flows 399
 - A.3.7 The Methods Employ a Strictly Top-Down Decomposition Approach 399
 - A.3.8 The Methods Are Little Different from Basic Structured Analysis 399
 - A.3.9 The Methods Mostly Apply to Software 399
 - A.3.10 The Methods Are Incompatible with Object-Orientation 400
 - A.3.11 PSPECs Must Contain Sufficient Detail for Detailed Design 400
 - A.3.12 The Models Should Be Executable 400

Glossary 401

Bibliography 419

Index 425

Figures

- Figure 2.1: Universal Hierarchy of Systems 13*
- Figure 2.2: Avionics Context and Content Hierarchy 13*
- Figure 2.3: A Subsystem Network 15*
- Figure 2.4: System Errors 15*
- Figure 2.5: Project Chaos and Order 16*
- Figure 2.6: System Processing 19*
- Figure 2.7: Will the Real System Please Stand Up! 21*
- Figure 2.8: System Classification 23*
- Figure 2.9: A Class Diagram of Requirements 29*
-
- Figure 3.1: The Role of a Model 42*
- Figure 3.2: The Bathtub Model 44*
- Figure 3.3: Equations Representing the Bathtub Model 44*
- Figure 3.4: A Data Flow Diagram Representing the Bathtub Model 45*
- Figure 3.5: A Context Diagram of the Model in Figure 3.4 45*
- Figure 3.6: Hierarchies to Reduce Complexity 46*
- Figure 3.7: Partitioning a System 49*
- Figure 3.8: The Architecture Template 51*
- Figure 3.9: The Template Bridging Requirements and Architecture Models 53*
- Figure 3.10: Classification of Activities in a Manual System 54*
- Figure 3.11: Classification of Activities in an Automated System 55*

- Figure 3.12: *Separating Material/Energy Processing from Information Processing* 56
- Figure 3.13: *Generic Description of Subsystem Responsibilities* 57
- Figure 3.14: *A Template for Subsystems of a Deliverable, Human-Operated System* 58
- Figure 3.15: *Subsystems of a Cruise Control System* 59
- Figure 3.16: *A Template for an Organization That Builds Deliverable Systems* 61
- Figure 3.17: *Subsystems of a Garment Factory* 62
- Figure 3.18: *Summary of Relationships in Layered Models* 69
- Figure 3.19: *The Modeling Framework* 71
-
- Figure 4.1: *A Summary of the Development Models* 73
- Figure 4.2: *Overview of the Architecture Model* 75
- Figure 4.3: *Various Architecture Modules* 77
- Figure 4.4: *Architecture Module Symbols* 78
- Figure 4.5: *Terminator Symbols* 79
- Figure 4.6: *Architecture Flow Symbols* 79
- Figure 4.7: *Message Symbol* 81
- Figure 4.8: *Flows and Messages* 84
- Figure 4.9: *Inheritance Symbols* 86
- Figure 4.10: *Interconnect Symbols* 87
- Figure 4.11: *Architecture Flow Context Diagram* 88
- Figure 4.12: *Architecture Message Context Diagram* 89
- Figure 4.13: *Architecture Interconnect Context Diagram* 90
- Figure 4.14: *A Class Diagram of Architecture Diagrams* 91
- Figure 4.15: *Communication Hierarchies, Networks, Layers, and Partitions* 93
- Figure 4.16: *Architecture Flow Diagram* 94
- Figure 4.17: *Architecture Flow Diagram with Push and Pull Indicators* 95
- Figure 4.18: *Child Architecture Flow Diagram* 96
- Figure 4.19: *Architecture Message Diagram* 97
- Figure 4.20: *Architecture Message Diagram—Hierarchy Style* 100
- Figure 4.21: *Interconnect Hierarchies, Networks, Layers, and Partitions* 104
- Figure 4.22: *Architecture Interconnect Diagram* 104
- Figure 4.23: *Child Interconnect Diagram* 105
- Figure 4.24: *Inheritance Model* 107
- Figure 4.25: *Module Inheritance Diagram* 108
- Figure 4.26: *Architecture Dictionary* 109
- Figure 4.27: *Requirements Model Overview* 113
- Figure 4.28: *Chemical Reaction Analogy* 114
- Figure 4.29: *Entity Model Overview* 115

- Figure 4.30: *Entity-Class Symbols* 118
- Figure 4.31: *A Binary Relationship* 119
- Figure 4.32: *A Ternary Relationship and a Recursive Relationship* 119
- Figure 4.33: *Superclass/Subclass Relationship* 121
- Figure 4.34: *Different Kinds of Subclasses* 122
- Figure 4.35: *Aggregation Relationship* 123
- Figure 4.36: *Associative Entity Classes* 125
- Figure 4.37: *A Class Diagram* 126
- Figure 4.38: *Process Model Overview* 130
- Figure 4.39: *Data Flow Symbol* 132
- Figure 4.40: *Splitting a Flow* 132
- Figure 4.41: *Process Symbol* 133
- Figure 4.42: *Store Symbols* 134
- Figure 4.43: *Data Context Diagram* 136
- Figure 4.44: *Data Flow Diagram* 137
- Figure 4.45: *Process Detailing and Description* 140
- Figure 4.46: *Generic PSPEC Format* 142
- Figure 4.47: *Different Styles for PSPECs* 143
- Figure 4.48: *Requirements Model with Control Model Highlighted* 145
- Figure 4.49: *Classification of CSPEC Components* 146
- Figure 4.50: *Typical Combinational CSPEC* 147
- Figure 4.51: *Typical Sequential CSPEC* 147
- Figure 4.52: *A Generic State Chart* 148
- Figure 4.53: *Alternative Representation for State Models* 152
- Figure 4.54: *Generic Form of a Decision Table* 153
- Figure 4.55: *A CSPEC Combining Decision Table and Activation Table* 154
- Figure 4.56: *Control Flow Symbol* 155
- Figure 4.57: *A CSPEC Bar with Incoming and Outgoing Control Flows* 156
- Figure 4.58: *Control Context Diagram* 157
- Figure 4.59: *Control Flow Diagram* 158
- Figure 4.60: *Data Conditions* 159
- Figure 4.61: *Hints on Distinguishing Data and Control Flows* 161
- Figure 4.62: *Hints on Distinguishing Processes and Their Control* 161
- Figure 4.63: *Notation for Structuring Dictionary Entries* 163
- Figure 4.64: *Properties of Basic Dictionary Entries* 163
- Figure 4.65: *Architecture Template* 167
- Figure 4.66: *Logical and Physical Interfaces* 168
- Figure 4.67: *Requirements Model Summary* 170
- Figure 4.68: *Changing Requirements Scope* 172
- Figure 4.69: *Superbubbles* 173

- Figure 4.70: Allocation of a Split Process 175
- Figure 4.71: Generic Traceability Matrix 176
- Figure 5.1: The Total System Life Cycle 182
- Figure 5.2: Order of Precedence: Process, Methods, Tools 183
- Figure 5.3: Concurrent Work Flow 189
- Figure 5.4: System Specification Models 191
- Figure 5.5: Avionics Hierarchy 193
- Figure 5.6: Mappings Between Specification Layers 195
- Figure 5.7: The Traceability Tree 199
- Figure 6.1: A Roadmap for the System Specification Model 207
- Figure 6.2: Environment Model for Nurse's Patient-Monitoring Task 211
- Figure 6.3: Environment Model with Superbubble for Context Definition 213
- Figure 6.4: Requirements Context Diagram for Patient-Monitoring System 214
- Figure 6.5: Option for an Expanded Context 216
- Figure 6.6: DFDO, Observe And Analyze Patients 217
- Figure 6.7: Selected Requirements Dictionary Entries for Flows on DFDO 218
- Figure 6.8: Process Specifications for Processes in DFDO 218
- Figure 6.9: Enhanced DFDO Patient-Monitoring System 220
- Figure 6.10: DFD5, Communicate With Nurses 221
- Figure 6.11: AFCD, Patient-Monitoring System 223
- Figure 6.12: AICD Patient-Monitoring System 225
- Figure 6.13: Enhanced DFDO with Allocation Superbubbles 226
- Figure 6.14: DFD5, Communicate With Nurses, with Superbubbles for Allocation 227
- Figure 6.15: AFDO, Patient-Monitoring System 229
- Figure 6.16: AIDO: Patient-Monitoring System 230
- Figure 6.17: Traceability Matrix for the Patient-Monitoring System 231
- Figure 6.18: DFD for Requirements Allocated to the Portable Monitoring Computers 235
- Figure 6.19: Hardware/Software Configurations 237
- Figure 6.20: Central Monitoring Computer AFD 238
- Figure 6.21: Central Monitoring Computer AID 239
- Figure 6.22: Single-Hardware/Multiple-Software Module Configuration AFD 240
- Figure 6.23: Single-Hardware/Multiple-Software Module Configuration AID 241
- Figure 6.24: Multiple Hardware and Software with Static Allocation 242
- Figure 6.25: Multiple-Hardware/Distributed-Software with Dynamic Allocation 243
- Figure 6.26: Software Enhancements for Single-Hardware/Multiple-Software with Dynamic Allocation 243

- Figure 6.27: Traceability Matrix for Single-Hardware/Multiple-Software with Dynamic Allocation 244*
- Figure 6.28: AFD and AID with Power and Ground Distribution 246*
- Figure 6.29: AFD and AID of an Electromechanical Device 247*
- Figure 6.30: AFD and AID of a Mechanical Subsystem 248*
- Figure 6.31: A Manufacturing Process 251*
- Figure 6.32: Typical Computer Hardware Configuration 253*
- Figure 6.33: Typical Structure Chart 254*
- Figure 6.34: Forming Architecture Modules 260*
- Figure 6.35: Suggested Information-Hiding Categories 262*
- Figure 6.36: Illustration of Abstract Interfaces 263*
- Figure 6.37: Early Software Architectures 269*
- Figure 6.38: Horseshoe Architecture 270*
- Figure 6.39: An Alternative View of a Control Hierarchy Architecture 270*
- Figure 6.40: A Workflow-Centered Architecture 271*
- Figure 6.41: Business Object-Oriented Architecture 272*
-
- Figure 7.1: The Architecture Template, Adapted for a System Development Project 276*
- Figure 7.2: Development Process Requirements Context 277*
- Figure 7.3: DFD0, Develop System 279*
- Figure 7.4: DFD1, Specify Deliverable System 280*
- Figure 7.5: EDFD0: Enhanced Develop System 282*
- Figure 7.6: Development Process Architecture Context 286*
- Figure 7.7: Development Process Architecture Modules 289*
- Figure 7.8: Hardware/Software Template 292*
- Figure 7.9: System Integration Template 293*
- Figure 7.10: Customer Specification Template 293*
- Figure 7.11: System Specification Template 294*
- Figure 7.12: Hardware Specification Template 294*
- Figure 7.13: Software Specification Template 295*
-
- Figure 9.1: Context Diagram 305*
- Figure 9.2: System Timing Specification 306*
- Figure 9.3: Initial Entity Model 309*
- Figure 9.4: Enlarged Entity Model 310*
- Figure 9.5: Enhanced Requirements Model, Existing Sampling Module 312*
- Figure 9.6: Selected PSPECs of Enhanced Existing Sampling Module 314*
- Figure 9.7: Architecture Model for Existing Sampling Module 315*

- Figure 10.1: DFD0, Analyze Groundwater 317
- Figure 10.2: CSPEC0, Analyze Groundwater 318
- Figure 10.3: DFD1, Sample And Mix Specimen And Reagent, with Associated PSPECS 319
- Figure 10.4: DFD2, Stabilize Mixture, with Associated PSPECS 320
- Figure 10.5: DFD3, Test Mixture, with Associated PSPECS 321
- Figure 10.6: DFD4, Analyze Raw Results, with Associated CSPEC, PSPECS 322
- Figure 10.7: DFD4.5, Interpret Analysis, with Associated PSPECS 323
- Figure 10.8: DFD4.6, Calibrate For Pollutant, with Associated PSPECS 324
- Figure 10.9: DFD5, Respond To Operator Requests, with Associated PSPEC 325
- Figure 10.10: CSPEC5, Respond To Operator Requests 326
- Figure 10.11: DFD5.1, Route Operator Requests, with Associated PSPECS 327
- Figure 10.12: DFD5.3, Store And Select Historical Results, with Associated PSPECS 328
- Figure 10.13: PSPEC6, Read Specimen Data; and PSPEC7, Remove Analyzed Specimen And Data 328
- Figure 10.14: DFD8, Select Specimens, Pollutants, And Reagents 329
- Figure 10.15: PSPEC8.1, Run Specimen 329
- Figure 10.16: PSPEC8.2, Rerun Specimen 330
- Figure 10.17: PSPEC8.3, Select Pollutant And Reagent 330
- Figure 10.18: EDFD0, Enhanced Analyze Groundwater 332
- Figure 10.19: PSPECS in Maintenance and Support Region of EDFD0 335
- Figure 10.20: PSPEC in Input-Processing Region of EDFD0 336
- Figure 10.21: PSPECS in User-Interface Region of EDFD0 336
- Figure 10.22: DFD17, Get Operator Entries, with Associated PSPECS 337
- Figure 10.23: DFD14, Display System Status And Mode, with Associated PSPECS, CSPEC 338
- Figure 10.24: DFD12, Print Results, with Associated PSPECS 339
- Figure 10.25: DFD13, Transmit Results, with Associated PSPECS 339
- Figure 10.26: DFD19, Monitor Operational Status, with Sample PSPECS 340
- Figure 10.27: Enhanced CSPEC0 341
- Figure 10.28: Architecture Flow Context Diagram 342
- Figure 10.29: Architecture Interconnect Context Diagram 343
- Figure 10.30: Sample Analyzer Architecture Modules 345
- Figure 10.31: Requirements for Optical Sensor, Waste Handler, and Reagent Handler 346
-
- Figure 11.1: Architecture Modules 349
- Figure 11.2: Allocated EDFD0, Analyze Groundwater 350
- Figure 11.3: DFDs of Split Processes, with Specific Superbubble Allocation 352

- Figure 11.4: System-Level Traceability Matrix 353*
- Figure 11.5: DFD, Sample Analyzer 02 355*
- Figure 11.6: DFD, Workstation 01 355*
- Figure 11.7: CSPEC, Workstation 01 356*
- Figure 11.8: DFD, Operator Procedures 06 356*
- Figure 11.9: DFDs for Printer 04, Transceiver 05, and Bar Code Reader 03 357*
- Figure 11.10: EDFD, Sample Analyzer 02 358*
- Figure 11.11: EDFD, Workstation 01 360*
- Figure 11.12: EDFD, Bar Code Reader 03 362*
- Figure 11.13: System-Level AFD 363*
- Figure 11.14: System-Level AID 363*
- Figure 11.15: Modified DFD1, with Allocations 364*
- Figure 11.16: Allocated Child Diagrams of DFD1, with Typical Associated PSPECs 365*
- Figure 11.17: DFD2, Stabilize Mixture, with Allocations 366*
- Figure 11.18: DFD2.3, Conclude Stabilization, with Associated PSPECs 367*
- Figure 11.19: Sample Analyzer Traceability Matrix 368*
- Figure 11.20: EDFD, Sample Analyzer Processor 370*
- Figure 11.21: EDFDs for Optical Sensor, Waste Handler, and Reagent Handler 372*
- Figure 11.22: Sample Analyzer AFD 373*
- Figure 11.23: Sample Analyzer AID 374*
-
- Figure 12.1: Groundwater Analysis System AFD, with Dual Concurrent Sample Analyzer Modules 377*
- Figure 12.2: Groundwater Analysis System AID, with Dual Concurrent Sample Analyzer Modules 378*
- Figure 12.3: Requirements Dictionary 383*
- Figure 12.4: Architecture Dictionary 385*

This page intentionally left blank

Chapter 3

A Framework for Modeling Systems

3.1 A MODEL FRAMEWORK

In the preceding chapter, we discussed numerous facets of systems in the categories of general characteristics, views, and requirements.

We now focus on the development of such systems. In this chapter, we explore the role of models in system development and introduce a framework to organize the many different models created during the development process. This framework captures what we know about systems, including the properties discussed in Chapter 2.

The framework serves as a road map for the development process, providing a cue for *what* models to build and *how* to build them. It assists us in keeping all the models related and linked to each other. In the first sections that follow, we start with a discussion of models and their usefulness.

3.2 MODELS IN GENERAL

Most industries use models for purposes such as studying requirements for systems, examining feasibility and manufacturability, and determining how to build an actual system. In the computer hardware and software industry, models are used for some parts of the development process (usually for software require-

ments or design, or for hardware layout), but no techniques are widely used for modeling the *entire* system. Before discussing such modeling techniques, we discuss why models are useful.

3.2.1 Models Are Useful Abstractions

As Figure 3.1 shows, a model is an abstraction highlighting some aspects of real-world systems in order to depict those aspects more clearly. A model has an objective (the question we want it to answer) and a viewpoint (the point of view of one or more stakeholders: users, developers, and so on). Abstract models reduce the complexity of the real world to digestible chunks that are simpler to understand.

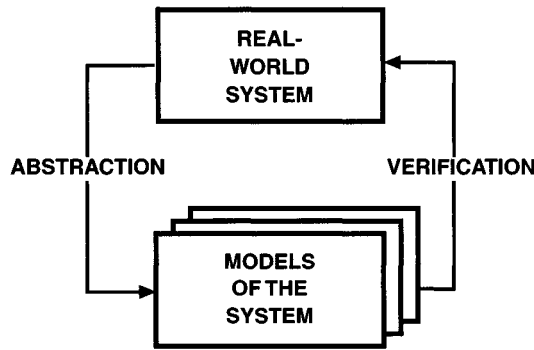


Figure 3.1: The Role of a Model.

On the other hand, abstract models are just representations, omitting some aspects of real-world systems, at least temporarily, but mapping what we hope to understand into a form that we can understand. Different types of models answer different types of questions about the system they represent, but even if we build a hundred different models, they could not answer every possible question about the system. That can only be done by the final system itself.

If we decide to build more than one model of a given system to investigate different aspects, then we should somehow organize these models according to their relationships to each other and to the system. This is why we need a framework.

3.2.2 Model Representations and Reuse

Before we discuss the framework for modeling systems, we expand on the idea of using models in two ways: First, models can be expressed using different notations; second, good models can be reused in different applications.

Though they appear very different from each other, Figures 3.2 through 3.5 can all represent the same scenario. Consider this description of a junior high school:

Students enter school in seventh grade. Most of the students proceed to eighth grade, but some skip directly to ninth. Nobody graduates directly from eighth grade, but some leave school before graduating. The rest go on to ninth grade and then graduate.

In Figure 3.2, “The Bathtub Model”—adapted by permission from *General Principles of Systems Design* by Gerald M. Weinberg and Daniela Weinberg [Weinberg 88]—flows into and out of the various tubs can represent the flows of students into and out of grades. Tubs 1, 2, and 3 represent 7th, 8th, and 9th grades, respectively. S indicates the set of students entering school. P1 represents the students progressing to 8th grade. Q1 depicts the small number of students skipping 8th grade and going directly to 9th grade. P2 shows the normal progress from 8th to 9th grade. Q2 and Q3 show students leaving school without graduating. P3 represents the students graduating from 9th grade.

The Weinbergs use the bathtub model to explain the set of differential equations given here as Figure 3.3: Those equations can abstract the same junior high school situation in a different manner. N1, N2, and N3 either can represent the quantities of water in the three tubs or can indicate the number of students in three grades. N1' represents the rate of change of N1 over time, and so on.

Figure 3.4—a Structured Analysis data flow diagram—shows yet another representation of the bathtub model, and of the same real-world system.

Finally, Figure 3.5 gives the context diagram for Figure 3.4, once again representing the same real-world system, but in a more abstract form.

The models in Figures 3.2 through 3.5 also can show how models can be reused. The four models can be used to illustrate different applications, fitting the following description of a company’s training program just as well as they fit the junior high school scenario, and just as well as they could fit many other similar scenarios.

Everyone joining Company X starts as an unskilled worker. The company's policy is to provide training and education for its employees. No one is allowed to work without some minimal vocational training that gets him or her into the semi-skilled labor pool. Those who have college degrees move to the skilled category, bypassing the semi-skilled pool. After five years in the semi-skilled category, workers automatically progress to the skilled pool. Eventually, employees either leave for better opportunities or retire.

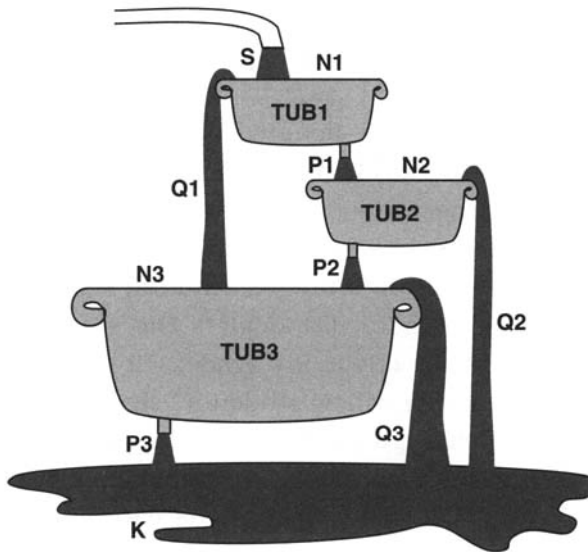


Figure 3.2: The Bathtub Model.

$$\begin{aligned}
 N1' &= S - (P1 + Q1) \\
 N2' &= P1 - (P2 + Q2) \\
 N3' &= (Q1 + P2) - (P3 + Q3)
 \end{aligned}$$

Figure 3.3: Equations Representing the Bathtub Model.

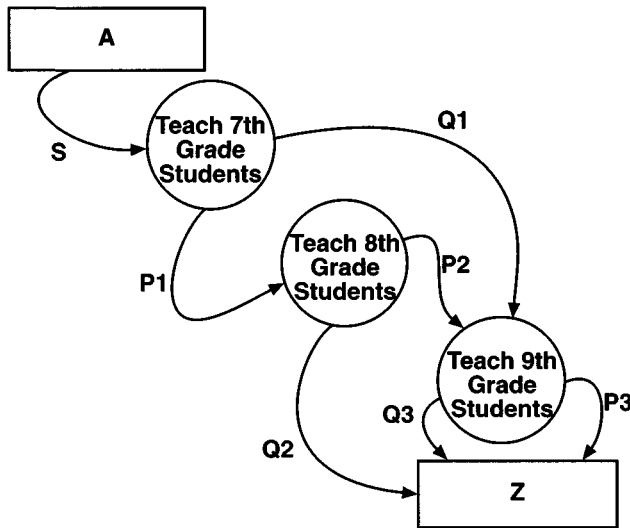


Figure 3.4: A Data Flow Diagram Representing the Bathtub Model.

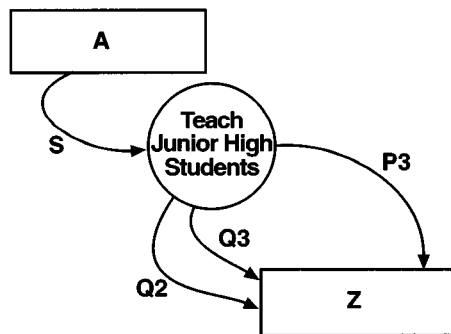


Figure 3.5: A Context Diagram of the Model in Figure 3.4.

3.3 EXPLOITING SYSTEM HIERARCHIES

In Sections 2.1.2 and 2.1.3, we explained that all real-world systems consist of subsystems, or—looking in the other direction—that every system is part of a larger system. In other words, systems come in hierarchies. Using these hierarchies is the first step in constructing our modeling framework.

3.3.1 Why Exploit Hierarchies?

Why do we want to exploit the idea of system hierarchies? Because we want to reduce complexity by not thinking about everything at once.

- At the highest level of a model, we establish the place of the system in its environment and define the broad objectives of the system and its relationships with that environment (for example, communication and physical linkage).
- Using these broad objectives, we proceed into the requirements and architecture of our system, remembering that it can be manual, automated (by various technologies), or both. Creating an architecture for the system partitions it into subsystems that can themselves be considered self-contained systems—similar to the top-level system. By iterating this partitioning procedure, as illustrated in Figure 3.6, we simplify the problem by treating each subsystem, sub-subsystem, and so on, as a system in its own right, with its external interconnections and interactions represented in the level above.

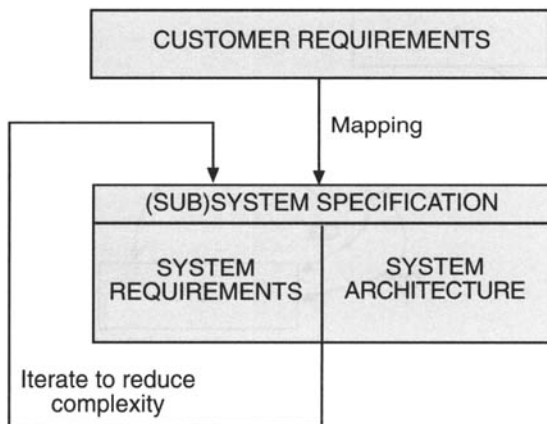


Figure 3.6: Hierarchies to Reduce Complexity.

3.3.2 What Are the Benefits and Pitfalls of Layered Systems?

There are many benefits in hierarchically organized systems and subsystems. Every layer of system definition supplies some of the requirements for the layer

below. At the top, a firm link is established between the system and its environment. If we can stabilize the upper-level requirements and architecture early, the lower-level design can proceed much more effectively. We can anticipate high-risk subsystems and use prototyping to resolve those risks. Working on a certain level of abstraction helps us concentrate on that level and not get too detailed too fast.

One point needs to be strongly emphasized:

System specification and development are not necessarily top-down processes.

Overlooking this heuristic can be a major pitfall. The fact that, for convenience, many of our descriptions of the process are presented top-down does not detract from this statement; neither does the top-down appearance of the figures. The top layers do not have to be complete before we can work on the lower layers; in some cases, it is appropriate to work upward from the lower layers. Think of development as a concurrent or iterative process—there is always some work going on in every layer.

The layered model results in a specification hierarchy and a representation of the requirements flows between layers. The process of filling in the framework and developing these models is discussed in Chapter 5, and illustrated in Part II.

Integration is key to developing the system: “The whole is greater than the sum of its parts.” The systems we develop require that all of their components are integrated: software with software; hardware with hardware; software with hardware; automated with manual; and especially, system with environment. So, despite developing many separate models, we need subsystems linked to other subsystems, and layers linked to higher and lower layers. This is the purpose and benefit of the modeling framework.

3.3.3 How Many Models?

Many methods insist on building one large analysis model (as in Structured Analysis or entity-relationship modeling) and, separately, one large design model (as in Structured Design or in many object-oriented methods). We, too, use the “divide and conquer” approach in our framework, but we ensure that the sub-models are integrated.

How many models do we build, then? If we consider an integrated set of sub-models as a single model, then we build a model for the overall system and a model for each of its subsystems, sub-subsystems, and so on. Each of these

models itself can contain numerous models of its own: models for requirements, design, architecture, information structures, interconnects, and many more. Good methods, and tools that automate them, will support all of these multifarious models and the links between them.

3.3.4 Where Do We Stop?

We have established that every system is part of a larger system. Looking in the other direction, How far down should we decompose a subsystem into further subsystems?

In larger systems, system developers will stop at the level where direct system responsibility ends, or where they have no constraints to impose internally to a subsystem. Then, specialists in those subsystems can decide whether to continue with the same process or to switch to some other approach that is specific to their discipline. For example,

- we decompose a multidisciplinary system into parts that are, say, mechanical or hydraulic, and pass those subsystems to the corresponding specialists
- in MIS, we often decompose until we can clearly differentiate between human activities (for example, clerks doing part of the work) and software activities (computer programs doing the remaining work)
- in embedded, real-time systems, we might decompose until we have a better understanding of the hardware/software split

A software subsystem can usually be decomposed into further subsystems, and an organizational subsystem can be organized as cooperating groups of further organizational subsystems. The techniques, methods, and tools for specialized subsystem development are often more mature and better automated than those used for overall system development. In this book, we do not discuss specialized hardware, software, or organizational methods, but instead refer to other publications on these topics.

On the other hand, we do not have to decompose every system into subsystems that comprise only one technology. Sometimes, the system levels stop where several related technologies are used in a single subsystem. For example, in a hydraulic subsystem with electromechanical valves, it would not make much sense to separate these two technologies, because they exist to support each other.

Figure 3.7 shows various alternatives for decomposing subsystems: The top-level decomposition separates a human subsystem from a purely mechanical subsystem and leaves a multi-technology subsystem to be further decomposed. On the next level, a software subsystem is further decomposed into two software subsystems. At the lowest level, we find a software subsystem, a human subsystem, and a subsystem that uses mixed technology but is treated as one unit.

In later chapters, we discuss more criteria to determine where to stop decomposing.

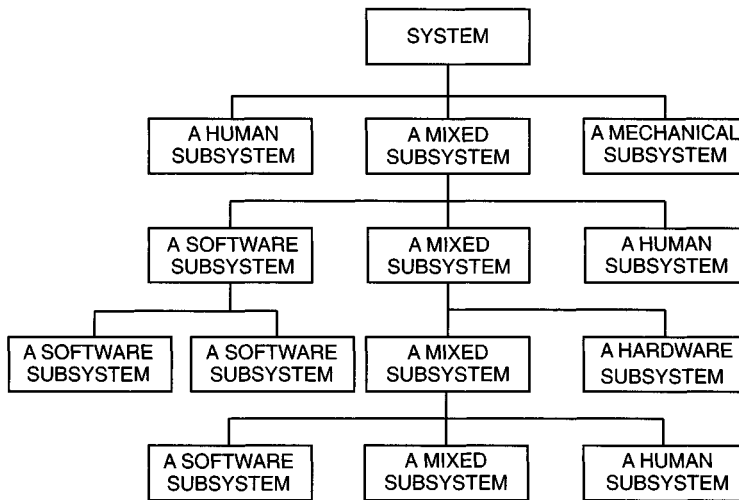


Figure 3.7: Partitioning a System.

3.4 EXPLOITING THE *WHAT/HOW* CLASSIFICATION

The next step in constructing our modeling framework is to use the *what/how* classification of systems that we discussed in Section 2.2.3. As you may have noticed in Figure 3.6, every specification consists of two parts: system requirements and system architecture. Both of these parts contain models. The system requirements model is a technology-independent model of the problem the system is to solve: It represents the *what*. The system architecture model is a technology-dependent model of the solution to the problem: It represents the *how*. These two models are created for the entire system and for every subsystem—hardware, software, human, or mixed technology—down to the lowest level.

3.4.1 Separation of *What* and *How*

The separation of the *what* and the *how* is extremely important for the following (and possibly other) reasons:

- It is often very useful to understand a problem (the *what*) independently of any particular solution (the *how*). (Conversely, there are situations where it is useful to develop a single architecture that will satisfy a whole class of problems.)
- Any given problem has many possible solutions. Selection of a particular solution (the *how*) is a trade-off process; we often need to make numerous different trade-offs while keeping the problem statement (the *what*) unchanged.
- The separation supports the generally recognized principle of separation of concerns, which means dealing with only one part of the system's complexity at a time. The requirements model (the *what*) only has to cope with essential problems; the architecture model (the *how*) has to cope with many constraints imposed by technology, organization, and so forth.
- Finally, seldom do we build systems totally from scratch. Most systems we build are either implementations using new technology (only changing the *how*) or the integration of several previous systems into a new system.

The separation of the *what* and the *how* gives us the power to reimplement the *what* using new technology, but it also gives us the power of reusability—not just for software or hardware, but for requirements as well. This is particularly important, because requirements are much more stable over much longer periods of time than technology [McMenamin 84].

In this book, we use the *what/how* classification for yet another important purpose. As we construct several different models later in the book, we would have to handle a lot of complexity at once if we addressed how to construct them at the same time as we addressed what to construct. So, we have split the following chapters using the *what/how* separation: Chapter 4 describes what models belong in our framework; Chapters 5 through 7, and all of Part II, describe how to develop these models.

As mentioned, Chapters 5 through 7 describe the *how* of constructing the models, but in Part II, we describe the *how* of applying these models to some real systems—from that perspective, Chapters 5 through 7 represent the *what!* This is analogous to the layered structure of systems, in which architectural or design decisions in one layer result in requirements in the layer below. This same principle applies to this book, which itself is a kind of system.

The purpose of Part II and of the on-line model is to exemplify what real projects must do—the *what*, the *how*, and also the *when*. The *when* refers to project planning and scheduling, including such issues as which tasks are conducted concurrently, and which sequentially. Throughout the 1970's and 1980's, simplistic process models like the waterfall model predominated. We know now that there are no simple solutions to project planning and scheduling. Rather, these are decisions that must be made for each project, by project management. It is the manager's job to observe the process, to watch and interpret the results of individual steps, to take into account many constraints, and based on all of that, to reconfigure dynamically the *what*, the *how*, and the *when* of the project.

3.4.2 The Architecture Template

Our modeling framework employs an extension of the *what/how* split to classify systems, subsystems, their components, and their activities according to a generic architecture template. Figure 3.8 shows that this architecture template classifies a system or subsystem into five categories:

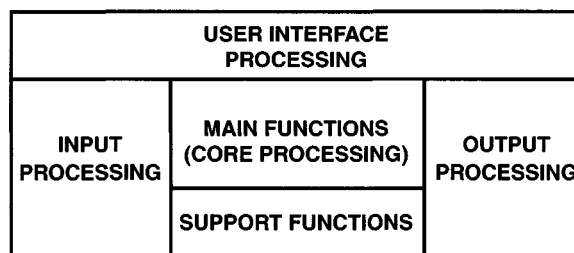


Figure 3.8: The Architecture Template.

1. The center region contains the main functions of the system: the core functional processing. Here, we model things that the system absolutely has to do, things that belong to the essence of the system, independent of any technology.

2. The top region hosts those parts of a system that interact with the users. It contains all subsystems, functions, and activities that make up the human-machine interface. It controls access to the system, and it accepts input from and prepares output for the human user, all in whatever forms are established with the user.
3. The left region contains the functions and subsystems that interface with other systems and subsystems to provide input for our system. It has to establish interconnections, request input, check it for acceptability, preprocess it, and perform many other input-related activities.
4. The right region provides similar resources for the output of our system to other systems. This includes establishing interconnection, converting output to the form needed for transfer, sending it, and so on.
5. The bottom region houses any functions or subsystems that provide support to the rest of the system to keep it running. These include self-test procedures, error logging, fault detection, and also maintenance functionality. This region is fundamentally different from the other outer regions: those regions deal with various kinds of interfaces between the system and its environment, whereas the bottom region deals with functions that support the system internally. The support functions might require additional inputs and outputs, but still, they are internal functions.

How do we use the architecture template in our modeling framework? As shown in Figure 3.9, the template mainly helps us with mapping between requirements (the *what*) and architecture (the *how*). This mapping can be applied on any level: for the overall system and for every subsystem on any layer.

From the requirements viewpoint, we augment or enhance the required functionality of the system (which is modeled in the core part of the template) with a ring of functionality supporting the core processing of the system. These augmented, or enhanced, requirements are packaged into architectural subsystems.

From the architecture viewpoint, the template provides an excellent starting point for building information-hiding subsystems [Parnas 71]. The center hides the essential functions, the top region hides the user-interface technology and behavior, the left and right regions hide input/output specifics, such as device characteristics and protocols, and the bottom region hides support functionality,

such as service and maintenance modules, and many more. An alternative name for the template might be the information-hiding template.

Once we establish the functionality of the system and subsystems, we can easily categorize and extract the core requirements for future reuse.

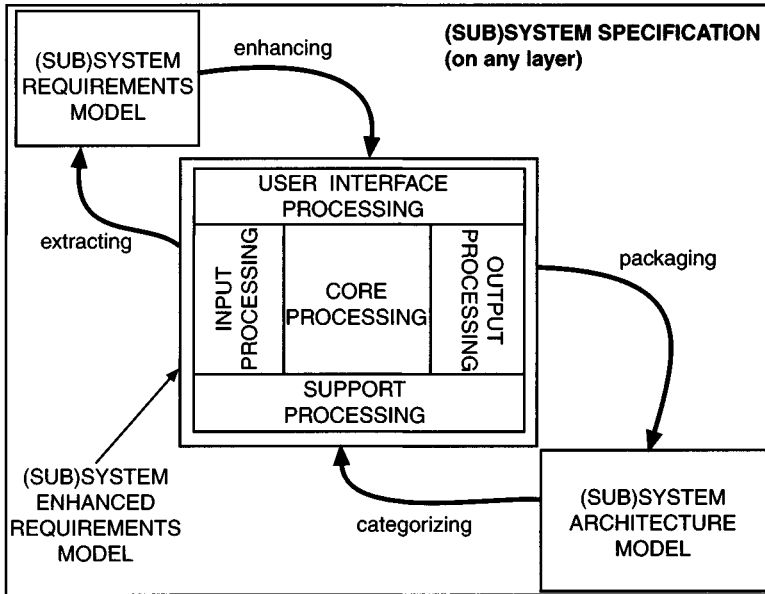


Figure 3.9: The Template Bridging Requirements and Architecture Models.

3.4.3 Using the Architecture Template

In this section, we present two examples of using the architecture template to classify the functionality of systems. To demonstrate that it can be used on any layer of a system hierarchy and for any kind of application (automated or not), we start with an organizational system composed of automated and manual parts.

The template in Figure 3.10 illustrates the activities that are performed by nurses at their station in a hospital. We can divide the activities into the five categories. By doing so, if the procedures for helping visitors (in the user-interface part of the template) are changed, the rest of the system can remain unchanged. The same is true for changing the policy for checking stock supplies in the maintenance region, and for any changes involving just a single region of the template.

| | | |
|--|---|--|
| <p>USER INTERFACE</p> <ul style="list-style-type: none"> • HELP VISITORS • GUIDE PATIENTS TO DOCTORS' OFFICES • ANSWER PHONE CALLS • RESPOND TO PATIENT EMERGENCY CALLS | | |
| <p>INPUT PROCESSING</p> <ul style="list-style-type: none"> • CHECK IN PATIENTS • SCHEDULE PATIENT MONITORING • RECEIVE AND LOG INCOMING MEDICATION • RECEIVE SUPPLIES | <p>MAIN FUNCTION</p> <ul style="list-style-type: none"> • MONITOR PATIENT HEALTH • DISPENSE PRESCRIBED MEDICATIONS • ADMINISTER ROUTINE TESTS • KEEP PATIENT HEALTH RECORDS • KEEP PATIENT BILLING RECORDS • ASSIST DOCTOR ON ROUNDS | <p>OUTPUT PROCESSING</p> <ul style="list-style-type: none"> • CHECK OUT PATIENTS • ISSUE ALERTS TO DOCTORS AND STAFF • FILE REPORTS • ORDER SUPPLIES • SEND PATIENT BILLING TO ACCOUNTING DEPARTMENT |
| | <p>SUPPORT</p> <ul style="list-style-type: none"> • MONITOR DESK OPERATIONS • CHECK NURSES' SCHEDULES • CHECK STOCK SUPPLIES | |

Figure 3.10: Classification of Activities in a Manual System.

This nurses' station is a subsystem of an overall hospital system, for which we could also classify the activities. Depending on the purpose of the whole hospital model, the nurses' station would be part of the system core or may be considered part of the maintenance subsystem (to support the doctors).

The second example is of an automated system. Using the hospital application, we show how to classify the automated activities of a patient-monitoring system in Figure 3.11.

The architecture template is a very powerful modeling tool that we use repeatedly to classify system or subsystem activities. Not only can it be used to bridge the requirements/architecture models, it can be used very early in a project, before we even know the requirements or make decisions on the architecture, to discover topics to be treated in more detail later. It can also be used in distributing work among project members, allowing them to work concurrently.

| | | |
|---|--|---|
| USER INTERFACE <ul style="list-style-type: none"> • GET NURSE ENTRIES • VALIDATE NURSE ENTRIES • DISPLAY ALERTS • FORMAT AND DISPLAY PATIENT-MONITORING STATUS | | |
| INPUT PROCESSING <ul style="list-style-type: none"> • MEASURE PATIENT VITAL SIGNS • MEASURE PATIENT MOTION • VALIDATE VITAL SIGN SENSOR READINGS • ISSUE EQUIPMENT FAILURE ALERT | MAIN FUNCTION <ul style="list-style-type: none"> • SET UP SCHEDULE FOR MONITORING • FIND UNSAFE VITAL SIGN RANGE FOR PATIENTS • COLLECT DATA FOR PATIENT • PERFORM ANALYSIS FOR SELECTED PATIENT • ISSUE PATIENT ALERT | OUTPUT PROCESSING <ul style="list-style-type: none"> • PRINT REPORT |
| | SUPPORT <ul style="list-style-type: none"> • UPDATE PATIENT SAFE RANGES • UPDATE EQUIPMENT OPERATING RANGES | |

Figure 3.11: Classification of Activities in an Automated System.

3.5 EXPLOITING THE INFORMATION/MATERIAL/ENERGY CLASSIFICATION

The final step in constructing our modeling framework is to classify systems according to their information, material, and energy processing characteristics that we discussed in Section 2.2.1.

3.5.1 A Generic Subsystem Structure

In Chapter 2, we showed that everything a system does can be classified into material processing, energy processing, and information processing. Since material and energy processing are quite different from information processing, we can treat these two areas separately. If we combine this decision with the idea of categorization provided by the architecture template, we end up with the system partitioning shown in Figure 3.12, which divides the processing into finer classifications. These finer classifications become subsystems of the overall system. As Figure 3.12 shows, there are subsystems that do the two different types of processing, but the overall, or boundary, subsystem does both. Figure 3.13 explains more about the functions of the various subsystems.

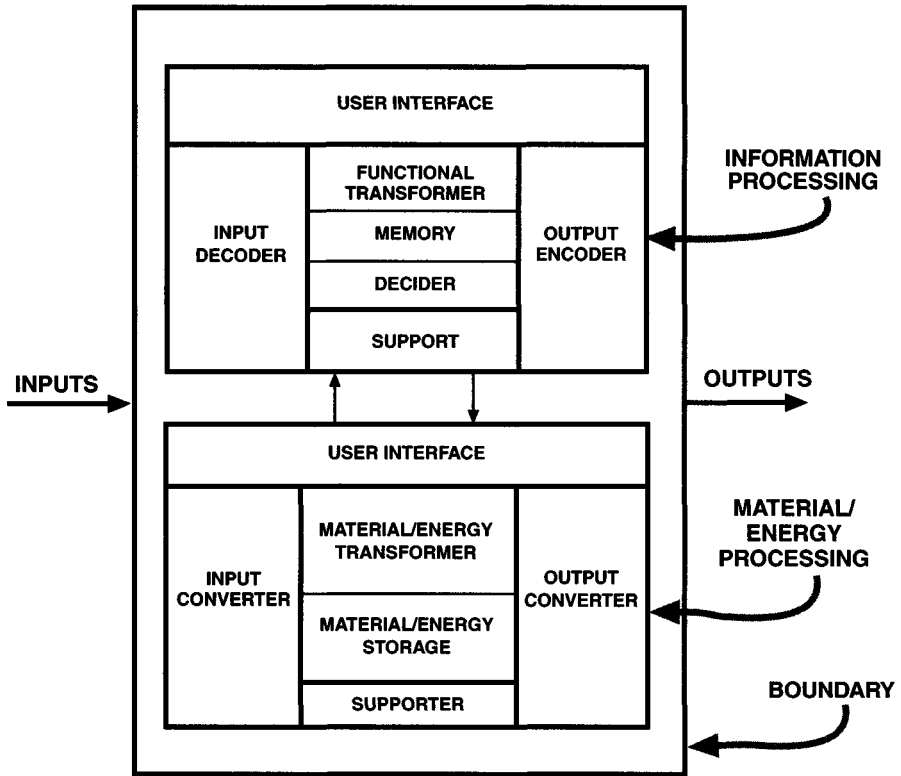


Figure 3.12: Separating Material/Energy Processing from Information Processing.

Note that there is no box in material/energy processing equivalent to the decider in information processing. With today's systems, the decision-making function is almost always an information processing function. Consequently, Figures 3.13 through 3.16 all have a blank entry in the material/energy processing side.

| | |
|--|---|
| Boundary: Subsystem(s) that form a barrier around a system, shielding it from its environment. | |
| Information Processing Subsystems | Material/Energy Processing Subsystems |
| User Interface: Subsystem(s) to allow information exchange with external human users. | User Interface: Subsystem(s) to allow material/energy exchange with external human users. |
| Input Decoder: Subsystem(s) to convert the coding of external information for internal use. | Input Converter: Subsystem(s) to transform material/energy from external to internal forms. |
| Functional Transformer: Subsystem(s) to transform input information into output information. | Material/Energy Transformer: Subsystem(s) to transform and associate material/energy inputs to outputs. |
| Memory: Subsystem(s) to retain for later use information, its relationships, and its organization. | Material/Energy Storage: Subsystem(s) to store material/energy for later use. |
| Decider: Subsystem(s) to control (for example, enable, inhibit, or trigger) functional transformers. | |
| Output Encoder: Subsystem(s) to convert the coding of internal information for external use. | Output Converter: Subsystem(s) to transform material/energy from internal to external forms. |
| Support: Subsystem(s) to support system monitoring, servicing, and reconfiguration. | Supporter: Subsystem(s) to enable maintenance, growth, and reconfiguration. |

Figure 3.13: Generic Description of Subsystem Responsibilities.

3.5.2 Categories of a Deliverable System

For a deliverable system, product, or component, the categorization scheme introduced above can be a useful starting point for brainstorming the subsystems. Figure 3.14 makes the generic categories of Figure 3.13 specific to all deliverable systems, and Figure 3.15 makes them specific to a cruise control system.

Note that the categories in the generic template are applicable to many systems, although there are usually some that are not applicable to a specific system. Consider the generic categories a pattern for thinking about a system and its subsystems.

| | |
|--|---|
| <p>Boundary: The external housing, casing, or such other exterior that shelters the system from its environment.</p> | |
| <p>Information Processing Subsystems</p> | <p>Material/Energy Processing Subsystems</p> |
| <p>User Interface: Data entry and display devices.</p> | <p>User Interface: Access mechanisms allowing operator insertion and extraction of physical items and electrical or mechanical energy.</p> |
| <p>Input Decoder: Processor(s) of information inputs from other systems, converting them, as needed, from their received formats to internal system formats.</p> | <p>Input Converter: Mechanism(s) for the physical manipulation of received physical items or energy into the orientation or form needed internally.</p> |
| <p>Functional Transformer: Input-to-output information conversion devices performing processes such as algorithms, functions, math equations, or string manipulations.</p> | <p>Producer: Electrical or mechanical devices that process received physical items or energy, and convert them into the desired product. For example, an automatic mechanism that receives component parts and assembles them into a finished product.</p> |
| <p>Memory: Device(s) that store, for later use, information from the operator, from other systems, or from the processes of this system retained, possibly with its relationships and organization.</p> | <p>Storage: Any part(s) of the system that store material or energy for later use, such as a storage room, a shelf, a battery, or a water reservoir.</p> |
| <p>Decider: Control processors by which information processing and resources are scheduled, and which establish the different states or modes of behavior of the system.</p> | |
| <p>Output Encoder: Processor(s) of information outputs to other systems, converting them, as needed, from their internal system formats to external formats.</p> | <p>Output Converter: Mechanisms for the physical manipulation of produced physical items or energy into the orientation or form needed externally. For example, the automatic packaging of manufactured products for shipment.</p> |
| <p>Support: Processors that perform tasks such as fault isolation, error handling, service monitoring, system reconfiguration, and graceful degradation.</p> | <p>Supporter: Access mechanisms for physical maintenance, growth, and reconfiguration.</p> |

Figure 3.14: A Template for Subsystems of a Deliverable, Human-Operated System.

| Boundary: The external housing, casing, or such other exterior that shelters the system from its environment. | |
|--|---|
| Information Processing Subsystems | Material/Energy Processing Subsystems |
| User Interface: Data entry and display devices. | User Interface: Access mechanisms allowing operator insertion and extraction of physical items and electrical or mechanical energy. |
| Input Decoder: Processor(s) of information inputs from other systems, converting them, as needed, from their received formats to internal system formats. | Input Converter: Mechanism(s) for the physical manipulation of received physical items or energy into the orientation or form needed internally. |
| Functional Transformer: Input-to-output information conversion devices performing processes such as algorithms, functions, math equations, or string manipulations. | Producer: Electrical or mechanical devices that process received physical items or energy, and convert them into the desired product. For example, an automatic mechanism that receives component parts and assembles them into a finished product. |
| Memory: Device(s) that store, for later use, information from the operator, from other systems, or from the processes of this system retained, possibly with its relationships and organization. | Storage: Any part(s) of the system that store material or energy for later use, such as a storage room, a shelf, a battery, or a water reservoir. |
| Decider: Control processors by which information processing and resources are scheduled, and which establish the different states or modes of behavior of the system. | |
| Output Encoder: Processor(s) of information outputs to other systems, converting them, as needed, from their internal system formats to external formats. | Output Converter: Mechanisms for the physical manipulation of produced physical items or energy into the orientation or form needed externally. For example, the automatic packaging of manufactured products for shipment. |
| Support: Processors that perform tasks such as fault isolation, error handling, service monitoring, system reconfiguration, and graceful degradation. | Supporter: Access mechanisms for physical maintenance, growth, and reconfiguration. |

Figure 3.15: Subsystems of a Cruise Control System.

3.5.3 Categories of a People System

The systems we build reflect the organizations that build them. With this in mind, we can devise a categorization of organizational systems, such as of whole companies, departments, or groups of people cooperating to achieve a certain goal. Figure 3.16 maps the generic categories onto an organizational structure describing the various subsystems in such a context. Figure 3.17 shows categories specific to a garment factory. This kind of categorization helps to distinguish between value-adding functions and overhead functions in an organization: It can be used as a starting point for modeling business processes and identifying essential parts of them.

3.6 LAYERED MODELS: THE TRUTH AT LAST!

We discussed numerous characteristics and features of models in the previous sections, but we kept the important issue of layered models for the end of this chapter. In any discussion of systems, models of systems, or the process of building systems, the term “layer” plays an important role. Here, we explore several unique aspects of layers, and of the different relationships between layers, between elements in one layer, and between elements in different layers.

One purpose of this section is to dispel a couple of myths. First, there is the myth that all layered models fall into the category of functional decomposition or, worse yet, top-down functional decomposition. And second, that layered models are fundamentally incompatible with object orientation.

In the first pages of *Strategies for Real-Time System Specification*, we introduced a diagram titled, “The Total System Life Cycle,” little realizing at the time just how significant it was. It showed various layers of the system modeling process and layers of specifications resulting from that process. We elaborate on that diagram in Figure 5.1 of this book, but for now, we discuss some of its implications. What we have realized since creating that diagram is that there is tremendous similarity between systems, system models, and the system development process. Layers are an important part of these similarities, but they are also the source of some confusion. There is not just one kind of relationship between layers or elements of layers: We can identify several basic relationships that keep recurring in different systems, system models, and in the development process.

| Boundary: The external housing, casing, or such other exterior that shelters the system from its environment. | |
|--|---|
| Information Processing Subsystems | Material/Energy Processing Subsystems |
| User Interface: Data entry and display devices. | User Interface: Access mechanisms allowing operator insertion and extraction of physical items and electrical or mechanical energy. |
| Input Decoder: Processor(s) of information inputs from other systems, converting them, as needed, from their received formats to internal system formats. | Input Converter: Mechanism(s) for the physical manipulation of received physical items or energy into the orientation or form needed internally. |
| Functional Transformer: Input-to-output information conversion devices performing processes such as algorithms, functions, math equations, or string manipulations. | Producer: Electrical or mechanical devices that process received physical items or energy, and convert them into the desired product. For example, an automatic mechanism that receives component parts and assembles them into a finished product. |
| Memory: Device(s) that store, for later use, information from the operator, from other systems, or from the processes of this system retained, possibly with its relationships and organization. | Storage: Any part(s) of the system that store material or energy for later use, such as a storage room, a shelf, a battery, or a water reservoir. |
| Decider: Control processors by which information processing and resources are scheduled, and which establish the different states or modes of behavior of the system. | |
| Output Encoder: Processor(s) of information outputs to other systems, converting them, as needed, from their internal system formats to external formats. | Output Converter: Mechanisms for the physical manipulation of produced physical items or energy into the orientation or form needed externally. For example, the automatic packaging of manufactured products for shipment. |
| Support: Processors that perform tasks such as fault isolation, error handling, service monitoring, system reconfiguration, and graceful degradation. | Supporter: Access mechanisms for physical maintenance, growth, and reconfiguration. |

Figure 3.16: A Template for an Organization That Builds Deliverable Systems.

| | |
|---|--|
| Boundary: The building(s) that house the garment factory and its offices. | |
| Information Processing Subsystems | Material/Energy Processing Subsystems |
| User Interface: People who negotiate the designs and contracts with customers. | User Interface: Direct factory sales outlet, returns department. |
| Input Decoder: Purchase order processing, inventorying of received materials. | Input Converter: People unpacking received materials, repackaging them in a form that supports the production process; people from personnel department screening new hires. |
| Functional Transformer: Design department and equipment that transforms customer requests into actual designs to be manufactured. | Producer: Production line personnel and equipment that convert the received materials into finished garments according to the selected designs. |
| Memory: Storage of the designs, accounting records, shipping records, the employee records, and so on. | Storage: Supply cabinets, storage lockers, stock room, and so on. |
| Decider: Management that determines the production schedules, the factory plans, and the coordination of the factory floor. | |
| Output Encoder: People responsible for invoicing the customers, for waste disposal coordination, and so on. | Output Converter: The shipping department, delivery truck drivers, and so on. |
| Support: Customer accounting, payroll department, and so on. | Supporter: The facilities and maintenance crews who remove trash and keep the factory in operating condition. |

Figure 3.17: Subsystems of a Garment Factory.

Systems, models of systems, and the system development process share the following attributes:

- They are layered.
- The layers—once they are identified—form a structure that can be read and interpreted in any sequence: from the top layers to the lower layers, from right to left, from bottom to top, and so on. Moreover, the layers can be *developed* in any sequence: top to bottom, right to left, bottom to top, and so on, and the interpretation and development sequences are quite independent of each other.
- The number of elements per layer typically increases downward, giving the whole structure a pyramidal shape; but note that we sometimes have an independent structure of elements within one of the main layers. For such a structure, the basic statement of this paragraph is still true: The number of elements tends to increase downward.
- The elements forming the layered structure can be considered a set, either of activities or of entities. In any particular system, system model, or development project, these elements may be carried out or used in some prescribed sequence, concurrently, or in any combination of sequence and concurrency.
- Elements in the layers usually communicate and cooperate up, down, and sideways within and between layers. Communication and cooperation can be in the form of information, material, or energy, depending on the kind of system, model, or process in question. Some earlier models restricted the development process by asserting, for example, that all information flows vertically through the top layer, and that only information (not material or energy) can be communicated with the outside world. However, our model recognizes that information, material, and energy can all flow sideways to and from individual layers, and can all interact with the outside world.
- Every layer includes, deals with, or is associated with, some requirements, some architecture or design, some construction or implementation, and some integration and testing. Also, each layer usually requires planning, quality assurance, management, and other items, but we are not addressing these in this book.

An important conclusion for us is that layered models—for systems or as meta-models for system development processes—do not inherently imply any particular sequence. They represent a static structure (of a system, its development, the development process, or models of any and all of these) that can be populated in any convenient sequence that makes sense for the problem at hand. This point was beautifully made by Parnas—arguably the father of information-hiding structures—in [Parnas 86].

So, the layers and the elements in layers are nondirectional, but we are interested in their relationships. There are probably many types of relationships in layered models, but four of them are of special interest in system development: aggregation/decomposition, abstraction/detailing, supertype/subtype, and controlling/controlled. Let us look at each of these in detail, discussing properties of their relationships and examples from our methods and other well-known approaches.

3.6.1 Aggregation/Decomposition Relationship in Models

The architecture model, resulting from the architecture method, is an example of an aggregation/decomposition model. Such models characterize real physical elements, their sub-elements or parts, and their super-elements or assemblies. Elements in the higher layers actually consist of the elements in the lower layers, or conversely, elements in the lower layers are decompositions of those in the higher layers. The structure is also known as a whole/part structure [Coad 91] or a container/content structure: A given layer provides the container for the layer below, which is the content of the layer above. In entity-relationship modeling, entities can be linked by composed-of or consists-of relationships. In manufacturing terms, it is an assembly/subassembly/component structure. This type of structure is pervasive in engineering and in everyday life.

An aggregate actually involves more than just collecting sub-elements into a set. The sub-elements must also interface with each other, requiring linkages between them that may not be evident when they are considered separately. This is why we discuss enhancement of abstract requirements, using the architecture template, in our methods when the requirements are mapped into real physical modules.

We can better imagine aggregation/decomposition structures applied at the system levels, where physical hardware of various kinds is involved. For software, which does not have a physical form, it is not so clear. The trick is to imagine

that software does have a physical form. A complete software program or assembly can be considered as an architecture module at the highest software layer; major subprograms it contains are modules in the next layer down; sub-subprograms or subroutines (if any) form a further layer, and so on. As we discuss elsewhere in this book, a transition can be made from an aggregation/decomposition model to an object-oriented representation by defining modules to be aggregate objects, as described, for example, in [Page-Jones 95, Section 4.2]. Once in the object-oriented domain, other structures may apply, depending on the particular object-oriented approach used.

To summarize the usage of this relationship: We build layers to show physical packaging of elements into larger groups or assemblies. In each layer, we can define physical interfaces between elements or between groups. The grouping forms a sort of fence around its elements, potentially protecting the visibility of the interior elements or regulating the access to them. In software development, we use terms like information hiding, scope, and visibility control to describe the nature of the aggregation/decomposition relationship.

3.6.2 Abstraction/Detailing Relationship in Models

When we use an abstraction/detailing relationship in models, the higher layers are simply more abstract expressions of the lower layers, or conversely, the lower layers are more detailed expressions of the higher layers. The most familiar example of this relationship occurs in Structured Analysis (SA), usually represented by data flow diagrams. (Note that the control model of the real-time extensions of SA does not use this relationship—see Section 3.6.4.) The process model part of the requirements model, being founded on SA, uses the abstraction/detailing relationship for processes and their child diagrams. In a sense, the whole requirements model—if applied correctly—is abstract throughout because it consists only of narrative statements (albeit in structured form) that do not necessarily correspond to real physical groupings of processes, entities, or control structures.

The abstraction/detailing relationship often has been erroneously named abstraction/decomposition. Although we, too, have been guilty of using this terminology, we now disagree with it. First, abstraction and decomposition are not opposites, and the essence of these relationship name pairs is that they should be opposites, reflecting the upward and downward viewpoints in a layered model. Second, decomposition *is* the opposite of aggregation, which is why we use it in

the aggregation/decomposition relationship described above, where something is broken into the elements it contains. Consider this: An abstract requirement statement does not contain the more detailed requirements statements that describe it; however, a physical system element does contain the separable sub-elements of which it is an aggregate. If we take the elements of a physical system and assemble them, we get the physical system; if we assemble a set of detailed requirements, we merely have a collection of detailed requirements—the abstract and detailed requirements exist independently of each other, with an abstraction/detailing relationship between them.

Our categorizations of layered structures, then, have led us to an interesting paradox. The terms to which we objected earlier—functional decomposition and top-down functional decomposition—are frequently applied to Structured Analysis and its data flow diagrams, yet data flow diagrams, when used correctly to represent abstract requirements, do not involve decomposition at all: They involve detailing. When we use Structured Analysis to create essential models according to our own guidelines and those in [McMenamin 84] and [Robertson 98], we are not decomposing downward through the layers; we are adding detail. Going upward, we are not packaging or aggregating; we are abstracting.

Of course, if you are misusing SA to represent the aggregation and decomposition of physical structures, then anything goes, and we cannot take responsibility for the results (which are usually awful).

How do detailing of the required capabilities and decomposition of the physical structure relate to each other? As a system is developed, they proceed in parallel, with sufficient detail added to the required capabilities to satisfy the needs of a particular physical layer. This point is illustrated further in Part II.

3.6.3 Supertype/Subtype Relationship in Models

In the supertype/subtype relationship, an element in the higher layer—the supertype—includes all of the features that are common to its associated elements in the lower layer—its subtypes. These features—in the simplest case—are attributes (as they are called in information modeling) that are inherited by the elements on the lower layer. Starting from the lower level, supertypes are formed for sets of elements that share common attributes. Thus, we might have at the top level “vehicle,” and at the level below “ship,” “aircraft,” and “land vehicle.” Below “land vehicle,” we might have “bicycle,” “motorcycle,” “ATV,” and “automobile.” This tells us, for example, that an automobile is a land vehicle and a land vehicle is a vehicle.

Supertype/subtype models are important in object orientation. This relationship is the foundation for inheritance—one of the essential and most powerful features of object orientation. Attributes of “vehicle,” in the above example, are inherited by all the other elements, and attributes of “land vehicle” are inherited by all of the elements in its subtypes. Object orientation has taken this relationship and extended it to more complex forms of inheritance than just attribute inheritance: The lower layer may also inherit functions (or operations, or “methods,” as they are sometimes called) and the behavior of the supertypes.

Supertype/subtype relationships are also referred to as generalization/specialization relationships, class hierarchies, inheritance structures, and “is-a” hierarchies. With the supertype/subtype relationship, it is important that the supertype contains all the commonalities of the subtypes. The main use of this relationship is to discover commonalities and to describe them only once, thus reducing redundancy. The structure then allows the lower layers to inherit whatever commonalities have been discovered.

Now that we have defined the supertype/subtype relationship, we can see that the relationship is, in fact, a subtype of the abstraction/detailing relationship. A supertype is an abstraction of its subtypes, and the subtypes are detailed instances of the supertype. So, all supertype/subtype relationships are also abstraction/detailing relationships, but the converse is not true: Not all abstraction/detailing relationships are supertype/subtype relationships, because not all abstraction/detailing relationships follow the “is-a” principle. For example, a process on a data flow diagram and its child diagram are an abstraction/detailing pair, but it is not true that a child diagram “is-a” parent process.

3.6.4 Controlling/Controlled Relationship in Models

This relationship distinguishes between up and down by having the upper layers control elements of the lower layers. Other terms used for this relationship are the control hierarchy, or the is-boss-of/is-supervised-by relationship. Sometimes, we simply say that the higher element uses the lower elements. The higher layer must have knowledge of the lower layer but the lower layer—that is, the one being used—does not necessarily have to know anything about the boss. In terms of client/server models, the client is the boss that delegates work to the server; the server provides certain services that are performed whenever a client asks for them.

In the requirements method, the control model controls processes in the process model, by activating or deactivating them. In the architecture method, we can model client/server behavior to avoid iterative cycles between architecture modules.

Structured Design, the software design method, provides another example of a controlling/controlled relationship. In its main graphical model, the structure chart, a given module invokes (that is, it uses, calls, or controls the execution of) modules in the layer below. Structured Design is one of several methods that can be used in conjunction with the requirements and architecture methods, as described in *Strategies for Real-Time System Specification*, Section 24.3, and in Chapter 4 of this book.

3.6.5 Layered Models Summary

We hope we have succeeded in dispelling the myth that all layered models are built top-down using decomposition, and have shown that this simplistic, one-size-fits-all view of these models is wrong. The four types of relationships in layered models, described above, are distinctly different from each other; they all serve distinct and important roles in system development; and they can be integrated smoothly, where appropriate, with other models, including object-oriented models. Figure 3.18 summarizes the key aspects of the four relationships in and between layers or their elements.

Even though the four relationships are different, it is convenient to have at least one terminology that can be used with all of them. For this purpose, the “family tree” relationship analogy—of parent/child, grandparent/grandchild, and ancestor/descendant—is commonly used. Although close inspection shows that the analogy does not really fit all four of the relationships (for example, children are not decompositions of their parents), these terms sufficiently describe above/below relationships.

We can now enlarge on the statements of Section 2.1.3 “Multiple Hierarchies.” The four layered models we have described can be—and frequently are—used simultaneously to represent different aspects of one system. Using the requirements and architecture methods, the required functional capabilities of a system are captured by the process model—an abstraction/detailing model; the required behavioral capabilities are captured by the control model—a controlling/controlled model; information structures in the system might include supertype/subtype relationships, captured in an entity-relationship model; and the physical

| | Aggregation/ Decomposition | Abstraction/ Detailing | Supertype/ Subtype | Controlling/ Controlled |
|------------------------------|--|---|---|--|
| Aliases | Whole/part; container/content; composed-of; consists-of; assembly/ subassembly/ component | (Erroneously: abstraction/ decomposition) | Generalization/ specialization; class hierarchy; inheritance structure; "is-a" hierarchy | Is-boss-of/ is- supervised- by; uses hierarchy; client/server |
| Downward Usage | Decompose; dismantle | Add detail; specialize | Inherit; specialize | Control |
| Upward Usage | Aggregate; assemble | Make abstract; generalize | Set membership; "is-a"; generalize; categorize | Controlled by |
| Where Used, Roles | Architecture model; object orientation (aggregate objects); requirements and architecture dictionaries | Requirements model; Structured Analysis; nesting in statecharts | Architecture model when used with object orientation; entity- relationship- attribute modeling | Control model (of requirements model); Structured Design; system control structures (occurs within a layer as well as between layers) |
| Purpose of Usage | Physical packaging; information-hiding; defining scope and visibility | Coping with complexity; reducing complexity | Similar to abstraction/ detailing; in addition: inheritance of attributes, functions, behavior | Separation of concerns; creating noncyclic client/server structures; simplifying cooperation |

Figure 3.18: Summary of Relationships in Layered Models.

structure is captured by the architecture model—an aggregation/decomposition model. Thus, to model a single system, not only can we use layered models of the same kind, as described in Section 2.1.3, we can also use layered models of different kinds. This allows us to represent different views of the system separately, but, when done as part of the requirements and architecture methods, the links between these views are carefully maintained.

This highlights the great flexibility of layered models. They are extraordinarily versatile, and allow us to represent just about any facet of systems and system development separately and at any desired level of detail, but with the links to the other facets also represented.

3.7 MODEL FRAMEWORK SUMMARY

Our modeling framework, shown in Figure 3.19—which is derived from Figure 3.7—now combines all the ideas described in this chapter. For most systems, especially larger ones, we exploit the idea that systems come in hierarchies. We have layers of specifications for the system, subsystems, sub-subsystem, and so on. But note that the flows between the layers go both ways—up and down. There is no sequence of development implied in this framework. We build groups of models as we discover subsystems in the hierarchy, and we do so in any order we want.

Forming subsystems is a difficult architectural or design decision, but exploitation of the information/material/energy classifications and of the generic subsystem categorization will guide us along the way. For software developers, we have many more guidelines in Chapter 6.

Each group of models separates the *what* from the *how*: We build separate requirements and architecture models. To help with the transition between *what* and *how* models, we build enhanced requirements models based on the architecture template. Thus, each group of models consists of three separate but related types: requirements, enhanced requirements, and architecture. Note that the arrows between the three models in the subsystems of Figure 3.19 go both ways—again, allowing these models to be developed in any sequence.

The information-hiding categories described in Section 3.6 are helpful for structuring all three models. They are generic enough to be applicable in many different application areas, yet precise enough to give a head start for partitioning. In many large applications, such subsystems have produced flexible, extendable, and maintainable systems. The case study in Part II demonstrates this idea with a specific example from a unique application domain.

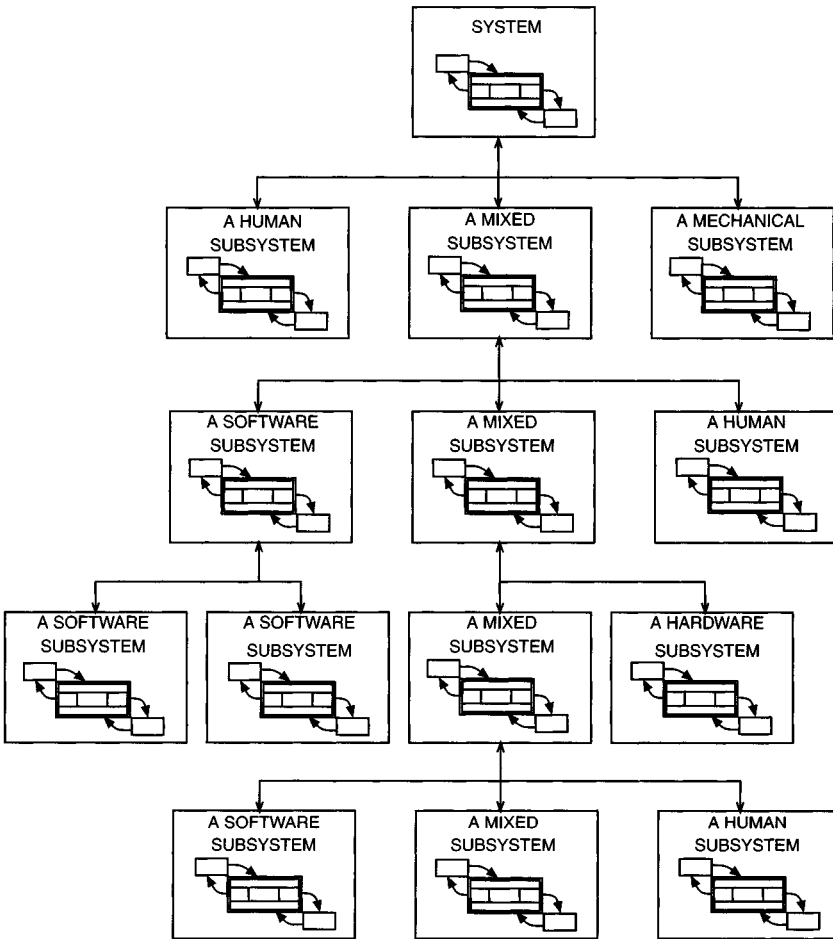


Figure 3.19: The Modeling Framework.

This page intentionally left blank

Index

- abstract interface module* 261, 263–66
- abstraction/detailing* 65–66, 67, 92, 233, 272, 274, 402
- ACD* See *architecture context diagram*
- Ada-based Design Approach for Real-Time Systems (ADARTS)* 255, 402
- AFCD* See *architecture flow context diagram*
- AFD* See *architecture flow diagram*
- aggregation* 272, 402
- aggregation/decomposition* 64–65, 92, 93, 96, 99, 103, 106, 123, 233, 257, 258, 402
 - rules and guidelines* 123
- AICD* See *architecture interconnect context diagram*
- AID* See *architecture interconnect diagram*
- aircraft example* 193, 194
- AIS* See *architecture interconnect specification*
- AMCD* See *architecture message context diagram*
- AMD* See *architecture message diagram*
- AMS* See *architecture module specification*
- architect* 200–204, 273, 402
- architecture* 226, 268–73, 348, 393, 403
 - channel* 204, 357, 393, 403
 - communication model* 92–103, 403
 - context* 286–87, 346
 - context level* 206, 222
 - diagram* 90–92, 107, 227, 403
 - inheritance model* 106–8
 - interface* 343
 - level* 393, 400
 - method* 4, 7, 64, 179, 190–99, 200, 205, 244, 245, 250, 253, 257, 273, 275, 389, 391, 396, 398, 400, 404
 - structure* 274
- architecture context diagram (ACD)* 75, 215, 222, 284, 288, 341–43, 391, 392, 403
- architecture dictionary (AD)* 75, 80, 92, 108–10, 111–12, 177, 197, 233, 362, 376, 385–86, 388, 403
 - rules and guidelines* 110
- architecture flow* 79–82, 83, 84, 88, 94, 177, 206, 222, 223, 229, 239, 249, 250, 357, 362, 373, 383, 388, 393–94, 403
 - rules and guidelines* 110

- architecture flow context diagram (AFCD)* 75, 87, 88–89, 171–72, 222, 223, 224, 284–85, 288, 241–42, 392, 403
 - rules and guidelines* 88–89
- architecture flow diagram (AFD)* 76, 91, 92, 94–96, 97, 98, 103, 108, 110, 111, 172, 174, 228, 229–30, 238ff., 247, 248, 254, 255, 258, 348, 363, 373, 376–77, 380, 381, 393
 - rules and guidelines* 95–96
- architecture inheritance model* 403
- architecture interconnect* 86–87, 240, 249, 250, 362, 393, 403
 - rules and guidelines* 87
- architecture interconnect context diagram (AICD)* 75, 87, 88–90, 171, 223, 225, 288, 341–43, 382, 392, 403
- architecture interconnect diagram (AID)* 76, 103, 104–6, 110, 111, 229–30, 238ff., 245–46, 247, 248, 258, 288, 290, 362, 363, 373–74, 376, 378, 381, 391–92, 403
 - rules and guidelines* 105–6
- architecture interconnect specification (AIS)* 76, 106, 111, 239, 250, 382, 388, 395, 403
 - rules and guidelines* 106
- Architecture Level 1 Model* 220–23, 224–27, 228, 229, 233
- architecture message context diagram (AMCD)* 75, 87, 89, 403
- architecture message diagram (AMD)* 76, 89, 91–92, 97–98, 99–100, 103, 110, 111, 403
 - balancing* 111–12
 - hierarchy style* 99–100
 - network style* 97–98
- architecture model* 64, 69, 72, 74–112, 135, 136, 141, 142, 164, 167, 169, 170, 177, 180, 190, 192, 194, 197, 200, 206, 209, 224–27, 230, 233–34, 257ff., 274, 276, 277, 288–89, 304, 311, 315, 316, 318, 334, 387–88, 391, 396, 398ff., 404
 - allocation of* 224–27
 - balancing* 110–12
 - development* 233
 - requirements model and* 170–78
- architecture module* 76, 77ff., 81, 83, 85, 88, 92, 94ff., 103ff., 108, 112, 138, 172, 174ff., 192, 194, 196, 198, 224ff., 230, 233, 236, 247, 250, 254, 256–57, 259–61, 264, 267–68, 271, 274, 290, 315, 345, 347ff., 352, 357, 362, 366, 377, 382, 387–88, 390, 395, 404
 - concurrent* 376
 - rules and guidelines* 78
- architecture module specification (AMS)* 76, 92, 101–2, 106, 110, 111–12, 126, 198, 212, 230, 250, 252, 303, 375, 378–82, 388, 395, 404
 - rules and guidelines* 101–2
- architecture template* 53–54, 88, 97, 99, 166–69, 171–72, 198, 220–21, 224, 228, 257, 259, 261, 275, 276, 291–95, 331, 393, 394, 404
 - customer specification* 293
 - hardware/software* 292
 - hardware specification* 295
 - rules and guidelines* 168–69
 - software specification* 295
 - system integration* 293
 - system specification* 295
 - variations on* 291–95
- assembly/subassembly/component* 193
- associative entity class* 123–25
 - attributes and* 123–25
 - rules and guidelines* 125
- attribute* 116, 123–25, 162, 164, 233, 267, 308, 404
 - associative entity classes and* 123–25
 - rules and guidelines* 116
- automated tool* 8, 183, 200, 212, 397, 404
- Axiom/Sys* 404, 423
- balancing* 177–78, 404
- black box* 186, 226, 244, 315, 404
- Boehm, Barry W.* 184, 419
- Booch, Grady* 85, 119, 178, 180, 256, 419, 421
- Brooks, Frederick P., Jr.* 256, 419
- bubtangle* 78, 404

- buffer 221, 263
 - input and output 266
 - output-processing 221
 - support 262
 - user interface 221
- capability 315, 404
- Capability Maturity Model (CMM) 181, 404, 420
- cardinality See multiplicity
- CASE 3, 4, 164, 183, 254, 352, 389, 396, 399, 405
- case study 6, 7, 8, 171, 180
- CCD See control context diagram
- CFD See control flow diagram
- channel 76, 198, 288, 342, 349, 362, 381, 405
- Chen, Peter 118, 419
- child 405
 - diagram 65, 96, 98, 105, 138, 158, 172, 174, 196, 221, 226, 227, 234, 364-65, 393
 - model 236
 - processes 233
- child architecture flow diagram 234, 288
- child control flow diagram 158
- child DFD 138, 140, 142
- child process specification 140
- class diagram 28, 33, 73, 90-92, 115, 125-26, 128, 170, 281, 307-8, 397, 405
 - rules and guidelines 126
- class model 270
- Clemens, Paul C. 399, 422
- client 277-78, 281, 405
 - requirements 278, 281, 284
 - system 277-78
- client/server 274
- CMM See Capability Maturity Model
- Coad, Peter 64, 256, 420
- COBOL 269
- CODARTS 255-56, 268, 405
- combinational machine 153-54, 405
- commercial off-the-shelf system (COTS) 252, 405
- component 162, 175, 234, 247, 259, 271-72, 278, 345, 405
 - kinds of 278
- composed-of 193
- Concurrent Design Approach for Real-Time Systems (CODARTS) 255-56, 268, 405
- concurrent
 - engineering 184, 187-90, 291, 405
 - model 188-90
 - process 268
- consists of 143-44
- constraint 301-3
 - design 301-2
 - input/output 301
 - other 303
- content perspective 12
- context 12, 213, 215
 - control vs. 160-61
 - diagram 87, 88, 96, 138, 285, 304-5, 406
- control 406
 - hierarchy 271
 - layers 269
 - model 68, 113-14, 139, 143-56, 166, 169, 170, 206, 233, 267-68, 395, 398-99, 406
 - module 269
 - process 161, 165, 168, 169, 406
- control context diagram (CCD) 156-57, 171-72, 406
 - rules and guidelines 157
- control/data flow diagram (C/DFD) 172, 174, 352
 - enhanced 172
- control flow 106, 114, 129, 131, 144, 151, 155-56, 157, 158, 159-60, 161, 162, 164, 169, 399
 - input 153
 - model 155-56
 - output 153-54
 - rules and guidelines 155-56
 - sources for 144
- control flow diagram (CFD) 131, 156-60, 164, 166, 168, 221, 406
 - data condition 159
 - rules and guidelines 159-160
- controlling/controlled 64, 67-68, 93, 97, 254, 258, 268-69, 271-72, 274, 406
- control specification (CSPEC) 101, 114,

- 144, 146–47, 150, 154ff., 158, 159–60, 161, 164, 166, 169, 174ff., 227, 233, 334, 352, 356, 388, 395, 396, 406
- bar* 156, 158, 159–60, 174, 175, 407
- guide* 175, 407
- COTS 252, 405
- Crosby, Philip B. 36, 420
- CSPEC See *control specification*
- customer 307–8, 342, 407
 - requirements* 376, 387
- customer specification 210, 299–303, 305, 318, 322, 326, 331, 338, 341, 375, 378, 380
 - for groundwater analysis system* 299–303
- DARTS 255
- data 407
 - condition* 160, 166, 169, 407
 - control and* 394–95
 - flow* 383, 399, 407
 - information modeling and* 24
 - model* 267, 395, 407
- database management system (DBMS) 268
- data context diagram (DCD) 131, 135–37, 157, 171, 407
 - rules and guidelines* 136–37
- data flow 83, 106, 129, 131–33, 137ff., 144, 150, 155, 157ff., 164, 169, 249, 395, 399, 407
 - rules and guidelines* 132–133
- data flow diagram (DFD) 65, 110, 131, 137–39, 140, 159–60, 164, 166, 168, 169, 216, 217, 218, 227, 230, 233–35, 275, 278–80, 317–25, 327–29, 334, 337–40, 352, 355–57, 362, 364–67, 390, 395, 397, 407
 - definition of* 137
 - elements of* 137–38
 - rules and guidelines* 138–39
- data flow model 155
- data process 165, 168
- data store 131, 164, 175, 270, 278, 407
 - splitting a* 175
- DBMS 268
- DCD See *data context diagram*
- DDL 268
- decision table (DT) 154, 158, 160, 268, 408
- DeMarco, Tom 130, 420
- Deming, W. Edwards 36, 420
- derived requirement 32, 141, 364, 391, 393, 408
- design 257, 301–2, 408
- Design Approach for Real-Time Systems (DARTS) 255, 301–2, 408
- detailing diagrams 139–41
 - with child DFD* 139–41
 - with PSPEC* 139–41
- development process 181, 183–90, 205, 275, 277–78, 408
 - architecture* 288–90
 - concurrent development process* 184–87
 - evolution of* 183–84
 - model* 278, 291
 - PSARE 6, 181, 413
 - task allocation* 291
- DFD See *data flow diagram*
- dictionary See *architecture dictionary, requirements dictionary*
- domain 273, 288
- DT See *decision table*
- dynamic allocation 177, 242–44, 408
- EDFD See *enhanced data flow diagram*
- Eeles, Peter 271, 420
- encapsulation 256–57
- Engineering of Computer-Based Systems (ECBS) 244, 409
- enhanced control/data flow diagram (EC/DFD) 172
- enhanced data flow diagram (EDFD) 222, 233, 282–83, 284, 291, 332–33, 335–36, 349–51, 357–62, 364, 370–72, 380–81, 409
- enhancement
 - of models* 221, 277, 281–84, 288, 290
 - of requirements* 206, 222, 224–25, 228, 343, 348, 388, 409
 - of requirements model* 166, 220–23, 225–26, 228, 233, 259, 275–76, 289, 311–12, 315, 334, 342, 346, 349–56, 387, 409
- entity 233, 307–8, 409

- model* 112-14, 119-29, 166-67, 169, 170, 174, 180, 304, 307-11, 392
 - modeling* 121, 124
 - specification* 308, 409
- entity class* 32, 33, 73, 116-18, 120, 121, 123-24, 126, 127, 128, 129, 162, 164, 409
 - rules and guidelines* 118
 - specification* 126-27
- environment* 210, 212, 217, 222, 331, 391, 409
- essential model* 222, 271, 276, 284-85, 290, 291, 312, 315, 316-30, 331-40, 346, 409
 - enhancing the* 331-40
- essential requirements* 166, 222, 289, 311, 357, 388
- event* 160, 409
- Existing Sampling Module* 311-15
- external stakeholders* 285
-
- feasibility analyses* 202-3
- field programmable gate arrays (FPGA)* 252
- flight management system* 149-50
- flow* 76, 81, 83-85, 88, 95, 100, 106, 109, 131, 132-33, 134-35, 136, 138, 139, 153, 158, 161, 198, 212, 217, 218, 222, 227, 229, 233, 240, 247, 249, 257, 274, 288, 304, 311, 344, 357, 362, 381, 394, 395, 410
 - controlled* 144
 - data* 144
 - input* 144
 - messages and* 83-85
 - output* 144, 304
 - stores and* 131, 212
- flow diagram* 106, 133, 177, 278, 410
 - child architecture* 96
- function* 254, 410
- functional primitive* 139
-
- Gomaa, Hassan* 255, 420
- groundwater analysis system* 6, 249, 250, 299-388, 391, 396
 - Existing Sampling Module for* 302, 304, 311-15, 343-46, 348, 364-66, 379, 381, 387
 - Groundwater Specimen for* 307-8, 311
 - Sample Analyzer Module for* 344-47, 348-49, 355, 358-59, 370-71, 376-78, 379-80, 381, 387
-
- hardware layers* 251-52
- hardware/software* 236-44, 392
 - configuration* 239, 241
 - interfaces* 239, 392
 - partitioning* 14
- hardware technologies* 244-51
 - chemical* 249
 - electrical* 245-46
 - electromechanical* 247
 - electronic* 247
 - hardware design* 250
 - hydraulic* 248-49
 - manufacturing* 250
 - mechanical* 248
 - mixed technologies* 250-51
 - optical* 249
 - pneumatic* 248-49
- Harel, David* 147, 396, 420
- Hatley, Derek* 3, 180, 420-21
- Hatley/Hruschka/Pirbhai methods (H/H/P)* 6, 7, 136, 299, 315, 389, 401, 410
- Hatley/Pirbhai methods (H/P)* 3, 5, 389, 410
- H/H/P methods* See *Hatley/Hruschka/Pirbhai methods*
- hierarchy* 46, 67, 410
 - aggregation/decomposition* 93, 99
 - class* 67
 - control* 269, 271
 - controlling/controlled* 93
 - is-a* 67
 - networks and* 90-92
 - supertype/subtype* 107
- hospital monitoring system* 6
- H/P methods* See *Hatley/Pirbhai methods*
- Hruschka, Peter* 5, 422
- Humphrey, Watts* 183
-
- IEEE* 244
- "-ilities"* 303, 380
- implementation* 267-68, 410
- INCOSE* 410

- information hiding* 256, 259, 262, 269
 - module 261, 410
- information/material/energy* 55–60, 161, 212, 257
- information model* 206, 233, 261, 410
- inheritance* 76, 77, 86, 107, 256–57, 258, 268, 274, 410
 - structure 274
- inheritance relationship* 85–86, 106, 107, 108
 - rules and guidelines 86
- input and output*
 - flows 82, 138
 - processing 331
- input processing* 187, 410
- interconnect* 87, 206, 212, 228, 230, 234, 245, 249, 250, 344, 382, 392, 393–94, 395, 410
- interface* 236, 277, 288, 315, 331, 410
- International Council on Systems Engineering (INCOSE)* 410
- is-a relationship* 121, 122, 411

- Jackson, Michael* 255, 421
- Jackson System Development (JSD)* 255
- Jacobson, Ivar* 178, 180, 256, 271, 419, 421

- layer* 185–87, 270–71, 280–81, 411
 - application 269
 - architectural 223
 - control 269
 - Deliverable System Development* 186
 - Implementation* 185
 - in model 60–70, 411
 - structure 205, 253
 - in systems 46–47, 185
 - Top System Element* 186

- Maier, Mark* 18, 421, 422
- Manhattan Island example* 27–28
- McConnell, Steve* 16, 421
- McMenamin, Stephen* 66, 112, 180, 421
- message* 76, 79, 81–85, 97, 100, 256–57, 411
 - definition of 81
 - diagrams 98
 - flows and 83–85
 - rules and guidelines 82
- message/call* 98
- message specification* 102–3, 111, 411
 - definition of 102–3
- meta-model* 275–96, 411
- method* 178, 181–83, 200, 209, 250, 256, 389, 390, 397–400, 411
 - See also *Hatley/Hruschka/Pirbhai method (H/H/P)*, *Hatley/Pirbhai method (H/P)*
 - process, tool, and 181–83
- MID* 76, 106, 107–8, 110
- model* 205–74, 275, 278, 280, 311, 344, 346–47, 393, 411
 - See also *architecture model*, *concurrent model*, *control model*, *meta-model*, *requirements model*
 - abstraction/detailing* 68, 192, 197
 - aggregation/decomposition* 64, 69, 192–93, 197
 - client/server* 67
 - context-level* 210
 - environment* 212, 213, 217, 222
 - essential* 222
 - layered* 60–70, 411
 - requirements/architecture/design* 191
 - supertype/subtype* 67
 - system specification* 191–93, 207
- modeling elements* 76–87
 - See also *architecture flow*, *architecture interconnect*, *architecture model*, *flow*, *inheritance relationship*, *message*, and *terminator*
- module* 87, 100, 106, 168, 212, 225, 233, 247, 249, 254, 259, 268, 315, 344, 347, 349, 357, 393, 396, 411
 - See also *architecture module*
 - application* 270
 - control* 269
 - hardware* 236
 - hardware/software* 237
 - software* 236, 240
 - specification* 376
- module inheritance diagram (MID)* 76, 106, 107–8, 110, 411
 - rules and guidelines 108
- multiple architecture models* 174–75

- multiple hierarchies 9
- multiple inheritance 107
- multiplicity 119–20, 121, 124, 128, 305, 308, 412
- Naval Research Laboratory's Software Cost Reduction Method 255
- networks 90–92
- object orientation (OO) 67, 94, 178–79, 256–58, 268, 271, 391, 392, 399, 412
 - architecture and 272
 - constructs of 391
 - methods of 178–79, 257, 271
- object-oriented design (OOD) 255, 256–57, 412
- output processing 221, 276, 412
- Page-Jones, Meilir 86, 254, 255, 258, 421
- Palmer, John 102, 421
- Parnas, David 52, 64, 255, 259, 299, 421–22
- partitioning 49, 412
- PAT See process activation table
- patient-monitoring system 6, 209–36, 238, 252
 - architecture 229–33
 - Architecture Level 1 Model 220–23, 224ff., 233
 - context-level model 213–18
 - enhanced requirements model 220–23
 - enhancements 228
 - environment 210–13
 - interconnects 228
 - lower-level models 233
 - problem statement 210
 - technology constraints 219–20
- people-made systems 9, 10, 11, 27
- Pirbhai, Imtiaz 413, 420
- Premerlani, William 422
- primitive 164, 413
- process 16, 131, 133, 137, 138, 139, 154, 174, 176, 177, 181–83, 200, 212, 221, 233, 249, 250, 271, 277, 284, 395, 398, 399, 413
 - See also Process for System Architecture and Requirements Engineering (PSARE)
 - activators 154, 395
 - definition of 133
 - method 413
 - methods, tools, and 181–83
 - model 113–14, 130–42, 150, 166, 167, 170, 180, 206, 233, 267, 270, 395, 398, 399, 413
 - module 250
 - rules and guidelines 133
 - for system development 16
 - process activation table (PAT) 146, 154, 268, 396, 413
 - Process for System Architecture and Requirements Engineering (PSARE) 6, 181, 413
 - processing view 18, 20–21
 - processors 230
 - processor view 18, 19–20
 - process specification (PSPEC) 139–43, 144, 150, 161, 164, 166, 169, 174, 175, 196, 212, 216, 217, 221, 233, 267, 311, 314, 318–25, 327–30, 334, 335–40, 364–65, 367, 376, 388, 393ff., 400, 413
 - rules and guidelines 142–43, 394
 - Programmable Logic Arrays (PLAs) 20
 - programming language 268, 269
 - project coordination 203
 - prototype 202, 413
 - PSARE 6, 181, 413
 - PSPEC See process specification
 - push and pull indicators 81, 82, 83–85, 94, 95, 97, 99
- quality 36–37
- quick-ticketing system (QTS) 7
- RD See requirements dictionary
- real-time (RT) 48, 398–99, 413
- Rechtin, Eberhardt 18, 37, 422
- relationship 118, 121, 123–25, 129, 162, 164, 267, 307–8, 311, 413
 - abstraction/detailing 65–66, 92
 - aggregation/decomposition 64–65, 92, 123
 - binary 118–19
 - class 91
 - controlling/controlled 67–68
 - generalization/specialization 67

- is-a* 121, 122
- is-boss-of/is-supervised-by* 67
- multiplicity* 119–20
- name* 119–20
- n-ary* 128
- parent/child* 198
- rules and guidelines* 120–21
- specification* 127–28, 413
- subclass/superclass* 92, 121
- superclass/subclass* 121
- supertype/subtype* 66–67, 85
- symbols* 118–20
- required*
 - capability* 210, 219, 387, 413
 - constraint* 387, 414
- requirements* 24–40, 226, 228, 230, 233, 235, 242, 257, 264, 281, 285, 288–89, 299–303, 304, 307–8, 345–46, 347, 348–49, 380, 387–88, 390, 393, 414
 - See also *requirements model*
 - aggregation/decomposition of* 33
 - capabilities and* 26–27, 33, 39
 - categorizing* 39–40, 202
 - class diagram* 28ff., 32, 34, 36
 - constraints* 26, 27, 33, 34, 39, 301–3
 - core* 267
 - customers and* 24
 - decomposing* 202
 - defined* 26–28
 - derived* 30–31, 39, 194–97, 198, 202, 393
 - detailing* 30–31, 38–39, 194–97
 - enhancement and allocation* 194, 202, 267
 - entity class* 38–39
 - essential* 167–68, 222, 266, 393
 - external* 37–38, 185
 - feasibility analysis of* 38, 202
 - functional* 26, 138
 - gathering of* 30, 37–38, 202
 - hiding* 264, 266–67
 - integrity analysis of* 38, 202
 - management of* 37–40, 202
 - managers and* 24, 25
 - performance* 27, 32–33, 164, 196, 375, 387, 414
 - primitive* 32–34, 194, 196
 - for quality* 36–37
 - sources of* 24–26
 - specification* 299
 - stakeholders* 24ff., 200, 266
 - standards as source for* 24, 25
 - sub/supertyping relationship and* 34
 - for timing* 164–65, 375, 377
 - users and* 24, 25, 331
- requirement context diagram* 172, 214, 277, 284, 290, 304–5, 341
- requirement entity class* 28, 38–39
- requirements/architecture relationships*
 - See also *superbubbles, traceability*
 - balancing* 177–78
 - scope differences* 171–72
- requirements dictionary (RD)* 75, 113, 129, 131, 132, 155, 162–70, 177, 218, 221, 233, 311, 376, 383–84, 388, 414
- requirements-driven systems* 22
- requirements flow* 250
- requirements method* 4, 7, 160–61, 179, 190–99, 200, 205, 244, 250, 257, 273, 275, 389, 391, 392, 396, 398, 400, 414
- requirements model* 28–36, 65, 72, 84, 112–61, 162ff., 169–70, 171, 174, 177, 180, 190, 192, 194, 197, 200, 206, 209, 218, 221, 222, 224, 233–34, 249, 250, 261, 267, 270, 275, 279, 304, 311, 315, 316, 318, 347, 366, 387, 391ff., 398ff.
 - architecture model and* 170–78
 - balancing* 141, 166
 - components of* 169
 - is-a relationship* 34
 - summary of* 169–70
- Robertson, James* 66, 180, 422
- Robertson, Suzanne* 66, 180, 422
- Rumbaugh, James* 178, 256, 421, 422
- SA* See *Structured Analysis*
- SD* See *Structured Design*
- sequential machine* 147–48, 149–50, 151–53, 414
 - defined* 147–48
 - other representations of* 151–53
- Sheard, Sarah* 200, 422
- Shenhar, Aaron J.* 189, 422
- Shlaer, S.* 256, 422

- Society of Concurrent Engineering (SOCE)* 422
- software architecture* 259, 272–72
- software design* 269–70
 - methods* 254, 258
- software development* 84–85, 254–58, 393
 - history of* 254–58
- software layers* 253–73
- specifications* 311, 315, 414
- spiral model* 184
- stakeholder* 289, 414
- starvation of bubbles* 139, 414
- state* 415
- state chart* 154, 158, 160, 396, 415
- state machine* 152
- state transition* 165, 415
- state transition diagram (STD)* 146, 147, 148, 149, 150, 151, 268, 396, 415
- state transition matrix (STM)* 146, 415
- state transition table (STT)* 146, 415
- state/event matrix* 152, 415
- state/state matrix* 152, 415
- STG See *Structured Technology Group*
- store* 131, 134–35, 137, 138, 157, 162, 164, 168, 174, 176, 177, 212, 227, 233, 257, 397, 415
 - flows and* 131, 387
 - rules and guidelines* 134–35
 - symbol for* 134–135, 397
- Strategies for Real-Time System Specification 3, 4, 5, 6, 60, 68, 79, 84, 106, 144, 146, 147, 151, 164, 165, 181, 214, 389, 397
- StructSoft 415
- structure* 64, 97
 - aggregation/decomposition* 103, 106
 - assembly/subassembly/component* 64
 - container/content* 64
 - controlling/controlled* 97
 - message/call* 98
 - whole/part* 64
- Structured Analysis (SA)* 34, 47, 65–66, 73, 92, 96, 110, 130, 131, 135, 136, 141, 209, 397, 398, 400, 415
- Structured Design (SD)* 47, 68, 86, 102, 254–55, 258, 268–69, 415
- structured English* 142, 394, 415
- structured methods* 178–79, 415
- Structured Technology Group (STG)* 416
- subtype/supertype* 122, 256, 416
- superbubbles* 171, 172–75, 176, 213–14, 225–27, 228, 230, 233–34, 291, 349, 364, 390, 416
 - rules and guidelines* 174–75
 - traceability and* 176–77
- superclass/subclass* 121–23
 - rules and guidelines* 122–23
- supertype/subtype* 64, 66–67, 85, 107, 121, 416
- system* 10–40, 205, 416
 - See also *system development*
 - approach* 9
 - architect* 200–204, 273, 402
 - architecture* 49, 74, 224, 245, 281, 290, 395
 - artifacts* 5
 - categories of* 17–18
 - classification* 23
 - complexity of* 17–18
 - components of* 10
 - concept of* 9
 - engineer* 200–204, 214
 - engineering* 290, 416
 - errors* 15
 - hierarchies* 11–13, 14
 - life cycle* 15, 182
 - module specification* 375
 - networks* 14–15
 - people-made* 9, 10, 11, 27
 - properties of* 12
 - requirements* 24–40, 49
 - requirements context* 216
 - requirements driven* 22
 - role of* 200–204
 - specification of* 20, 280, 388
 - stakeholders* 281
 - views of* 18–24
- system development* 4, 16, 181–204, 275–96, 318, 348, 389, 416
 - essential model of* 277–81
 - meta-model for* 276
 - nature of* 183–90

- overview of 275–76
 - patient-monitoring system example 209–36
 - process and methods 16, 190–99
 - process, methods, and tools 181–83
- Taylor, David A.* 271, 422
- terminator* 76, 78–79, 87, 88–89, 109, 131, 135, 136, 137, 144, 156, 157, 204, 214, 222–23, 277, 284–85, 288, 290, 341–42, 349, 392, 417
- definition of 78
 - examples of 277–78, 285, 288
 - naming rules 79
 - symbol for 78
- testing* 187, 417
- timing specification (TSPEC)* 166, 177–78, 196, 222, 304, 305–7, 346, 377, 381, 392, 417
- tools* 181–83, 200, 423
- traceability* 176–77, 185, 196, 197–99, 226, 236, 393, 417
- traceability matrix* 101, 176–77, 194, 197, 198, 228, 230–33, 234, 242, 244, 352–54, 366, 368–69, 380, 388, 390, 393, 417
- rules and guidelines 176–77
- trade-off study* 202, 347, 348, 376, 377, 382, 417
- training* 423
- TSPEC* See *timing specification*
- Turbocase/Sys* 418, 423
- Unified Modeling Language (UML)* 7, 85, 118, 121, 125, 178, 179, 180, 418
- use case* 180, 418
- user* 394
- user interface* 89, 221, 259, 261, 264–65, 269, 271–73, 276, 290, 418
- user-interface processing* 331
- views of a system* 18–24
- activity level 19, 23–24
 - intelligence level 19, 22–23
 - processing 18, 19–20
 - processor 18, 20–21
 - what/how 18, 21–22
- waterfall model* 183–84, 188, 189, 190
- Weinberg, Daniela* 43, 422
- Weinberg, Gerald M.* 43, 422
- what/how classification* 49–54, 142
- architecture model and 50
 - requirements model and 50
 - separation of 50–51, 70
- whole/part* 64, 193
- www.psare.com* 7, 36, 391, 423
- Yourdon, Edward* 254, 420, 423