

Requirements Engineering: Frameworks for Understanding

R.J. Wieringa

Faculty of Mathematics and Computer Science

Vrije Universiteit

Amsterdam

©Wiley 1996-2006

Contents

1	Introduction	1
1.1	Computer-based Systems	1
1.2	System Development Methods	3
1.3	The Structure of the Book	4
1.4	Methods, Techniques, Heuristics	5
1.5	Bibliographical Remarks	6
I	An Analysis of Product Development	7
2	Systems	9
2.1	Introduction	9
2.2	System Boundary	9
2.2.1	The observability of systems	9
2.2.2	System boundary and modularity	12
2.3	System Structure	13
2.3.1	Subsystems and aspect systems	13
2.3.2	A hierarchy of system levels	14
2.4	System Behavior	16
2.4.1	System state	16
2.4.2	System transactions	17
2.4.3	System behavior	19
2.4.4	System properties	20
2.5	System Function	22
2.5.1	Products	22
2.5.2	Functions of computer-based systems	23
2.6	The Why, the What and the How	24
2.7	The Universe of Discourse of Computer-based Systems	26
2.8	Summary	27
2.9	Exercises	28
2.10	Bibliographical Remarks	30
3	Product Development	33
3.1	Introduction	33
3.2	Client-oriented and Market-oriented Development	33

3.3	The Product Life Cycle	36
3.3.1	Product development	36
3.3.2	Product implementation	39
3.3.3	Product evolution	40
3.3.4	The regulatory cycle	41
3.4	Product Engineering	42
3.4.1	The engineering cycle	42
3.4.2	Example of an application of the engineering cycle	45
3.5	System Modeling	48
3.5.1	Current system modeling and UoD modeling	48
3.5.2	The empirical cycle	49
3.5.3	Comparison of the empirical and engineering cycles	52
3.6	A Framework for Product Development Methods	52
3.6.1	The specification of needs, product behavior and product decomposition	53
3.6.2	Disentangling requirements and other ambiguities	56
3.6.3	Other framework dimensions	58
3.6.4	Application of the framework	60
3.7	Summary	61
3.8	Exercises	62
3.9	Bibliographical Remarks	64
4	Requirements Specifications	71
4.1	Introduction	71
4.2	Product Objectives	71
4.3	Behavior and Property Specifications	73
4.4	A Framework for Behavior Specifications	75
4.5	Desirable Properties of a Requirements Specification	76
4.6	Summary	78
4.7	Exercises	79
4.8	Bibliographical Remarks	79
II	Requirements Engineering Methods	81
5	ISAC Change Analysis and Activity Study	83
5.1	Introduction	83
5.2	Change Analysis	84
5.2.1	Make a list of problems	84
5.2.2	List problem owners	86
5.2.3	Analyze the problems	87
5.2.4	Make activity model of current business	91
5.2.5	Analyze goals	95
5.2.6	Define change needs	96
5.2.7	Generate change alternatives	97
5.2.8	Make activity model of desired situations	97
5.2.9	Evaluate alternatives	98
5.2.10	Choose an alternative	98

5.3	Activity Study	100
5.3.1	Decomposition into information subsystems	100
5.3.2	Analysis of information subsystems	101
5.3.3	Coordination of information subsystems	102
5.4	Methodological Analysis	102
5.4.1	The place of ISAC in the development framework	102
5.4.2	Activity modeling	105
5.4.3	Participation	106
5.5	Exercises	106
5.6	Bibliographical Remarks	107
6	Information Strategy Planning	109
6.1	Introduction	109
6.2	The Structure of ISP	111
6.3	Analysis of Business Strategy	113
6.3.1	Analyze business mission	113
6.3.2	Analyze business goals and objectives	115
6.3.3	Analyze business problems	116
6.3.4	Analyze critical success factors	116
6.4	Determination of Information Architecture	117
6.4.1	The function decomposition tree: structure and meaning	118
6.4.2	The function decomposition tree: construction heuristics	120
6.4.3	The entity model: structure and meaning	122
6.4.4	The entity model: construction and validation heuristics	126
6.5	Identification of Business Areas	129
6.6	Methodological Analysis	131
6.6.1	The place of Information Engineering in the development framework	131
6.6.2	Function decomposition	132
6.6.3	Entity models	133
6.6.4	Information Engineering and ISAC	134
6.7	Exercises	135
6.8	Bibliographical Remarks	136
7	The Entity-Relationship Approach I: Models	137
7.1	Introduction	137
7.2	Entities, Values and Attributes	138
7.2.1	ER entities	138
7.2.2	Attributes	138
7.2.3	Values	138
7.2.4	Null values	139
7.3	Types and Existence	139
7.3.1	Intension and extension	139
7.3.2	Representation	140
7.3.3	Existence	142
7.4	Entity Identification	142
7.4.1	The importance of identification	142
7.4.2	Identifiers	143

7.4.3	Surrogates	144
7.4.4	Keys	144
7.5	Relationships	145
7.5.1	Identity and existence	145
7.5.2	Representations	147
7.6	Cardinality Constraints	148
7.6.1	Representations	148
7.6.2	Special cardinalities	150
7.7	The <i>is_a</i> Relationship	155
7.8	Methodological Analysis	155
7.8.1	The place of ER models in the behavior specification framework	155
7.8.2	Cardinality constraints	156
7.8.3	Constraints on the UoD and constraints on the system	156
7.9	Summary	157
7.10	Exercises	158
7.11	Bibliographical Remarks	159
8	The Entity-Relationship Approach II: Methods	165
8.1	Introduction	165
8.2	Methods to Find an ER Model	165
8.2.1	Natural language analysis	167
8.2.2	Natural language analysis of transactions	169
8.2.3	Entity analysis of transactions	170
8.2.4	Case study: the library	170
8.3	Methods to Evaluate an ER Model	177
8.3.1	Entity/value checks	177
8.3.2	Entity/link checks	179
8.3.3	Specialization check	181
8.3.4	Elementary sentence check	182
8.3.5	Population check	182
8.3.6	Derivable relationship check	183
8.3.7	Minimal arity checks	184
8.3.8	Creation/deletion check	185
8.3.9	Cross-checking	185
8.3.10	View integration	186
8.3.11	Navigation check	186
8.4	Transformation to a Database System Schema	187
8.5	Methodological Analysis	189
8.5.1	The place of the ER method in the development framework	189
8.5.2	Modeling and engineering	190
8.5.3	Natural language analysis in NIAM	191
8.6	Summary	192
8.7	Exercises	192
8.8	Bibliographical Remarks	193

9	Structured Analysis I: Models	195
9.1	Introduction	195
9.2	Components of a Data Flow Model	195
9.3	Interaction Between the System and its Environment	197
9.3.1	External entities	197
9.3.2	Time and temporal events	199
9.3.3	Event recognition	199
9.3.4	Perfect technology	200
9.4	Data Stores and the ER Diagram	201
9.4.1	Data stores	201
9.4.2	The ER diagram	202
9.5	Data Flows	206
9.6	Data Transformations	208
9.6.1	Specification by DF diagram	208
9.6.2	Minispecs	211
9.7	Methodological Analysis	214
9.7.1	The place of DF models in the specification framework	214
9.7.2	Transactions	217
9.7.3	Transformation decomposition	218
9.8	Summary	220
9.9	Exercises	221
9.10	Bibliographical Remarks	222
10	Structured Analysis II: Methods	223
10.1	Introduction	223
10.2	Methods to Find a DF Model	223
10.2.1	Essential system modeling	223
10.2.2	Event partitioning	227
10.2.3	Process analysis of transactions	228
10.2.4	Case study: The Circulation Desk	229
10.3	Methods to Evaluate a DF Model	236
10.3.1	Walkthroughs and inspections	236
10.3.2	Simulation and animation	237
10.3.3	Minimality principles	237
10.3.4	Determinism	239
10.3.5	Vertical balancing	239
10.3.6	Horizontal balancing and data usage	239
10.4	System Decomposition	240
10.5	Methodological Analysis	241
10.5.1	Essential system modeling	241
10.5.2	Event partitioning and data flow orientation	243
10.5.3	Specification of user procedures	243
10.6	Summary	243
10.7	Exercises	244
10.8	Bibliographical Remarks	245

11 Jackson System Development I: Models	247
11.1 Introduction	247
11.1.1 Jackson Structured Programming	247
11.1.2 From JSP to JSD	249
11.2 The Structure of JSD Models	250
11.3 The UoD Model	252
11.3.1 JSD entities and attributes	252
11.3.2 Life cycles	253
11.3.3 Parallelism and communication	257
11.4 The System Model	261
11.4.1 Process communication in the system network	261
11.4.2 The specification of function processes	267
11.5 Methodological Analysis	271
11.5.1 The place of JSD models in the behavior specification framework	271
11.5.2 Transactions	271
11.5.3 Function specification	273
11.5.4 Entity concepts	274
11.5.5 Communication	274
11.6 Summary	275
11.7 Exercises	276
11.8 Bibliographical Remarks	276
12 Jackson System Development II: Methods	279
12.1 Introduction	279
12.2 Case Study: the Document Circulation System	279
12.2.1 Allocate actions to entity types	282
12.2.2 Specify JSD entity types and actions	285
12.2.3 Specify life cycle of JSD entities	286
12.2.4 Specify action effects	286
12.2.5 Context errors	290
12.2.6 System functions	290
12.3 Evaluation Methods for UoD Models	294
12.4 Implementation	294
12.5 Methodological Analysis	294
12.5.1 Separation of UoD modeling from function specification	294
12.5.2 JSD and Information Engineering	295
12.5.3 JSD and structured development	295
12.5.4 Modeling social processes	296
12.6 Summary	296
12.7 Exercises	297
12.8 Bibliographical Remarks	297
III Method Integration and Strategy Selection	299
13 A Framework for Requirements Engineering I: Models	301
13.1 Introduction	301

13.2	The Specification of Product Objectives	301
13.3	A Framework for Behavior Specifications	305
13.3.1	UoD models	305
13.3.2	Models of system behavior	307
13.3.3	Relationship between UoD models and SuD models	309
13.4	Integrated JSD-ER Models of the UoD	309
13.4.1	Entities	312
13.4.2	Using relationships to remember common actions	312
13.4.3	Reducing common actions by adding relationships	313
13.4.4	The transaction decomposition table	314
13.5	Integrated JSD-ER-DF Models of a System	315
13.5.1	A JSD-ER-DF initial system model	315
13.5.2	A JSD-ER-DF model of system functions	316
13.5.3	The transaction decomposition table	320
13.6	Modularization Principles	322
13.6.1	Object-oriented and Von Neumann modularization	322
13.6.2	UoD-oriented modularization	323
13.7	Integrating System Models with Environment Models	326
13.8	Summary	327
13.9	Exercises	328
13.10	Bibliographical Remarks	328
14	A Framework for Requirements Engineering II: Methods	333
14.1	Introduction	333
14.2	Starting Points	335
14.3	Finding a Behavior Specification	336
14.4	Evaluating a Behavior Specification	338
14.5	Requirements Engineering as Negotiation about Meanings	340
14.5.1	Negotiation in discovery and engineering	340
14.5.2	Implicit conceptual models and the ultimate communication breakdown	341
14.6	Summary	344
14.7	Exercises	344
14.8	Bibliographical Remarks	345
15	Development Strategies	347
15.1	Introduction	347
15.2	Process Management	348
15.3	Top-down Development Strategies	351
15.3.1	The linear strategy	352
15.3.2	The splashing waterfall strategy	354
15.3.3	The V-strategy	354
15.4	A Framework for Concurrent Development	358
15.5	Throw-away Prototyping	362
15.6	Strategies for Phased Development	363
15.6.1	Incremental development	364
15.6.2	Evolutionary development	365
15.6.3	Experimental development	366

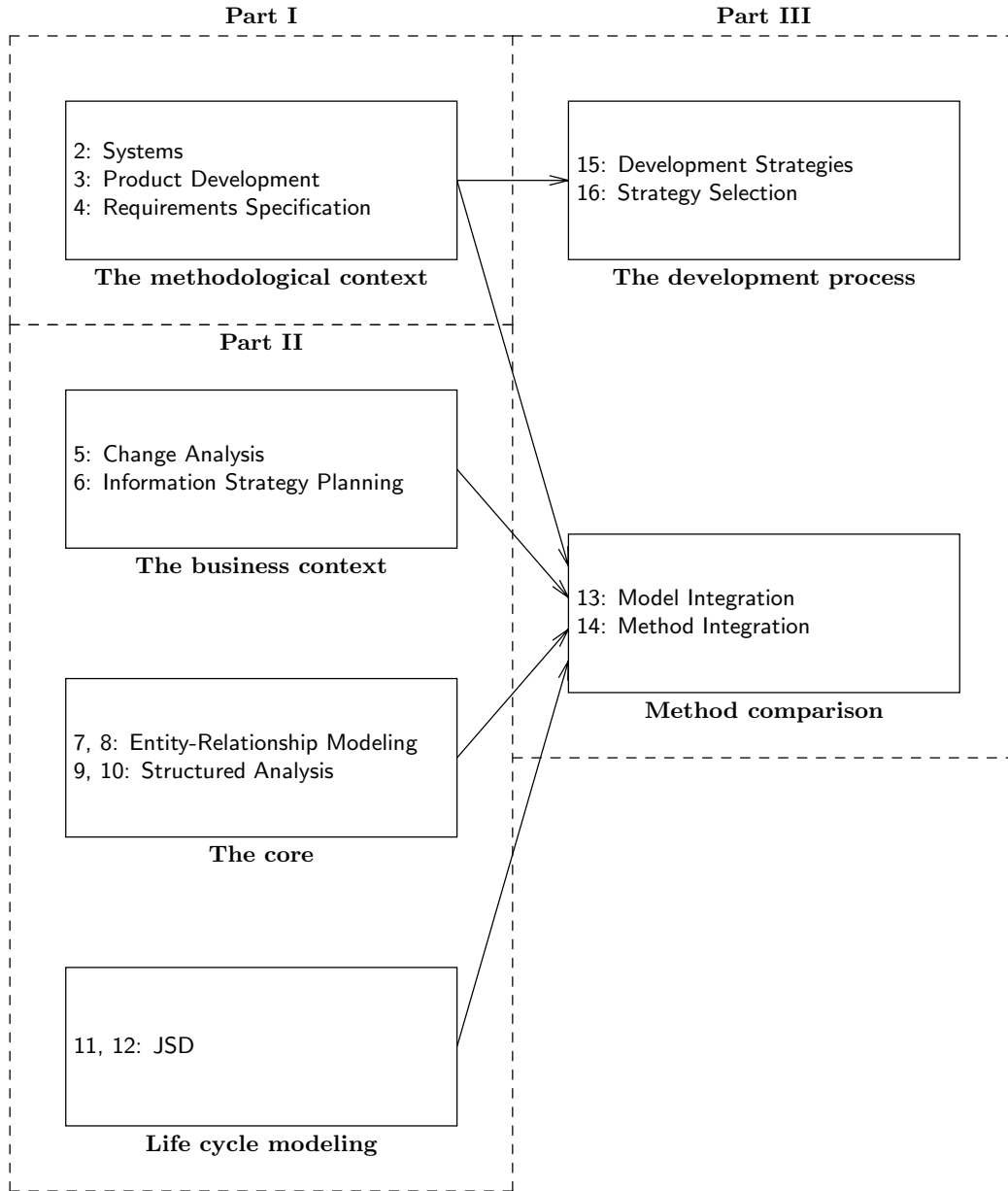
15.7 Rational Reconstruction of the Development Process	368
15.8 Summary	369
15.9 Exercises	370
15.10 Bibliographical Remarks	371
16 Selecting a Development Strategy	377
16.1 Introduction	377
16.2 The Spiral Model for Software Development	377
16.2.1 Structure	377
16.2.2 Strategies	379
16.2.3 Strategy selection heuristics	380
16.2.4 Evaluation of the spiral model	382
16.3 Euromethod	383
16.3.1 Structure	383
16.3.2 Strategies	386
16.3.3 Situational factors	387
16.3.4 Strategy selection heuristics	388
16.3.5 Evaluation of Euromethod	394
16.4 Summary	394
16.5 Exercises	395
16.6 Bibliographical Remarks	395
A Answers to Selected Exercises	397
B Cases	415
B.1 The Teaching Administration	415
B.2 The University Library	415
C An outline of some development methods	419
C.1 ETHICS	419
C.2 Structured Development	421
C.3 SSADM	421
Bibliography	426
Index	444

Preface

This book is about methods for determining computer system requirements. It is written primarily as an introduction to requirements engineering methods for computer science students, but the text has been organized in such a way that it can also be used by practitioners who want to place their work in a wider context.

Over the past 30 years, a jungle of methods and techniques has grown that can be used at different stages of development, from requirements determination to implementation and maintenance. This jungle is ill-structured in appearance, and students as well as practitioners are at a loss where to look for useful methodological advice. One may wonder if it is worthwhile to hack the methodological jungle at all. The goal of the book is to show that there is a structure in this jungle. The book starts in part I with the definition of a methodological framework that can be used to compare methods. In part II it then analyzes five methods for requirements determination, using this framework, and it ends in part III by collecting the results of these analyses into an integrated framework for requirements engineering methods. The text has the following features.

- Several frameworks for methods are defined. In part I, frameworks for system development and for requirements specifications are defined. In part III, the development framework is extended to a framework for development strategies.
- The development of computer-based systems is viewed as a species of industrial product development. Consequently, the frameworks are borrowed partly from the methodology of industrial product development.
- An engineering approach to system development is emphasized. Features of such an approach are the separation of specification from implementation, rational search for alternatives, and simulation of a solution before implementing it.
- The book focuses on methods for requirements engineering. These methods bridge the often informal world of human desires with the formal world of symbol manipulation. Mistakes are easily made in this task, it is hard to discover them, and the later they are discovered, the harder it is to repair them.
- The chapters on the five methods are written for computer science students without any knowledge of development methods. They present methods to the level of detail needed to do practical work with them, without getting buried in a mass of syntactic details. The chapters include two running case studies and each chapter finishes with exercises. Appendix A contains answers to selected exercises.



- There is an index containing all keywords and defined terms. The number of the page where a term is defined, is printed in boldface.
- As indicated in the figure, the substance of the chapters on the five methods can be read without knowing about the methodological frameworks of part I. The chapters are structured according to these frameworks, but it is not necessary to know these frameworks in order to read the chapters.
- Each chapter about a method ends with a methodological analysis of the method that presupposes the methodological frameworks of part I. This methodological discussion culminates in part III with an indication where and how the discussed methods can be integrated.
- The choice of methods has been very conservative. The methods included in part II are either widely used or they illustrate an interesting methodological point. One premise of this book is that every method contains at least one good idea and that there is no method that contains only good ideas. The approach has been to emphasize the good ideas. For reasons of space, only a fragment of some methods is explained.

It should be clear that this book advocates an eclectic approach to methods. The frameworks defined in this book can be viewed as empty toolboxes, to be filled with tools taken from different methods. These tools are conceptual. Their user should understand their possibilities and limitations, and should know which ones can be combined and which are mutually exclusive alternatives. It is one of the hallmarks of the engineer that he or she keeps an open mind about possibilities and does not choose a particular design too soon. This also applies to the choice of tools: even tools that have been invented long ago and are regarded as “outdated” may be useful.

Object-oriented methods are conspicuously absent from this book. The reason for this is that in order to achieve progress in the field of methods, we should understand and consolidate older methods before we advance to newer methods. A sequence of revolutions in which every revolution obliterates all memory of what has gone before, does not constitute progress. The book accordingly tries to consolidate the good ideas in structured methods. A companion volume currently in preparation (*Requirements Engineering: Semantic, Real-Time and Object-Oriented Methods*), uses the frameworks developed in this book to analyze advanced requirements modeling methods.

Chapters 1 through 14 and the practical work that goes with it can be covered in a one-semester course of 5 credit points (5 full-time weeks of student work) that includes practical work. Chapters 15 and 16 complete the methodological analysis begun in chapter 3 and can be reserved for a consolidation of these ideas in a follow-up course on process issues.

To understand methods, one should practice them. The chapters on the five methods should be accompanied with laboratory work in which students do case studies with these methods. A workbench that can be used to draw the diagrams of the different methods is available for academic and research purposes without fee at ftp site `ftp.cs.vu.nl` in directory `pub/tcm`. The workbench runs on Unix systems. For precise system requirements, refer to the file `README.TCM`.

As illustrated in the figure, the book has been structured in such a way that teachers can choose to omit some chapters.

- After the introductory chapter, chapters 7 to 10 form an introduction to the classical **core** of requirements specification, viz. ER modeling and Structured Analysis.
- To embed the core in its **methodological context**, part I can be used to discuss (software) product development and the framework for development methods used in this book. Each method chapter in part II ends with a methodological analysis, that presupposes knowledge of part I. By skipping these methodological analyses, the methods of part II can be treated without treating part I first.
- To explain the relation between requirements specification and the **business context**, the core can be extended with chapters 5 and 6 on Change Analysis and Information Strategy Planning.
- Jointly, chapters 1 to 10 give a fairly standard introduction to requirements specification. Chapters 11 to 14 treat JSD and method integration. These chapters are more difficult and can be used as part of an optional, more advanced course that focuses on **life cycle modeling** and on **method comparison**.
- Chapters 15 and 16 discuss strategies for the **development process** and can be used in any course that has covered part I of the book.

Parts of chapter 13 have been published in the *Computer Journal*, volume 38, number 1, by Oxford University Press. Thanks are due to five generations of students who, every year, patiently plodded through a version of this manuscript and took everything seriously that was written in it. Their problems with the text have taught me a lot. This book contains some of the fruits of many stimulating and enjoyable discussions with John-Jules Meyer, Frank Dignum and Hans Weigand about the formalization of system constraints, and with Wiebren de Jonge about the methodology of entity classification and identification. The book also benefited from discussions with and critical comments from Frank Dehne, Marcel Franckson, Remco Feenstra, Hanna Luden, Gunter Saake, Jeroen Scheerder and John Simons. Gaynor Redvers-Mutton of John Wiley & Sons, Inc. showed me how to rephrase my prose as natural language. Ameen Abu-Hanna provided some stimulating and useful comments on the first chapters of the book. Jan Broersen read the entire manuscript and prevented many typos and errors from going into print. Any remaining errors are to be blamed upon me.

Writing this book has been made bearable by the unseizing efforts of Mieke Poelman who, despite a busy career of her own, managed to find the time to keep me from my work. Her unconditional support has given me both the freedom and the strength to finish this project. I dedicate this book to her.

Bilthoven and Amsterdam, December 1995
R.J.W

Chapter 1

Introduction

1.1 Computer-based Systems

This book is about methods to specify requirements for computer-based systems. To focus thoughts, it is useful to identify three groups of computer-based systems: automated information systems, communication systems and control systems. These groups are not disjoint, but each group has particular characteristics. In this section, I give a number of examples of systems in each group and in the next, we turn to methods to develop these systems.

The characterizing feature of computer-based **information systems** is that they store and manipulate large amounts of data.

- A system that registers the current store of items held by an outlet of a supermarket chain is an information system. For each kind of item sold by the outlet, the information system contains, say, a record containing information about the price, the number of items still in store, the supplier of the item, etc. These records are updated when goods are delivered and when the goods are moved from the store to the shop. The system may be connected to point-of-sale terminals with bar code readers, that read the kind and number of products when they are sold and transmit this information to the information system.
- A reservation system for airline tickets is an information system. The system may be distributed over many different travel agencies, that all have concurrent access to the services of the system in real time. The business transactions processed by the reservation system are reservations of flights made by air transport companies.
- A system that helps managers to analyze market trends and to predict possible effects of changes in strategy or of different ways to implement a chosen strategy is an information system. The system collects data from corporate databases, summarizes this into aggregate data, plots trends, and uses econometric models to compute alternative future scenarios.

The point-of-sale system and the reservation system are called *transaction processing systems*, because their main function consists in registering or performing business transactions.

The management support system is an example of a *decision support system*, because it supports management in making strategic decisions. If these systems are used by senior executives, they are also called *executive information system*. In this book, we view all of these systems as examples of information systems.

The characteristic feature of **communication systems** is that they involve heavy communication traffic between nodes that are located at geographically different places. As can be seen from the examples above, some information systems can be classified as communication systems as well. Other examples of communication systems include the following.

- An *electronic data interchange system* (EDI system) is a system that connects information systems of different companies with each other. The EDI system can be set up in such a way that the information systems of all outlets of a supermarket chain can be connected, through an EDI network, with the order processing system of a supplier of dairy products. For example, every Friday before noon, the information systems in the supermarket chain outlets determine the current stock of dairy products, compute or retrieve the expected buying pattern for the next week, and place an order for dairy products at the supplier over the EDI network, to be delivered on Monday morning.
- The INTIS network in the port of Rotterdam connects information systems of transport companies, shipping agents, docks, ship brokers, insurance companies, the Dutch postal services and customs. Movement of goods into and out of the harbor is accompanied by an exchange of messages over this network, that replaces a labor-intensive and error-prone flow of manually written documents.
- Weapons systems typically involve intensive communication between a command center, ground stations, satellites, and remote systems such as aircraft, in a highly distributed environment in which systems must respond in real time.

The characteristic feature of **control systems** is that they respond to events in their environment by sending control messages to the environment. Some communication systems may be classified as control systems as well. Usually, control systems have interfaces to hardware other than computers, they control the behavior of some of this hardware, they must function in real time and there are strict limits on the response time of the system. For this reason, control systems are also called *real-time systems* or *embedded systems*. We will not use these terms, for any system must operate in real time and is embedded in an environment. For example, most administrative systems must perform certain actions before certain hard deadlines, such as the end of the month (salary payment) or the end of the year (financial reporting). All information and communication systems are embedded in a social system, and many must communicate with hardware, just as control systems do. Examples of control systems include the following.

- A computer-based system that controls the barriers at the gate of a parking garage is a control system. The system must be able to sense that a car wants to enter the building, check that the car has permission, raise the barrier, sense that the car has passed and lower the barrier before another car can enter. The system must monitor the number of cars in the building and refuse entry of a car as long as the building is full.
- Another example is an elevator control system that monitors requests for elevator service and directs the elevator cage to the appropriate floors.

- A computer integrated manufacturing control system that monitors the movement of material through a number of machines is an example of a control system.

Having given an idea of the kind of systems that we are interested in, we now turn to the topic of the book, methods to develop these systems.

1.2 System Development Methods

Since, at the end of the 1960s, the idea arose that computer-based systems must be developed in a methodical way, the field of system development methods has been in a state of flux. In the 1960s, system development was mainly concerned with implementation, viewed narrowly as programming. Wider issues such as requirements analysis and system specification were ignored. In the 1970s, a number of methods were introduced that in one way or another left the computer programming level and took these wider issues into account. Several methods for structured analysis and requirements specification came into being, culminating in methods for the structured specification of real-time systems in the mid-1980s. In parallel to this, methods were proposed to specify the meaning of data, such as entity-relationship modeling and these evolved into so-called semantic modeling methods in the early 1980s. By the end of the 1980s, the structured and semantic approaches were followed by an ever growing crop of object-oriented analysis and design methods. The bibliographical remarks in section 1.5 lists references to 26 methods for requirements specification, illustrative for each of the groups just mentioned: structured, semantic and object-oriented methods. This is not an exhaustive list: the actual number of methods in use by practitioners or proposed by researchers runs in the hundreds, if not thousands, if counted world-wide. Clearly, this multitude of methods poses problems for the novice as well as for the experienced practitioner.

- A problem for the novice is that it is not clear which of these methods one should learn, if any. Do the new, object-oriented (OO) methods make other methods obsolete? Can we save time by ignoring the older methods and limit our reading only to the object-oriented methods? Is it possible to understand object-oriented methods without knowing anything about the older methods? Conversely, is it possible to understand current practice after having read only about object-oriented methods?
- The practitioner too wonders what the relation between new object-oriented methods and the older structured ones is. How can methods be evaluated on their effectiveness and efficiency in developing the system that the user really wants? Supposing it is worthwhile to move to a new method, how can this transition best be accomplished? Which method is “best”, according to a set of criteria chosen by management, for a given development project? Is there a way in which a customized method can be built for a development project, using components from existing methods?

These questions revolve around the underlying problem of what the relationships between the different methods — new and old — are. It is the aim of this book to provide analytic frameworks with which to understand current and future methods, and to apply these frameworks to a number of important current methods.

1.3 The Structure of the Book

Although some of the frameworks given in this book apply to the entire development process, we focus on methods for *requirements specification*. The reason for this is that requirements specification is an identifiable and important activity within system development. Requirements specification is an identifiable activity for which many methods have been proposed. Indeed, it is arguable that all development methods listed in section 1.2 deal with requirements at some level of aggregation, and ignore other important topics like system decomposition, integration and testing. The focus on requirements specification can be justified because errors made in requirements become increasingly costly to repair the later we are in development, and are extremely costly to repair after delivery of the system — if they can be repaired at all in that stage.

In order to understand requirements specification methods, we look in part I at the wider context of product development. In chapter 2, we define **systems** as parts of the world that have an observable behavior and an internal structure, and **products** as artificial systems constructed to provide a function to users. In chapter 3, we look at product development, the product life cycle, product evolution and product engineering. Chapter 3 ends with the definition of a framework for product development methods, which allows us to identify the place of requirements specification in product development. The framework also shows what the logical structure of the requirements specification activity is. In chapter 4 we focus on the result of the requirements specification process, and give a framework that tells us what the logical structure of requirements specifications is. The two frameworks are used in part II to analyze five methods for requirements specification.

- In chapter 5, we look at Change Analysis and Activity Study, which are part of the ISAC method for developing information systems.
- In chapter 6, we look at Information Strategy Planning (ISP), which is part of Information Engineering.
- In chapters 7 and 8, we look at the Entity-Relationship (ER) method and at the structure of the specifications produced by this method.
- In chapters 9 and 10, we look at Structured Analysis (SA) and at the structure of the specifications produced by this method.
- In chapters 11 and 12, we look at a part of the Jackson System Development (JSD) method and at the structure of the specifications produced by this method.

In part III, we gather the results of our analyses and fill out the two frameworks. In chapter 13, we compare the structure of software requirements specifications produced by ER modeling, SA, and JSD by placing them in the framework developed in chapter 4. In chapter 14, we summarize the results about finding and evaluating requirements specifications by placing them in the framework developed in chapter 3.

The focus of the two frameworks is on the *logical* structure of requirements specifications and of methods to find and evaluate such specifications. In chapters 15 and 16 we extend our framework to incorporate the temporal dimension. This allows us to define alternative development **strategies** in chapter 15. These are all compatible with the logical framework for development defined in chapter 3, but choose different paths through the logical tasks.

In chapter 16, we conclude the book by discussing the spiral method and Euromethod as ways to select an appropriate strategy for a particular development process.

1.4 Methods, Techniques, Heuristics, Notations and Methodologies

As can be seen from the short overview above, a major element in the approach of this book is the distinction between methods and methodology. Part II of this book is a description of methods, parts I and III contain a methodological analysis of methods. In this section, we define some terms that are used throughout the book.

A **method** is a systematic way of working by which one can obtain a desired result. The desired result may be the specification of a more cost-effective way of operating a business, a specification of product requirements, a specification of the decomposition of a system, a specification of a marketing plan, a specification of a production process, an implemented product, an installed product, etc.

A **technique** is a *recipe* for obtaining a certain result. Since a recipe is a systematic way to obtain a certain result, all techniques are methods. However, not all methods are techniques. Usually, techniques prescribe a way of working in detail, whereas methods need not contain detailed instructions. There are techniques to serve a volleyball, to perform a dance, to bake a pancake, and to write a structured program. Many methods contain techniques to perform particular tasks. Examples of techniques treated in this book are the diagonalization technique of Information Engineering and the technique of transforming an ER model into a relational database schema. Techniques can often be practiced to perfection and in many cases can be automated. When applied to the right problem in the right context, they are guaranteed to deliver the desired result. However, applied outside their proper context, they lead to garbage.

Most methods additionally provide heuristics to help the developer find or evaluate a system specification. A **heuristic** is a problem-solving advice that has proved to lead to a good solution in many cases. Application of a heuristic is not guaranteed to lead to the desired result. Heuristics given by Polya [263] to solve mathematics problems are to look at related problems, to try a more accessible related problem first, to go back to definitions, etc. Examples of heuristics to find a system specification are to analyze natural language descriptions of system behavior, to look at possible use scenarios, to list the events to which the system must respond, etc.

A **notation** is a systematic way to *represent* something. Notations may be linguistic, consisting of textual symbols, or graphical, consisting of diagrams. All methods discussed in this book use diagrams as part of their notation to represent a system. Examples are ER diagrams, data flow diagrams, Jackson process structure diagrams, etc. Most methods supplement the diagram notation with textual notation, in the form of a data dictionary, annotations to diagrams, narrative text, etc.

Methodology is the study of methods. For example, the methodology of empirical science is the study of methods used to discover laws of nature; the methodology of mathematics is the study of methods used to find and prove mathematical truths; and the methodology of engineering is the study of methods used to produce useful artifacts. This book is an example of engineering methodology, in particular of the methodology of building computer-based systems.

1.5 Bibliographical Remarks

Computer-based systems. The engineering of computer-based systems (ECBS) was the subject of a workshop held in Neve-Ilan, Israel in May 1990 [189]. This workshop led to the institution of an IEEE Computer Society task force on ECBS [360], which summarized the state of the practice in this area and identified topics for research. In Europe, the ATMOSPHERE project was launched in 1990, partly funded as an Esprit II technology integration project [245]. Its aim was to contribute to the state of the art in the engineering of computer-based systems. An overview of methodological results of the ATMOSPHERE project is given by Thomé [345].

Introductions to particular kinds of computer-based systems are Davis and Olson [82] and Kendall and Kendall [174] for information systems, and Keen and Scott Morton [171] for decision support systems. Good introductions to control systems are given by Ward and Mellor [354, 355, 227], Hatley and Pirbhay [141] and by Goldsmith [118]. These references include some discussion of distribution and communication aspects.

System development methods. Examples of information system development methods developed in the 1970s and 1980s are ISAC [204], SSADM [14, 91], Information Engineering [217], ETHICS [235] and Multiview [16], which is built from components of other methods. Examples of conceptual modeling methods that have their roots in the 1970s are ER modeling [65], Structured Analysis [84, 108] and SADT [210]. Two methods with their roots in the 1970s, but which were published in the 1980s, are NIAM [244] and JSD [57, 56, 158]. Another development in the 1980s is the advent of semantic modeling methods, such as the Event Model [179, 180], SDM [135], TAXIS [46], and ACM/PCM [52]. Important methods for the development of control systems (real time and embedded) are the Ward-Mellor extension of SA [354, 355, 227] and the Hatley-Pirbhay extension [141]. Goldsmith [118] develops the Ward-Mellor method further and Shumate and Keller [313] integrate the Hatley-Pirbhay method with elements of the Ward-Mellor method. Gomaa developed a family of structured methods for control systems called DARTS, ADARTS and CODARTS [119, 120, 122]. These methods contain useful advice on structuring criteria for control systems. Examples of methods composed of elements of other methods are Information Engineering [213], SSADM [14, 91, 98], and Multiview [16].

In the 1990s, object-oriented methods became the focus of interest. Examples are the Booch method [45], OMT [296], the Shlaer-Mellor method [308, 309], the Coad-Yourdon [68, 377], the Martin-Odell [218], Objectory [163] and the ROOM method [307]. The FUSION method [71] is built from components of other methods, notably OMT and some elements of formal specification. An important part of research in object-oriented methods is concerned with the question whether object-oriented methods can be integrated with older, well-known methods like SA and ER modeling and whether semantic modeling can be made part of object-oriented modeling [6, 19, 304, 353, 363].