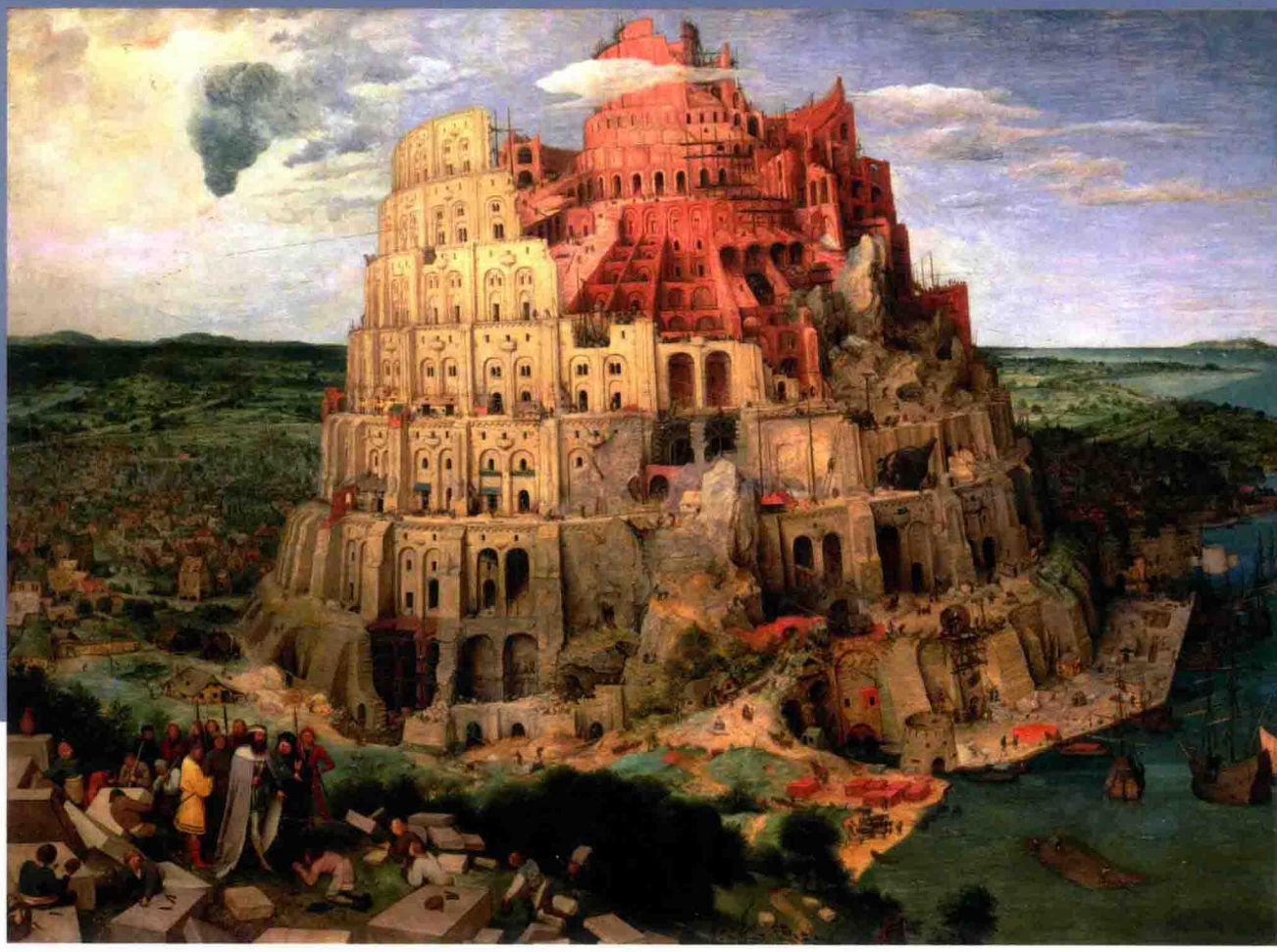
Axel van Lamsweerde



Requirements Engineering

From System Goals to UML Models to Software Specifications

Requirements Engineering

From System Goals to UML Models to Software Specifications

Axel van Lamsweerde

WILEY

A John Wiley and Sons, Ltd., Publication

John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester,
West Sussex PO19 8SQ, England

Telephone (+44) 1243 779777

Email (for orders and customer service enquiries): cs-books@wiley.com
Visit our Home Page on www.wiley.com

Reprinted June 2010

Other Wiley Editorial Offices

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA
Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA
Wiley-VCH Verlag GmbH, Boschstr. 12, D-69469 Weinheim, Germany
John Wiley & Sons Australia Ltd, 42 McDougall Street, Milton, Queensland 4064, Australia
John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01, Jin Xing Distripark, Singapore 129809
John Wiley & Sons Canada Ltd, 6045 Freemont Blvd, Mississauga, Ontario, L5R 4J3, Canada

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Cataloging-in-Publication Data:

Lamsweerde, A. van (Axel)
Requirements engineering : from system goals to UML models to software specifications / Axel van Lamsweerde.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-470-01270-3 (pbk.)
1. Software engineering. 2. Systems engineering. I. Title.
QA76.758.L28 2009
005.1 – dc22

2008036187

Credit for cover image: Kunsthistorisches Museum
Vienna/Bridgeman Art Library; Bruegel/Tower of Babel, 1563

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN 978-0-470-01270-3

Typeset in 10/13pt Sabon by Laserwords Private Limited, Chennai, India
Printed and bound in Great Britain by Bell & Bain, Glasgow

Requirements Engineering

Pour Dominique
Avant tout, et pour tout

Foreword

During the past 60 years of software development for digital computers, development technique, in one of its dimensions, has evolved in a cyclical pattern. At each successive stage, developers recognize that their task has been too narrowly conceived: the heart of the problem is further from the computer hardware than they had thought. Machine code programming led to Fortran, Cobol and Algol, languages aimed at a more problem-oriented way of programming. Then, as program size grew with increasing machine capacities, mere program writing led to notions of program design, software architecture, and software function specification in the large. In a culminating step, functional specification led to a more explicit focus on system requirements – the needs and purposes that the system must serve.

As a wider range of applications embraced more ambitious systems, it gradually became apparent that identifying and capturing system requirements was not an easy task. Published surveys showed that many systems failed because their requirements had not been accurately identified and analysed. Requirements defects proved enormously costly to repair at later stages. By the mid-1980s *requirements engineering* became recognized as an inchoate discipline, or sub-discipline, in its own right. Since the early 1990s it has had its own conferences and a growing literature. It embraces a large spectrum of activities, from discovering the needs and purposes of stakeholders – everyone who would be in any substantial way touched by the proposed system – and resolving the inevitable conflicts, to devising detailed human and computer processes to satisfy the identified system requirements. Requirements engineering must therefore include investigation and analysis of the world in which the requirements have their meaning, because it is in, and through, that world that the computer, executing the developed software, must bring about the desired effects.

Requirements engineering is hard. It is hard to elicit human needs and purposes and to bring them into harmony. Furthermore, there is an inherent dissonance between the quasi-formal world of computer programs – defining the programmed *machine* in each system – and the non-formal *problem world* of the system requirements. Programs can be treated as formal mathematical objects, capable of being proved to satisfy a given formal specification. The

world of system requirements, by contrast, may comprise parts drawn from the natural world, from human participants, from engineered devices, from the built environment, and from every context with which the system interacts directly or indirectly. The problem world is typically heterogeneous and inherently non-formal. We implant the machine in this world, and we program the machine to monitor and control the world through the narrow interface of states and events that it can sense and affect directly. To the extent that the system aims at automation, we are handing to a formally programmed machine a degree of control over a complex and non-formal reality. The common sense and everyday practical knowledge with which human beings can deal with the world is replaced by the formal rules embodied in the software. Even if a system is adaptive, or intelligent or self-healing, its abilities are rigidly bounded by the machine's programmed behaviour, and by the narrow interface which provides it with its sole window on the problem world.

Requirements engineers, then, must be at home in both formal and non-formal worlds, and must be able to bring them together into an effective system. Axel van Lamsweerde has been among the leaders of the requirements engineering discipline since the 1980s, well qualified for this role by a strong background in formal computer science – his early publications were formal papers on concurrency – and an intense practical interest in all aspects of the engineering of computer-based systems. This splendid book represents the culmination of nearly two decades of his research and practical experience. He and his colleagues have developed the KAOS method associated with his name, and have accumulated much practical experience in developing solutions to realistic problems for its customers and users.

As we might expect, the book does what a book on requirements engineering must ideally do. The conceptual basis of the book and the KAOS method is the notion of a *goal*. A goal is a desirable state or effect or property of the system or of any part of it. This notion is flexible enough to apply through many levels of analysis and decomposition, from the largest ambitions of the organization to the detailed specification of a small software module. This book brings together the most formal and the most non-formal concerns, and forms a bridge between them. Its subject matter ranges from techniques for eliciting and resolving conflicting requirements of stakeholders, through the structuring of system goals and their allocation to agents in the machine and the problem world, to the definition and use of a temporal logic by which requirements can be formally analysed and the necessary software functionality derived from the analysis results.

The explanations are copious. Three excellent running examples, drawn from very different kinds of system, illuminate detailed points at every level. Each chapter includes exercises to help the reader check that what has been read has also been understood, and often to stimulate further thought about deeper issues that the chapter has recognized and discussed. Readers who are practising requirements engineers will find the book an excellent source for learning or recapitulating effective approaches to particular concerns. To take one example, there is an incisive discussion – to be found in a section of Chapter 16 – of the task of evaluating alternative architectures and how to set about it. Another example is the crisp account of temporal logic, given in a few pages in the following chapter. This account is so clear and well judged that it can act as both an introduction and a reference tool for all developers who recognize the power and utility of the formalism and want to use it. The comprehensive

bibliographical commentaries in every chapter map out the terrain of what has by now become a substantial literature of the requirements engineering discipline.

The author's friends and colleagues, who know him well, have been waiting for this book with high expectations. These expectations have been amply fulfilled. Readers who have not yet acquainted themselves deeply with the author's work should begin here, immediately. They will not be disappointed.

Michael Jackson,
The Open University and Newcastle University
February 2008

Preface

Requirements Engineering (RE) is concerned with the elicitation, evaluation, specification, analysis and evolution of the objectives, functionalities, qualities and constraints to be achieved by a software-intensive system within some organizational or physical environment.

The requirements problem has been with us for a long time. In their 1976 empirical study, Bell and Thayer observed that inadequate, incomplete, inconsistent or ambiguous requirements are numerous and have a critical impact on the quality of the resulting software. Noting this for different kinds of projects, they concluded that 'the requirements for a system do not arise naturally; instead, they need to be engineered and have continuing review and revision'. Some 20 years later, different surveys over a wide variety of organizations and projects in the United States and in Europe have confirmed the requirements problem on a much larger scale. Poor requirements have been consistently recognized to be the major cause of software problems such as cost overruns, delivery delays, failures to meet expectations or degradations in the environment controlled by the software.

Numerous initiatives and actions have been taken to address the requirements problem. Process improvement models, standards and quality norms have put better requirements engineering practices in the foreground. An active research community has emerged with dedicated conferences, workshops, working groups, networks and journals. Requirements engineering courses have become integral parts of software engineering curricula.

The topic has also been addressed in multiple textbooks. These fall basically into two classes. Some books introduce the requirements engineering process and discuss general principles, guidelines and documentation formats. In general they remain at a fairly high level of coverage. Other books address the use of modelling notations but are generally more focused on modelling software designs. Where are such models coming from? How are they built? What are their underlying requirements? How are such requirements elaborated, organized and analysed? Design modelling books do not address such issues.

In contrast, this book is aimed at presenting a systematic approach to the engineering of high-quality requirements documents. The approach covers the entire requirements lifecycle

and integrates state-of-the-art techniques for requirements elicitation, evaluation, specification, analysis and evolution. *Modelling* plays a central role in this approach. Rich models provide a common interface to the various requirements engineering activities. Such models capture the multiple facets of the system as it is before the software project starts *and* as it should be after project completion. Such a system generally comprises both software components, pre-existing or to be developed, external devices and people playing specific roles. The book's main emphasis is on the *technical* aspects of the requirements engineering process; the socio-psychological issues involved in that process are merely introduced together, with references to dedicated books where such issues are covered in greater depth.

Organization and content

The book is structured in three parts:

- A comprehensive introduction to the fundamentals of requirements engineering (Chapters 1–7).
- A thorough treatment of system modelling in the specific context of engineering requirements (Chapters 8–15).
- A presentation of various forms of reasoning about system models for model building, analysis and exploitation, from semi-formal to qualitative to formal reasoning (Chapters 6–18).

Part I of the book introduces the fundamental concepts, principles and techniques for requirements engineering. It discusses the aim and scope of requirements engineering, the products and processes involved, requirements qualities to aim at and flaws to avoid, the critical role of requirements engineering in system and software engineering, and obstacles to good requirements engineering practices. Key notions such as 'requirement', 'domain property' and 'assumption' are precisely defined. State-of-the-art techniques for supporting the various activities in the requirements lifecycle are reviewed next.

- For *requirements elicitation*, techniques such as interviews, observation or group sessions are based on different forms of interaction with system stakeholders. Other techniques such as scenarios, prototypes or knowledge reuse are based on artefacts to help acquire relevant information.
- For *requirements evaluation*, various techniques may help us manage conflicting concerns, analyse potential risks, evaluate alternative options and prioritize requirements.
- For *requirements documentation*, a wide variety of techniques may help us specify and structure large sets of requirements, from the use of structured natural language to diagrammatic notations to formal specifications.
- For *requirements quality assurance*, we may conduct inspections and reviews, submit queries to a requirements database, validate requirements through animation or verify requirements through formal checks.

- For *requirements evolution*, various techniques are available for change anticipation, traceability management, change control and on-the-fly change at system runtime.

To conclude the first part of the book and introduce the next parts, goal orientation is put forward as a basic paradigm for requirements engineering. Key elements such as goals, agents and scenarios are defined precisely and related to each other.

Part II is devoted to system modelling in the specific context of engineering requirements. It presents a goal-oriented, multiview modelling framework integrating complementary techniques for modelling the system-as-is and the system-to-be.

- AND/OR goal diagrams are used for capturing alternative refinements of functional and non-functional objectives, requirements and assumptions about the system.
- AND/OR obstacle diagrams are used for modelling what could go wrong with the system as modelled, with the aim of deriving new requirements for a more robust system. This view is especially important for mission-critical systems where safety or security concerns are essential.
- UML class diagrams are used for defining and structuring the conceptual objects manipulated by the system and referred to in goal formulations.
- Agent diagrams are used for modelling active system components, such as people playing specific roles, devices and software components, together with their responsibilities and interfaces.
- Operationalization diagrams and UML use cases are used for modelling and specifying the system's operations so as to meet the system's goals.
- UML sequence diagrams and state diagrams are used for modelling the desired system behaviours in terms of scenarios and state machines, respectively.

Each modelling technique is explained separately first, with a strong emphasis on well-grounded heuristics for *model building*. The full system model is obtained from those various views through mechanisms for view integration.

To conclude the second part of the book, a constructive method is presented for elaborating a full, robust and consistent system model through incremental integration of the goal, object, agent, operation and behaviour sub-models. Goals and scenarios drive the elaboration and integration of these sub-models. The elaboration proceeds both top down, from strategic objectives, and bottom up, from operational material available. The requirements document is then generated systematically by mapping the resulting model into some textual format annotated with figures. The document produced preserves the goal-oriented structure and content of the model, and fits prescribed standards if required.

The model-based requirements engineering approach described in Part II, known as *KAOS*, has been developed and refined over more than 15 years of research, tool development and experience in multiple industrial projects. KAOS stands for '**K**eep **A**ll **O**bjectives **S**atisfied'. (*Kaos* happens to be the name of an allegorical movie by the Taviani brothers based on Luigi Pirandello's five tales on the multiple facets of our world.)

Part III reviews goal-based reasoning techniques that support the various steps of this requirements engineering approach. The transition from requirements to software architecture is discussed as well. The analysis techniques fall into three complementary classes:

- Query-based techniques can be used for checking model well-formedness, for managing traceability among model items, and for retrieving reusable model fragments.
- Qualitative and quantitative techniques help evaluate alternative options arising during the requirements engineering process. Such options correspond to alternative goal refinements, responsibility assignments, conflict resolutions or countermeasures to the identified hazards or threats. The evaluation of options is based on the non-functional goals identified in the goal model.
- Formal techniques can be used incrementally and locally, where and when needed, to support goal refinement and operationalization, conflict management, analysis of obstacles to goal achievement, analysis of security threats for countermeasure exploration, synthesis of behaviour models, and goal-oriented model checking and animation. Such techniques require the corresponding goals, operations and domain properties to be specified formally.

Approach

The book presents both a comprehensive state of the art in requirements engineering (Part I) *and* a systematic method for engineering high-quality requirements (Parts II and III), anchored on this state of the art.

Like the method and supporting tools, this book is 'two-button' in nature. The material covering formal methods for requirements engineering is optional and is concentrated near the end of the book; the 'formal button' is mostly pressed in Chapters 17 and 18. Formal techniques are useful in requirements engineering to enforce higher precision in specifications and to support much richer forms of analysis for requirements quality assurance. They turn out to be essential for reasoning about critical goals concerning system safety and security. Formal techniques are, however, mostly hidden from Chapters 1 to 16, even though they are to some extent involved at different places here and there. The aim is to make solid modelling techniques more accessible to a much wider audience. For example, formal refinement patterns are seen in Chapter 18 to produce goal refinements that are provably correct and complete (Section 18.1). They are introduced informally in Chapter 8 to support the critical task of refining goals in a systematic way (see the model-building heuristics in Section 8.8). Similarly, obstacle analysis is handled formally in Chapter 18 but introduced informally in Chapter 9. Extensive experience with students, tutorial attendees and practitioners over the years shows that this way of hiding the underlying mathematical apparatus works remarkably well. Like Molière's Monsieur Jourdain, who is writing prose without being aware of it, they are using temporal logic without really knowing it.

On the other hand, other readers with some background in formal methods might be interested in a more formal treatment of model-based RE from the beginning. Such readers can

press the 'formal button' earlier, as they will have no difficulty in making the hidden formal apparatus visible. The semi-formal techniques and numerous examples presented in Parts II and III can easily be translated into the simple formalism based on temporal logic introduced in Section 4.4.2 and further detailed in Chapter 17.

Unlike many books consisting of a mere exposition of a catalogue of notations and illustrations of their use, this book puts a strong emphasis on *constructive* techniques for building high-quality system models using a coherent subset of notations. A rich variety of heuristic rules is provided that combines model-building strategies, tactics and patterns, common pitfalls and bad smells. Much more than specific notations, what matters here is the quality and usefulness of the models and documents elaborated, and the process according to which such artefacts are built. Experience in teaching modelling for more than 20 years to students and practitioners has convinced us that effective guidance in model building is what is needed most – in the same way as good programming methods, techniques and patterns are known to be much more important than the use of a specific programming language.

Speaking of notations, we will use standard ones wherever we can. In particular, we will see how UML class diagrams, use cases, sequence diagrams and state diagrams can be systematically derived from goal models, and vice versa. The only new notations introduced in the book refer to abstractions that are crucially missing in the UML for requirements engineering; namely, goal diagrams, obstacle diagrams and context diagrams.

The concepts, principles and techniques throughout the book are illustrated by numerous examples from *case studies* to give the reader more concrete insights into how they can be used in practical settings. The wide applicability of the techniques is demonstrated through running examples from completely different domains: an information system, an embedded control system and a distributed collaborative application to be developed as a product family. These running examples arise from simplifications of real systems for library management, train control and meeting scheduling, respectively. The method is also shown in action in the stepwise elaboration of an entire multi-view model of a mine safety control system. The requirements document generated semi-automatically from the latter model is shown in the book's accompanying website.

For more active reading, each chapter ends with a series of exercises, problems and bibliographical notes. Some of the exercises provide additional case studies for more substantial experimentation, in particular in student projects. The bibliographical notes are intended to open the window on past achievements in the field and directions for further study.

A professional modelling tool that supports the goal-oriented RE method in this book is freely accessible to the reader for building limited-size models and requirements documents (http://www.objectiver.com). The tool includes, among other components, a graphical model editor, an HTML generator for navigation and zooming in/out through large models, a model database query engine with pre-defined model consistency checks, and a requirements document generator. The book does not assume that the reader will use this tool. However, playing with it for building models involved in the book's exercises and case studies, and generating requirements documents semi-automatically from the models, will result in more

active and enjoyable learning. As a side effect, further insight will be gained on the benefits of using tools for requirements engineering.

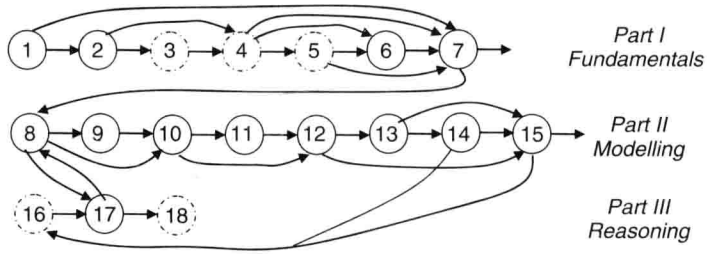
Readership

The book is primarily intended for two categories of reader:

- Students in computing science, information systems or system engineering who need a solid background in techniques for requirements engineering and system modelling – typically, final-year undergraduate or first-year graduate students who take a course on software engineering or a more dedicated course on requirements engineering or system modelling. The book can be used as a supplement to other textbooks for a course on software engineering, or as main support for a one-term course on requirements engineering or system modelling.
- Professional engineers, business analysts, system analysts, consultants or project leaders who, beyond general guidelines, need systematic guidance for elaborating and analysing high-quality requirements.

Parts I and II, covering the fundamentals of requirements engineering and model building, have no real prerequisite. The more advanced techniques in Part III, and Chapters 17 and 18 in particular, assume some elementary background in the logical foundations of computing science together with more analytical reasoning skills.

How to use the book



Reading graph

The material in the book has been organised to meet different needs. Multiple tracks can therefore be followed corresponding to different selections of topics and levels of study. Such tracks define specific paths in the book's reading graph. Arrows in this graph denote reading precedence, whereas dotted circles indicate partial reading of the corresponding chapter by skipping some sections.

- *Track 1: Model-free introduction to RE.* Part I of the book can be used for an RE course with very little modelling. Along this track, students are expected to follow or have followed another course on system modelling. Section 4.3 is provided to summarize popular

modelling notations for RE, defining each of them concisely, highlighting their complementarity and illustrating their use in the running case studies. Optionally, Section 4.4 on formal specification and Section 5.4 on formal verification may be skipped for shorter courses or students with no background in the logical foundations of computing.

- *Track 2: Model-based introduction to RE.* This track is intended for an RE course with substantial coverage of modelling techniques. The material in Part I up to Section 4.2 is taken. Section 4.3 is provided as a contextual entry point to subsequent chapters emphasizing model construction. Section 4.4 (formal specification), Chapter 5 (requirements inspection, validation and verification) and/or Chapter 6 (requirements evolution) are skipped depending on course length or reader focus. The track then proceeds with Chapter 7 and key chapters from Part II; namely, Chapter 8 on goal modelling, Chapter 10 on object modelling, Chapter 12 on operation modelling and Chapter 15 showing how the techniques introduced in these chapters fit together to form a systematic model-building method. Ideally, Chapters 9, 11 and 13 should be included as well to cover risk, responsibility and behaviour models.

- *Track 3: Introduction to early model building for model-driven software engineering.* This track is intended for the RE part of a software engineering course. (I used to follow it for the first third of my SE course.) It consists of Chapter 1, introducing the basics of RE, Chapter 7, introducing system modeling from a RE perspective, and then Chapters 8–13 on model building, concluded by Chapter 15 showing a fully worked-out case study. For shorter coverage, Chapter 11 may be skipped, as key material there is briefly introduced in Chapters 7, 8 and 10 and briefly recalled in Chapter 12.

- *Tracks 4.n: Hybrid RE tracks.* Depending on student profile, teacher interests and course length, multiple selections can be made out of Parts I and II so as to cover essential aspects of RE and model-based RE. Chapter 1 is required in any selection. Typical combinations include Chapter 1, Chapter 2, [Chapter 3], Chapter 4 limited to Sections 4.1 and 4.2, [Chapter 5], [Chapter 6], Chapter 7, Chapter 8, [Chapter 9], Chapter 10, Chapter 12, [Chapter 13] and Chapter 15, where brackets indicate optional chapters. (I have used such combinations on several occasions.)

- *Track 5: The look-ahead formal track.* Students with some background in formal methods do not necessarily have to wait until Part III to see formal modelling and analysis in action. They will have no difficulty making the material in Part II more formal by expressing the specifications and patterns there in the temporal logic language introduced in Section 4.4 and detailed in Chapter 17.

- *Track 6: The advanced track.* A more advanced course on RE, for students who have had an introductory course before, can put more emphasis on analysis and evolution by in-depth coverage of the material in Chapter 3, Section 4.4 in Chapter 4, Chapter 5, Chapter 6, Chapter 9 (if not covered before), Chapter 14, Chapter 16, Chapter 17 and Chapter 18. This track obviously has prerequisites from preceding chapters.

Additional resources

Lecture slides, additional case studies, solutions to exercises and model-driven requirements documents from real projects will gradually be made available on the book's Web site.

Acknowledgement

I have wanted (and tried) to write this book for a long time. This means that quite a few people have been involved in some way or another in the project.

My first thanks go to Emmanuel Letier. The book owes much to our joint work over 10 years. Emmanuel contributed significantly to some of the techniques described in Parts II and III, notably the techniques for agent-based refinement and goal operationalization. In addition to that, he created initial models and specifications for several case studies, examples and exercises in the book. Emmanuel was also instrumental in making some of the pillars of the modelling framework more solid.

Robert Darimont deserves special thanks too. He initiated the refinement pattern idea and provided initial insights on goal conflicts. Later he gave lots of feedback from his daily use of the method and supporting tools in industry. This feedback had a large influence on enhancements, notably through considerable simplification and polishing of the original framework.

Speaking of the original framework, Steve Fickas and Martin Feather had a strong influence on it through their work on composite system design. I still believe that Martin's simple but precise semantics for agent responsibility is the one to rely on.

Many people joined the research staff in the KAOS project and contributed in some way or another. I wish to thank in particular Christophe Damas, Anne Dardenne, Renaud De Landtsheer, Bruno Delcourt, Emmanuelle Delor, Françoise Dubisy, Bernard Lambeau, Philippe Massonet, Cédric Nève, Christophe Ponsard, André Rifaut, Jean-Luc Roussel, Marie-Claire Schayes, Hung Tran Van and Laurent Willemet.

Quite a few students provided valuable feedback from using some of the techniques in their MS thesis or from studying draft chapters. I would like to acknowledge in particular Nicolas Accardo, Pierre-Jean Fontaine, Olivier Haine, Laurent Hermoye, Jonathan Lewis, Florence Massen, Junior F. Monfils, Alessandra de Schrynmakers and Damien Vanderveken.

Many thanks are also due to all those who provided helpful comments and suggestions on earlier drafts of the book, including Alistair Suttcliffe, Klaus Pohl, Steve Fickas, Bill Robinson and the Wiley reviewers. Martin Feather gave substantial feedback on my attempts to integrate his DDP approach in the section on risk analysis. I am also very much indebted to Michael Jackson for taking time to read the manuscript and write such a nice foreword.

Earlier colleagues at Philips Research Labs provided lifetime stimulation for technical precision, highly needed in RE, including Michel Sintzoff, Philippe Delsarte and Pierre-Jacques Courtois. François Bodart at the University of Namur opened a window on the area for me and excited my attraction to real-world case studies.

Writing a book that in places tries to reconcile requirements engineering (RE) and formal methods (FM) is quite a challenge. I am indebted to the many RE researchers and practitioners I met for their scepticism about formal methods, and to the many FM researchers I met for their scepticism about RE as a respectable area of work. Their combined scepticism contributed a great deal to the never-ending quest for the Holy Grail.

Besides the multiple laptops and typesetting systems I used during the painful process of book writing, I would like to acknowledge my cellos and Johann Sebastian Bach's genial suites, which helped me a great deal in recovering from that pain.

Last but not least, the *real* thanks go to Dominique for her unbounded patience and endurance through years and years – she would most probably have written this sort of book three times faster; to Nicolas, Florence and Céline for making me look ahead and for joking about book completion on every occasion; and to Agathe, Inès, Jeanne, Nathan, Louis and Nina for reminding me constantly that the main thing in life cannot be found in books.