Software Requirements
Third Edition

Karl Wiegers and Joy Beatty
Praise for this book

“Software Requirements, Third Edition, is the most valuable requirements guidance you will find. Wiegers and Beatty cover the entire landscape of practices that today’s business analyst is expected to know. Whether you are a veteran of requirements specification or a novice on your first project, this is the book that needs to be on your desk or in your hands.”

—Gary K. Evans, Agile Coach and Use Case Expert, Evanetics, Inc.

“It’s a three-peat: Karl Wiegers and Joy Beatty score again with this third edition. From the first edition in 1999 through each successive edition, the guidance that Software Requirements provides has been the foundation of my requirements consulting practice. To beginning and experienced practitioners alike, I cannot recommend this book highly enough.”

—Roxanne Miller, President, Requirements Quest

“The best book on requirements just got better! The third edition’s range of new topics expands the project circumstances it covers. Using requirements in agile environments is perhaps the most significant, because everyone involved still needs to understand what a new system must do—and agile developers are now an audience who ought to have a good grasp of what’s in this book.”

—Stephen Withall, author of Software Requirement Patterns

“The third edition of Software Requirements is finally available—and it was worth waiting so long. Full of practical guidance, it helps readers identify many useful practices for their work. I particularly enjoy the examples and many hands-on solutions that can be easily implemented in real-life scenarios. A must-read, not only for requirements engineers and analysts but also for project managers.”

—Dr. Christof Ebert, Managing Director, Vector Consulting Services

“Karl and Joy have updated one of the seminal works on software requirements, taking what was good and improving on it. This edition retains what made the previous versions must-have references for anyone working in this space and extends it to tackle the challenges faced in today’s complex business and technology environment. Irrespective of the technology, business domain, methodology, or project type you are working in, this book will help you deliver better outcomes for your customers.”

—Shane Hastie, Chief Knowledge Engineer, Software Education

“Karl Wiegers’s and Joy Beatty’s new book on requirements is an excellent addition to the literature. Requirements for large software applications are one of the most difficult business topics of the century. This new book will help to smooth out a very rough topic.”

—T. Capers Jones, VP and CTO, Namcook Analytics LLC
“Simply put, this book is both a must-read and a great reference for anyone working to define and manage software development projects. In today’s modern software development world, too often sound requirements practices are set aside for the lure of “unencumbered” agile. Karl and Joy have detailed a progressive approach to managing requirements, and detailed how to accommodate the ever-changing approaches to delivering software.”

—Mark Kulak, Software Development Director, Borland, a Micro Focus company

“I am so pleased to see the updated book on software requirements from Karl Wiegers and Joy Beatty. I especially like the latest topic on how to apply effective requirements practices to agile projects, because it is a service that our consultants are engaged in more and more these days. The practical guide and real examples of the many different requirement practices are invaluable.”

—Doreen Evans, Managing Director of the Requirements and Business Analysis Practice for Robbins Gioia Inc.

“As an early adopter of Karl’s classic book, Software Requirements, I have been eagerly awaiting his new edition—and it doesn’t disappoint. Over the years, IT development has undergone a change of focus from large, new, ‘green-field’ projects towards adoption of ready-made off-the-shelf solutions and quick-release agile practices. In this latest edition, Karl and Joy explore the implications of these new developments on the requirements process, with invaluable recommendations based not on dogma but on what works, honed from their broad and deep experience in the field.”

—Howard Podeswa, CEO, Noble Inc., and author of The Business Analyst’s Handbook

“If you are looking for a practical guide into what software requirements are, how to craft them, and what to do with them, then look no further than Software Requirements, Third Edition. This usable and readable text walks you through exactly how to approach common requirements-related scenarios. The incorporation of multiple stories, case studies, anecdotes, and examples keeps it engaging to read.”

—Laura Brandenburg, CBAP, Host at Bridging the Gap

“How do you make a good requirements read better? You add content like Karl and Joy did to address incorporating product vision, tackling agility issues, covering requirements reuse, tackling packaged and outsourced software, and addressing specific user classes. You could take an outside look inside of requirements to address process and risk issues and go beyond just capturing functionality.”

—Donald J. Reifer, President, Reifer Consultants LLC

“This new edition keeps pace with the speed of business, both in deepening the foundation of the second edition and in bringing analysts down-to-earth how-to’s for addressing the surge in agile development, using features to control scope, improving elicitation techniques, and expanding modeling. Wiegers and Beatty have put together a must-read for anyone in the profession.”

—Keith Ellis, President and CEO, Enfocus Solutions Inc., and author of Business Analysis Benchmark
Software Requirements, Third Edition

Karl Wiegers and Joy Beatty
For Chris, yet again. Eighth time’s the charm.

—K.W.

For my parents, Bob and Joanne, for a lifetime of encouragement.

—J.B.
# Contents at a glance

*Introduction*  
*Acknowledgments*

<table>
<thead>
<tr>
<th>PART I</th>
<th>SOFTWARE REQUIREMENTS: WHAT, WHY, AND WHO</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAPTER 1</td>
<td>The essential software requirement 3</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>Requirements from the customer’s perspective 25</td>
</tr>
<tr>
<td>CHAPTER 3</td>
<td>Good practices for requirements engineering 43</td>
</tr>
<tr>
<td>CHAPTER 4</td>
<td>The business analyst 61</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PART II</th>
<th>REQUIREMENTS DEVELOPMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAPTER 5</td>
<td>Establishing the business requirements 77</td>
</tr>
<tr>
<td>CHAPTER 6</td>
<td>Finding the voice of the user 101</td>
</tr>
<tr>
<td>CHAPTER 7</td>
<td>Requirements elicitation 119</td>
</tr>
<tr>
<td>CHAPTER 8</td>
<td>Understanding user requirements 143</td>
</tr>
<tr>
<td>CHAPTER 9</td>
<td>Playing by the rules 167</td>
</tr>
<tr>
<td>CHAPTER 10</td>
<td>Documenting the requirements 181</td>
</tr>
<tr>
<td>CHAPTER 11</td>
<td>Writing excellent requirements 203</td>
</tr>
<tr>
<td>CHAPTER 12</td>
<td>A picture is worth 1024 words 221</td>
</tr>
<tr>
<td>CHAPTER 13</td>
<td>Specifying data requirements 245</td>
</tr>
<tr>
<td>CHAPTER 14</td>
<td>Beyond functionality 261</td>
</tr>
<tr>
<td>CHAPTER 15</td>
<td>Risk reduction through prototyping 295</td>
</tr>
<tr>
<td>CHAPTER 16</td>
<td>First things first: Setting requirement priorities 313</td>
</tr>
<tr>
<td>CHAPTER 17</td>
<td>Validating the requirements 329</td>
</tr>
<tr>
<td>CHAPTER 18</td>
<td>Requirements reuse 351</td>
</tr>
<tr>
<td>CHAPTER 19</td>
<td>Beyond requirements development 365</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PART III</th>
<th>REQUIREMENTS FOR SPECIFIC PROJECT CLASSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAPTER 20</td>
<td>Agile projects 383</td>
</tr>
<tr>
<td>CHAPTER 21</td>
<td>Enhancement and replacement projects 393</td>
</tr>
<tr>
<td>CHAPTER 22</td>
<td>Packaged solution projects 405</td>
</tr>
<tr>
<td>CHAPTER 23</td>
<td>Outsourced projects 415</td>
</tr>
</tbody>
</table>
CHAPTER 24  Business process automation projects  421
CHAPTER 25  Business analytics projects  427
CHAPTER 26  Embedded and other real-time systems projects  439

PART IV  REQUIREMENTS MANAGEMENT

CHAPTER 27  Requirements management practices  457
CHAPTER 28  Change happens  471
CHAPTER 29  Links in the requirements chain  491
CHAPTER 30  Tools for requirements engineering  503

PART V  IMPLEMENTING REQUIREMENTS ENGINEERING

CHAPTER 31  Improving your requirements processes  517
CHAPTER 32  Software requirements and risk management  537

Epilogue  549
Appendix A  551
Appendix B  559
Appendix C  575
Glossary  597
References  605

Index  619
# Contents

*Introduction* ................................................................. xxv

*Acknowledgments* ......................................................... xxxi

## PART I SOFTWARE REQUIREMENTS: WHAT, WHY, AND WHO

### Chapter 1 The essential software requirement 3

- Software requirements defined ....................................... 5
- Some interpretations of “requirement” ............................. 6
- Levels and types of requirements ................................... 7
- Working with the three levels ....................................... 12
- Product vs. project requirements ................................. 14

#### Requirements development and management .............. 15

- Requirements development ........................................... 15
- Requirements management ........................................... 17

#### Every project has requirements ............................... 18

#### When bad requirements happen to good people .......... 19

- Insufficient user involvement ....................................... 20
- Inaccurate planning ................................................... 20
- Creeping user requirements ........................................ 20
- Ambiguous requirements ............................................ 21
- Gold plating ............................................................... 21
- Overlooked stakeholders ............................................ 22

#### Benefits from a high-quality requirements process ...... 22

### Chapter 2 Requirements from the customer’s perspective 25

- The expectation gap .................................................... 26
- Who is the customer? ..................................................... 27

#### The customer-development partnership .................... 29

- Requirements Bill of Rights for Software Customers ........ 31
- Requirements Bill of Responsibilities for Software Customers 33
Chapter 3  Good practices for requirements engineering  43

A requirements development process framework ........................................ 45
Good practices: Requirements elicitation .................................................. 48
Good practices: Requirements analysis .................................................... 50
Good practices: Requirements specification .............................................. 51
Good practices: Requirements validation .................................................. 52
Good practices: Requirements management ............................................. 53
Good practices: Requirements management ............................................. 53
Good practices: Knowledge .................................................................... 54
Good practices: Project management ......................................................... 56
Getting started with new practices ............................................................ 57

Chapter 4  The business analyst  61

The business analyst role ........................................................................... 62
The business analyst's tasks ....................................................................... 63
Essential analyst skills ............................................................................... 65
Essential analyst knowledge ...................................................................... 68
The making of a business analyst ............................................................... 68
  The former user ....................................................................................... 68
  The former developer or tester ................................................................. 69
  The former (or concurrent) project manager .......................................... 70
  The subject matter expert ...................................................................... 70
  The rookie ............................................................................................... 71
The analyst role on agile projects ............................................................... 71
Creating a collaborative team ................................................................. 72
## PART II REQUIREMENTS DEVELOPMENT

### Chapter 5 Establishing the business requirements 77

- Defining business requirements ........................................ 78
  - Identifying desired business benefits .......................... 78
  - Product vision and project scope .................................. 78
  - Conflicting business requirements .............................. 80

- Vision and scope document ....................................... 81
  - 1. Business requirements ...................................... 83
  - 2. Scope and limitations ...................................... 88
  - 3. Business context ........................................ 90

- Scope representation techniques .............................. 92
  - Context diagram ........................................... 92
  - Ecosystem map ............................................. 94
  - Feature tree ............................................... 95
  - Event list ................................................ 96

- Keeping the scope in focus .................................... 97
  - Using business objectives to make scoping decisions ..... 97
  - Assessing the impact of scope changes ...................... 98

- Vision and scope on agile projects .......................... 98

- Using business objectives to determine completion ...... 99

### Chapter 6 Finding the voice of the user 101

- User classes ...................................................... 102
  - Classifying users .......................................... 102
  - Identifying your user classes .............................. 105

- User personas .................................................. 107

- Connecting with user representatives .................... 108

- The product champion ......................................... 109
  - External product champions ................................ 110
  - Product champion expectations ......................... 111
  - Multiple product champions ............................. 112
Chapter 7  Requirements elicitation

Requirements elicitation techniques ................................................. 121
  Interviews .................................................................................. 121
  Workshops .................................................................................. 122
  Focus groups .............................................................................. 124
  Observations ............................................................................... 125
  Questionnaires ........................................................................... 127
  System interface analysis .............................................................. 127
  User interface analysis ................................................................. 128
  Document analysis ....................................................................... 128
Planning elicitation on your project ............................................... 129
Preparing for elicitation ................................................................. 130
Performing elicitation activities ..................................................... 132
Following up after elicitation ......................................................... 134
  Organizing and sharing the notes .................................................. 134
  Documenting open issues .............................................................. 135
Classifying customer input ............................................................ 135
How do you know when you’re done? .......................................... 138
Some cautions about elicitation .................................................... 139
Assumed and implied requirements ............................................... 140
Finding missing requirements ....................................................... 141

Chapter 8  Understanding user requirements

Use cases and user stories .............................................................. 144
The use case approach .................................................................. 147
  Use cases and usage scenarios ..................................................... 149
  Identifying use cases ................................................................. 157
### Exploring use cases
- Validating use cases
- Use cases and functional requirements
- Use case traps to avoid

Benefits of usage-centric requirements

### Chapter 9  Playing by the rules
A business rules taxonomy
- Facts
- Constraints
- Action enablers
- Inferences
- Computations
- Atomic business rules

Documenting business rules

Discovering business rules

Business rules and requirements

Tying everything together

### Chapter 10  Documenting the requirements
The software requirements specification
- Labeling requirements
- Dealing with incompleteness
- User interfaces and the SRS

A software requirements specification template
- 1. Introduction
- 2. Overall description
- 3. System features
- 4. Data requirements
- 5. External interface requirements
- 6. Quality attributes
- 7. Internationalization and localization requirements
- 8. [Other requirements]
Appendix A: Glossary .................................................. 199
Appendix B: Analysis models ........................................ 199
Requirements specification on agile projects .................. 199

Chapter 11 Writing excellent requirements ................................. 203
  Characteristics of excellent requirements ......................... 203
  Characteristics of requirement statements ...................... 204
  Characteristics of requirements collections .................... 205
Guidelines for writing requirements .................................. 207
  System or user perspective ........................................ 207
  Writing style ...................................................... 208
  Level of detail ..................................................... 211
  Representation techniques ....................................... 212
  Avoiding ambiguity .............................................. 213
  Avoiding incompleteness ....................................... 216
Sample requirements, before and after .............................. 217

Chapter 12 A picture is worth 1024 words ................................. 221
  Modeling the requirements ...................................... 222
  From voice of the customer to analysis models ................ 223
  Selecting the right representations .............................. 225
  Data flow diagram .............................................. 226
  Swimlane diagram ............................................... 230
  State-transition diagram and state table ....................... 232
  Dialog map ......................................................... 235
  Decision tables and decision trees .............................. 239
  Event-response tables ......................................... 240
  A few words about UML diagrams ................................ 243
  Modeling on agile projects ..................................... 243
  A final reminder .................................................. 244
## Chapter 13 Specifying data requirements 245
- Modeling data relationships ........................................ 245
- The data dictionary .................................................... 248
- Data analysis ............................................................ 251
- Specifying reports ...................................................... 252
  - Eliciting reporting requirements ...................................... 253
  - Report specification considerations .................................. 254
  - A report specification template ........................................ 255
- Dashboard reporting .................................................. 257

## Chapter 14 Beyond functionality 261
- Software quality attributes ........................................... 262
- Exploring quality attributes ........................................... 263
- Defining quality requirements ......................................... 267
  - External quality attributes ........................................ 267
  - Internal quality attributes .......................................... 281
- Specifying quality requirements with Planguage .................... 287
- Quality attribute trade-offs ........................................... 288
- Implementing quality attribute requirements .......................... 290
- Constraints ............................................................... 291
- Handling quality attributes on agile projects ......................... 293

## Chapter 15 Risk reduction through prototyping 295
- Prototyping: What and why ........................................... 296
- Mock-ups and proofs of concept .................................... 297
- Throwaway and evolutionary prototypes .............................. 298
- Paper and electronic prototypes ...................................... 301
- Working with prototypes .............................................. 303
- Prototype evaluation .................................................. 306
Contents

Risks of prototyping .................................................. 307
Pressure to release the prototype ................................. 308
Distraction by details .............................................. 308
Unrealistic performance expectations ........................ 309
Investing excessive effort in prototypes ..................... 309
Prototyping success factors ......................................... 310

Chapter 16 First things first: Setting requirement priorities 313
Why prioritize requirements? ..................................... 314
Some prioritization pragmatics .................................. 315
Games people play with priorities .............................. 316
Some prioritization techniques ................................. 317
In or out ............................................................... 318
Pairwise comparison and rank ordering .................... 318
Three-level scale .................................................. 319
MoSCoW ........................................................... 320
$100 ................................................................. 321
Prioritization based on value, cost, and risk ............... 322

Chapter 17 Validating the requirements 329
Validation and verification ........................................ 331
Reviewing requirements ............................................ 332
The inspection process .......................................... 333
Defect checklist ................................................... 338
Requirements review tips ....................................... 339
Requirements review challenges ............................. 340
Prototyping requirements ....................................... 342
Testing the requirements ......................................... 342
Validating requirements with acceptance criteria .... 347
Acceptance criteria ............................................... 347
Acceptance tests ................................................... 348
Chapter 18 Requirements reuse

Why reuse requirements? ........................................ 352
Dimensions of requirements reuse .......................... 352
  Extent of reuse ........................................... 353
  Extent of modification ...................................... 354
  Reuse mechanism .......................................... 354
Types of requirements information to reuse .................... 355
Common reuse scenarios ........................................ 356
  Software product lines ..................................... 356
  Reengineered and replacement systems .................... 357
  Other likely reuse opportunities ......................... 357
Requirement patterns ............................................ 358
Tools to facilitate reuse ....................................... 359
Making requirements reusable .................................. 360
Requirements reuse barriers and success factors .............. 362
  Reuse barriers ............................................. 362
  Reuse success factors ...................................... 363

Chapter 19 Beyond requirements development

Estimating requirements effort .................................. 366
From requirements to project plans ......................... 369
  Estimating project size and effort from requirements .... 370
  Requirements and scheduling ............................ 372
From requirements to designs and code .................... 373
  Architecture and allocation .............................. 373
  Software design .......................................... 374
  User interface design .................................... 375
From requirements to tests .................................... 377
From requirements to success .................................. 379
PART III  REQUIREMENTS FOR SPECIFIC PROJECT CLASSES

Chapter 20  Agile projects  383
  Limitations of the waterfall ...........................................384
  The agile development approach ..................................385
  Essential aspects of an agile approach to requirements ..........385
    Customer involvement .............................................386
    Documentation detail ..............................................386
    The backlog and prioritization .................................387
    Timing ....................................................................387
    Epics, user stories, and features, oh my! ......................388
    Expect change ............................................................389
  Adapting requirements practices to agile projects ................390
  Transitioning to agile: Now what? .................................390

Chapter 21  Enhancement and replacement projects  393
  Expected challenges .....................................................394
  Requirements techniques when there is an existing system ....394
  Prioritizing by using business objectives ..........................396
    Mind the gap .............................................................396
    Maintaining performance levels ....................................397
  When old requirements don’t exist ....................................398
  Which requirements should you specify? ............................398
  How to discover the requirements of an existing system .......400
  Encouraging new system adoption ....................................401
  Can we iterate? ..............................................................402

Chapter 22  Packaged solution projects  405
  Requirements for selecting packaged solutions ....................406
  Developing user requirements .........................................406
  Considering business rules .............................................407
  Identifying data needs ..................................................407
Defining quality requirements ........................................ 408
Evaluating solutions .................................................. 408
Requirements for implementing packaged solutions .......... 411
Configuration requirements ....................................... 411
Integration requirements .......................................... 412
Extension requirements ........................................... 412
Data requirements .................................................... 412
Business process changes ......................................... 413
Common challenges with packaged solutions ................... 413

Chapter 23  Outsourced projects 415
Appropriate levels of requirements detail ....................... 416
Acquirer-supplier interactions .................................... 418
Change management .................................................. 419
Acceptance criteria ................................................... 420

Chapter 24  Business process automation projects 421
Modeling business processes ...................................... 422
Using current processes to derive requirements ............... 423
Designing future processes first ................................... 424
Modeling business performance metrics ....................... 424
Good practices for business process automation projects .... 426

Chapter 25  Business analytics projects 427
Overview of business analytics projects ....................... 427
Requirements development for business analytics projects .... 429
Prioritizing work by using decisions ............................. 430
Defining how information will be used ......................... 431
Specifying data needs ............................................. 432
Defining analyses that transform the data ....................... 435
The evolutionary nature of analytics ............................. 436
### Chapter 26  Embedded and other real-time systems projects  439

- System requirements, architecture, and allocation  440
- Modeling real-time systems  441
  - Context diagram  442
  - State-transition diagram  442
  - Event-response table  443
  - Architecture diagram  445
  - Prototyping  446
- Interfaces  446
- Timing requirements  447
- Quality attributes for embedded systems  449
- The challenges of embedded systems  453

### PART IV  REQUIREMENTS MANAGEMENT

### Chapter 27  Requirements management practices  457

- Requirements management process  458
- The requirements baseline  459
- Requirements version control  460
- Requirement attributes  462
- Tracking requirements status  464
- Resolving requirements issues  466
- Measuring requirements effort  467
- Managing requirements on agile projects  468
- Why manage requirements?  470

### Chapter 28  Change happens  471

- Why manage changes?  471
- Managing scope creep  472
- Change control policy  474
- Basic concepts of the change control process  474
A change control process description ........................................... 475
  1. Purpose and scope .......................................................... 476
  2. Roles and responsibilities .................................................. 476
  3. Change request status ....................................................... 477
  4. Entry criteria ................................................................. 478
  5. Tasks ............................................................................... 478
  6. Exit criteria ....................................................................... 479
  7. Change control status reporting .......................................... 479
Appendix: Attributes stored for each request .................................. 479

The change control board ............................................................. 480
  CCB composition .................................................................. 480
  CCB charter ......................................................................... 481
  Renegotiating commitments .................................................... 482

Change control tools ................................................................. 482
Measuring change activity .......................................................... 483
Change impact analysis .............................................................. 484
  Impact analysis procedure ..................................................... 484
  Impact analysis template ...................................................... 488

Change management on agile projects ......................................... 488

**Chapter 29** Links in the requirements chain ................................ 491
  Tracing requirements ............................................................ 491
  Motivations for tracing requirements ....................................... 494
  The requirements traceability matrix ....................................... 495
  Tools for requirements tracing ............................................... 498
  A requirements tracing procedure ......................................... 499
  Is requirements tracing feasible? Is it necessary? ..................... 501

**Chapter 30** Tools for requirements engineering ........................ 503
  Requirements development tools ........................................... 505
    Elicitation tools .................................................................. 505
    Prototyping tools ................................................................ 505
    Modeling tools .................................................................. 506
Requirements management tools ........................................... 506
Benefits of using an RM tool .............................................. 506
RM tool capabilities ........................................................... 508
Selecting and implementing a requirements tool .................... 510
Selecting a tool ............................................................... 511
Setting up the tool and processes .................................... 511
Facilitating user adoption ................................................... 513

PART V IMPLEMENTING REQUIREMENTS ENGINEERING

Chapter 31 Improving your requirements processes 517
How requirements relate to other project processes ................. 518
Requirements and various stakeholder groups .................... 520
Gaining commitment to change ........................................ 521
Fundamentals of software process improvement .................... 522
Root cause analysis .......................................................... 524
The process improvement cycle ......................................... 526
  Assess current practices ............................................... 526
  Plan improvement actions ........................................... 527
  Create, pilot, and roll out processes ................................ 528
  Evaluate results ........................................................... 529
Requirements engineering process assets ............................. 530
  Requirements development process assets ....................... 531
  Requirements management process assets ....................... 532
Are we there yet? ............................................................. 533
Creating a requirements process improvement road map ........... 535

Chapter 32 Software requirements and risk management 537
Fundamentals of software risk management ......................... 538
Elements of risk management ........................................... 538
Documenting project risks ............................................... 539
Planning for risk management ........................................... 542
Introduction

Despite decades of industry experience, many software organizations struggle to understand, document, and manage their product requirements. Inadequate user input, incomplete requirements, changing requirements, and misunderstood business objectives are major reasons why so many information technology projects are less than fully successful. Some software teams aren’t proficient at eliciting requirements from customers and other sources. Customers often don’t have the time or patience to participate in requirements activities. In many cases, project participants don’t even agree on what a “requirement” is. As one writer observed, “Engineers would rather decipher the words to the Kingsmen’s 1963 classic party song ‘Louie Louie’ than decipher customer requirements” (Peterson 2002).

The second edition of *Software Requirements* was published 10 years prior to this one. Ten years is a long time in the technology world. Many things have changed in that time, but others have not. Major requirements trends in the past decade include:

- The recognition of business analysis as a professional discipline and the rise of professional certifications and organizations, such as the International Institute of Business Analysis and the International Requirements Engineering Board.

- The maturing of tools both for managing requirements in a database and for assisting with requirements development activities such as prototyping, modeling, and simulation.

- The increased use of agile development methods and the evolution of techniques for handling requirements on agile projects.

- The increased use of visual models to represent requirements knowledge.

So, what *hasn’t* changed? Two factors contribute to keeping this topic important and relevant. First, many undergraduate curricula in software engineering and computer science continue to underemphasize the importance of requirements engineering (which encompasses both requirements development and requirements management). And second, those of us in the software domain tend to be enamored with technical and process solutions to our challenges. We sometimes fail to appreciate that requirements elicitation—and much of software and systems project work in general—is primarily a human interaction challenge. No magical new techniques have come along to automate that, although various tools are available to help geographically separated people collaborate effectively.
We believe that the practices presented in the second edition for developing and managing requirements are still valid and applicable to a wide range of software projects. The creative business analyst, product manager, or product owner will thoughtfully adapt and scale the practices to best meet the needs of a particular situation. Newly added to this third edition are a chapter on handling requirements for agile projects and sections in numerous other chapters that describe how to apply and adapt the practices in those chapters to the agile development environment.

Software development involves at least as much communication as it does computing, yet both educational curricula and project activities often emphasize the computing over the communication aspect. This book offers dozens of tools to facilitate that communication and to help software practitioners, managers, marketers, and customers apply effective requirements engineering methods. The techniques presented here constitute a tool kit of mainstream “good practices,” not exotic new techniques or an elaborate methodology that purports to solve all of your requirements problems. Numerous anecdotes and sidebars present stories—all true—that illustrate typical requirements-related experiences; you have likely had similar experiences. Look for the “true stories” icon, like the one to the left, next to real examples drawn from many project experiences.

Since the first edition of this book appeared in 1999, we have each worked on numerous projects and taught hundreds of classes on software requirements to people from companies and government agencies of all sizes and types. We’ve learned that these practices are useful on virtually any project: small projects and large, new development and enhancements, with local and distributed teams, and using traditional and agile development methods. The techniques apply to hardware and systems engineering projects, too, not just software projects. As with any other technical practice, you’ll need to use good judgment and experience to learn how to make the methods work best for you. Think of these practices as tools to help ensure that you have effective conversations with the right people on your projects.

**Benefits this book provides**

Of all the software process improvements you could undertake, improved requirements practices are among the most beneficial. We describe practical, proven techniques that can help you to:

- Write high-quality requirements from the outset of a project, thereby minimizing rework and maximizing productivity.
- Deliver high-quality information systems and commercial products that achieve their business objectives.
- Manage scope creep and requirements changes to stay both on target and under control.
- Achieve higher customer satisfaction.
- Reduce maintenance, enhancement, and support costs.

Our objective is to help you improve the processes you use for eliciting and analyzing requirements, writing and validating requirements specifications, and managing the requirements throughout the software product development cycle. The techniques we describe are pragmatic and realistic. Both of us have used these very techniques many times, and we always get good results when we do.

Who should read this book

Anyone involved with defining or understanding the requirements for any system that contains software will find useful information here. The primary audience consists of individuals who serve as business analysts or requirements engineers on a development project, be they full-time specialists or other team members who sometimes fill the analyst role. A second audience includes the architects, designers, developers, testers, and other technical team members who must understand and satisfy user expectations and participate in the creation and review of effective requirements. Marketers and product managers who are charged with specifying the features and attributes that will make a product a commercial success will find these practices valuable. Project managers will learn how to plan and track the project’s requirements activities and deal with requirements changes. Yet another audience is made up of stakeholders who participate in defining a product that meets their business, functional, and quality needs. This book will help end users, customers who procure or contract for software products, and numerous other stakeholders understand the importance of the requirements process and their roles in it.

Looking ahead

This book is organized into five parts. Part I, “Software requirements: What, why, and who,” begins with some definitions. If you’re on the technical side of the house, please share Chapter 2, on the customer-development partnership, with your key customers. Chapter 3 summarizes several dozen “good practices” for requirements development.
and management, as well as an overall process framework for requirements development. The role of the business analyst (a role that also goes by many other names) is the subject of Chapter 4.

Part II, “Requirements development,” begins with techniques for defining the project’s business requirements. Other chapters in Part II address how to find appropriate customer representatives, elicit requirements from them, and document user requirements, business rules, functional requirements, data requirements, and nonfunctional requirements. Chapter 12 describes numerous visual models that represent the requirements from various perspectives to supplement natural-language text, and Chapter 15 addresses the use of prototypes to reduce risk. Other chapters in Part II present ways to prioritize, validate, and reuse requirements. Part II concludes by describing how requirements affect other aspects of project work.

New to this edition, Part III contains chapters that recommend the most effective requirements approaches for various specific classes of projects: agile projects developing products of any type, enhancement and replacement projects, projects that incorporate packaged solutions, outsourced projects, business process automation projects, business analytics projects, and embedded and other real-time systems.

The principles and practices of requirements management are the subject of Part IV, with emphasis on techniques for dealing with changing requirements. Chapter 29 describes how requirements tracing connects individual requirements both to their origins and to downstream development deliverables. Part IV concludes with a description of commercial tools that can enhance the way your teams conduct both requirements development and requirements management.

The final section of this book, Part V, “Implementing requirements engineering,” helps you move from concepts to practice. Chapter 31 will help you incorporate new requirements techniques into your group’s development process. Common requirements-related project risks are described in Chapter 32. The self-assessment in Appendix A can help you select areas that are ripe for improvement. Two other appendices present a requirements troubleshooting guide and several sample requirements documents so you can see how the pieces all fit together.

Case studies

To illustrate the methods described in this book, we have provided examples from several case studies based on actual projects, particularly a medium-sized information system called the Chemical Tracking System. Don’t worry—you don’t need to know anything about chemistry to understand this project. Sample discussions among
participants from the case studies are sprinkled throughout the book. No matter what kind of software your organization builds, you’ll be able to relate to these dialogs.

From principles to practice

It’s difficult to muster the energy needed for overcoming obstacles to change and putting new knowledge into action. As an aid for your journey to improved requirements, most chapters end with several “next steps,” actions you can take to begin applying the contents of that chapter immediately. Various chapters offer suggested templates for requirements documents, a review checklist, a requirements prioritization spreadsheet, a change control process, and many other process assets. These items are available for downloading at the companion content website for this book:

http://aka.ms/SoftwareReq3E/files

Use them to jump-start your application of these techniques. Start with small improvements, but start today.

Some people will be reluctant to try new requirements techniques. Use this book to educate your peers, your customers, and your managers. Remind them of requirements-related problems encountered on previous projects, and discuss the potential benefits of trying some new approaches.

You don’t need to launch a new development project to begin applying better requirements practices. Chapter 21 discusses ways to apply many of the techniques to enhancement and replacement projects. Implementing requirements practices incrementally is a low-risk process improvement approach that will prepare you for the next major project.

The goal of requirements development is to accumulate a set of requirements that are good enough to allow your team to proceed with design and construction of the next portion of the product at an acceptable level of risk. You need to devote enough attention to requirements to minimize the risks of rework, unacceptable products, and blown schedules. This book gives you the tools to get the right people to collaborate on developing the right requirements for the right product.
Errata & book support

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site at:

http://aka.ms/SoftwareReq3E/errata

If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at mspinput@microsoft.com.

Please note that product support for Microsoft software is not offered through the addresses above.

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://aka.ms/tellpress

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

Stay in touch

Let’s keep the conversation going! We’re on Twitter: http://twitter.com/MicrosoftPress.
Acknowledgments

Writing a book like this is a team effort that goes far beyond the contributions from the two authors. A number of people took the time to review the full manuscript and offer countless suggestions for improvement; they have our deep gratitude. We especially appreciate the invaluable comments from Jim Brosseau, Joan Davis, Gary K. Evans, Joyce Grapes, Tina Heidenreich, Kelly Morrison Smith, and Dr. Joyce Statz. Additional review input was received from Kevin Brennan, Steven Davis, Anne Hartley, Emily Iem, Matt Leach, Jeannine McConnell, Yaaqub Mohamed, and John Parker. Certain individuals reviewed specific chapters or sections in their areas of expertise, often providing highly detailed comments. We thank Tanya Charbury, Mike Cohn, Dr. Alex Dean, Ellen Gottesdiener, Shane Hastie, James Hulgan, Dr. Phil Koopman, Mark Kulak, Shirley Sartin, Rob Siciliano, and Betsy Stockdale. We especially thank Roxanne Miller and Stephen Withall for their deep insights and generous participation.

We discussed aspects of the book’s topics with many people, learning from their personal experiences and from resource materials they passed along to us. We appreciate such contributions from Jim Brosseau, Nanette Brown, Nigel Budd, Katherine Busey, Tanya Charbury, Jennifer Doyle, Gary Evans, Scott Francis, Sarah Gates, Dr. David Gelperin, Mark Kerin, Norm Kerth, Dr. Scott Meyers, John Parker, Kathy Reynolds, Bill Trosky, Dr. Ricardo Valerdi, and Dr. Ian Watson. We also thank the many people who let us share their anecdotes in our “true stories.”

Numerous staff members at Seilevel contributed to the book. They reviewed specific sections, participated in quick opinion and experience surveys, shared blog material they had written, edited final chapters, drew figures, and helped us with operational issues of various sorts. We thank Ajay Badri, Jason Benfield, Anthony Chen, Kell Condon, Amber Davis, Jeremy Gorr, Joyce Grapes, John Jertson, Melanie Norrell, David Reinhardt, Betsy Stockdale, and Christine Wollmuth. Their work made ours easier. The editorial input from Candase Hokanson is greatly appreciated.

Thanks go to many people at Microsoft Press, including acquisitions editor Devon Musgrave, project editor Carol Dillingham, project editor Christian Holdener of S4Carlisle Publishing Services, copy editor Kathy Krause, proofreader Nicole Schlutt, indexer Maureen Johnson, compositor Sambasivam Sangaran, and production artists Balaganesan M., Srinivasan R., and Ganeshbabu G. Karl especially values his long-term relationship, and friendship, with Devon Musgrave and Ben Ryan.

The comments and questions from thousands of students in our requirements training classes over the years have been most helpful in stimulating our thinking about
requirements issues. Our consulting experiences and the thought-provoking questions we receive from readers have kept us in touch with what practitioners struggle with on a daily basis and helped us think through some of these difficult topics. Please share your own experiences with us at karl@processimpact.com or joy.beatty@seilevel.com.

As always, Karl would like to thank his wife, Chris Zambito. And as always, she was patient and good-humored throughout the process. Karl also thanks Joy for prompting him into working on this project and for her terrific contributions. Working with her was a lot of fun, and she added a great deal of value to the book. It was great to have someone to bounce ideas off, to help make difficult decisions, and to chew hard on draft chapters before we inflicted them on the reviewers.

Joy is particularly grateful to her husband, Tony Hamilton, for supporting her writing dreams so soon again; to her daughter, Skye, for making it easy to keep her daily priorities balanced; and to Sean and Estelle for being the center of her family fun times. Joy wants to extend a special thanks to all of the Seilevel employees who collaborate to push the software requirements field forward. She particularly wants to thank two colleagues and friends: Anthony Chen, whose support for her writing this book was paramount; and Rob Sparks, for his continued encouragement in such endeavors. Finally, Joy owes a great deal of gratitude to Karl for allowing her to join him in this co-authorship, teaching her something new every day, and being an absolute joy to work with!
Jeremy walked into the office of Ruth Gilbert, the director of the Drug Discovery Division at Contoso Pharmaceuticals. Ruth had asked the information technology team that supported Contoso’s research organization to build a new application to help the research chemists accelerate their exploration for new drugs. Jeremy was assigned as the business analyst for the project. After introducing himself and discussing the project in broad terms, Jeremy said to Ruth, “I’d like to talk with some of your chemists to understand their requirements for the system. Who might be some good people to start with?”

Ruth replied, “I did that same job for five years before I became the division director three years ago. You don’t really need to talk to any of my people; I can tell you everything you need to know about this project.”

Jeremy was concerned. Scientific knowledge and technologies change quickly, so he wasn’t sure if Ruth could adequately represent the current and future needs for users of this complex application. Perhaps there were some internal politics going on that weren’t apparent and there was a good reason for Ruth to create a buffer between Jeremy and the actual users. After some discussion, though, it became clear that Ruth didn’t want any of her people involved directly with the project.

“Okay,” Jeremy agreed reluctantly. “Maybe I can start by doing some document analysis and bring questions I have to you. Can we set up a series of interviews for the next couple of weeks so I can understand the kinds of things you expect your scientists to be able to do with this new system?”

“Sorry, I’m swamped right now,” Ruth told him. “I can give you a couple of hours in about three weeks to clarify things you’re unsure about. Just go ahead and start writing the requirements. When we meet, then you can ask me any questions you still have. I hope that will let you get the ball rolling on this project.”

If you share our conviction that customer involvement is a critical factor in delivering excellent software, you will ensure that the business analyst (BA) and project manager for your project will work hard to engage appropriate customer representatives from the outset. Success in software requirements, and hence in software development, depends on getting the voice of the user close to the ear of the developer. To find the voice of the user, take the following steps:

- Identify the different classes of users for your product.
- Select and work with individuals who represent each user class and other stakeholder groups.
- Agree on who the requirements decision makers are for your project.
Customer involvement is the best way to avoid the expectation gap described in Chapter 2, “Requirements from the customer’s perspective,” a mismatch between the product that customers expect to receive and what developers build. It’s not enough simply to ask a few customers or their manager what they want once or twice and then start coding. If developers build exactly what customers initially request, they’ll probably have to build it again because customers often don’t know what they really need. In addition, the BAs might not be talking to the right people or asking the right questions.

The features that users present as their “wants” don’t necessarily equate to the functionality they need to perform their tasks with the new product. To gain a more accurate view of user needs, the business analyst must collect a wide range of user input, analyze and clarify it, and specify just what needs to be built to let users do their jobs. The BA has the lead responsibility for recording the new system’s necessary capabilities and properties and for communicating that information to other stakeholders. This is an iterative process that takes time. If you don’t invest the time to achieve this shared understanding—this common vision of the intended product—the certain outcomes are rework, missed deadlines, cost overruns, and customer dissatisfaction.

User classes

People often talk about “the user” for a software system as though all users belong to a monolithic group with similar characteristics and needs. In reality, most products of any size appeal to a diversity of users with different expectations and goals. Rather than thinking of “the user” in singular, spend some time identifying the multiple user classes and their roles and privileges for your product.

Classifying users

Chapter 2 described many of the types of stakeholders that a project might have. As shown in Figure 6-1, a user class is a subset of the product’s users, which is a subset of the product’s customers, which is a subset of its stakeholders. An individual can belong to multiple user classes. For example, an application's administrator might also interact with it as an ordinary user at times. A product’s users might differ—among other ways—in the following respects, and you can group users into a number of distinct user classes based on these sorts of differences:

- Their access privilege or security levels (such as ordinary user, guest user, administrator)
- The tasks they perform during their business operations
- The features they use
- The frequency with which they use the product
- Their application domain experience and computer systems expertise
- The platforms they will be using (desktop PCs, laptop PCs, tablets, smartphones, specialized devices)
Their native language

Whether they will interact with the system directly or indirectly

It’s tempting to group users into classes based on their geographical location or the kind of company they work in. One company that creates software used in the banking industry initially considered distinguishing users based on whether they worked in a large commercial bank, a small commercial bank, a savings and loan institution, or a credit union. These distinctions really represent different market segments, though, not different user classes.

A better way to identify user classes is to think about the tasks that various users will perform with the system. All of those types of financial institutions will have tellers, employees who process loan applications, business bankers, and so forth. The individuals who perform such activities—whether they are job titles or simply roles—will have similar functional needs for the system across all of the financial institutions. Tellers all have to do more or less the same things, business bankers do more or less the same things, and so on. More logical user class names for a banking system therefore might include teller, loan officer, business banker, and branch manager. You might discover additional user classes by thinking of possible use cases, user stories, and process flows and who might perform them.

Certain user classes could be more important than others for a specific project. Favored user classes are those whose satisfaction is most closely aligned with achieving the project’s business objectives. When resolving conflicts between requirements from different user classes or making priority decisions, favored user classes receive preferential treatment. This doesn’t mean that the customers who are paying for the system (who might not be users at all) or those who have the most political clout should necessarily be favored. It’s a matter of alignment with the business objectives.

Disfavored user classes are groups who aren’t supposed to use the product for legal, security, or safety reasons (Gause and Lawrence 1999). You might build in features to deliberately make it hard for disfavored users to do things they aren’t supposed to do. Examples include access security
mechanisms, user privilege levels, antimalware features (for non-human users), and usage logging. Locking a user’s account after four unsuccessful login attempts protects against access by the disfavored user class of “user impersonators,” albeit at the risk of inconveniencing forgetful legitimate users. If my bank doesn’t recognize the computer I’m using, it sends me an email message with a one-time access code I have to enter before I can log on. This feature was implemented because of the disfavored user class of “people who might have stolen my banking information.”

You might elect to ignore still other user classes. Yes, they will use the product, but you don’t specifically build it to suit them. If there are any other groups of users that are neither favored, disfavored, nor ignored, they are of equal importance in defining the product’s requirements.

Each user class will have its own set of requirements for the tasks that members of the class must perform. There could be some overlap between the needs of different user classes. Tellers, business bankers, and loan officers all might have to check a bank customer’s account balance, for instance. Different user classes also could have different quality expectations, such as usability, that will drive user interface design choices. New or occasional users are concerned with how easy the system is to learn. Such users like menus, graphical user interfaces, uncluttered screen displays, wizards, and help screens. As users gain experience with the system, they become more interested in efficiency. They now value keyboard shortcuts, customization options, toolbars, and scripting facilities.

**Trap** Don’t overlook indirect user classes. They won’t use your application themselves, instead accessing its data or services through other applications or through reports. Your customer once removed is still your customer.

User classes need not be human beings. They could be software agents performing a service on behalf of a human user, such as bots. Software agents can scan networks for information about goods and services, assemble custom news feeds, process your incoming email, monitor physical systems and networks for problems or intrusions, or perform data mining. Internet agents that probe websites for vulnerabilities or to generate spam are a type of disfavored non-human user class. If you identify these sorts of disfavored user classes, you might specify certain requirements not to meet their needs but rather to thwart them. For instance, website tools such as CAPTCHA that validate whether a user is a human being attempt to block such disruptive access by “users” you want to keep out.

Remember, users are a subset of customers, which are a subset of stakeholders. You’ll need to consider a much broader range of potential sources of requirements than just direct and indirect user classes. For instance, even though the development team members aren’t end users of the system they’re building, you need their input on internal quality attributes such as efficiency, modifiability, portability, and reusability, as described in Chapter 14, “Beyond functionality.” One company found that every installation of their product was an expensive nightmare until they introduced an “installer” user class so they could focus on requirements such as the development of a customization architecture for their product. Look well beyond the obvious end users when you’re trying to identify stakeholders whose requirements input is necessary.
Identifying your user classes

Identify and characterize the different user classes for your product early in the project so you can elicit requirements from representatives of each important class. A useful technique for this is a collaboration pattern developed by Ellen Gottesdiener called “expand then contract” (Gottesdiener 2002). Start by asking the project sponsor who he expects to use the system. Then brainstorm as many user classes as you can think of. Don’t get nervous if there are dozens at this stage; you’ll condense and categorize them later. It’s important not to overlook a user class, which can lead to problems later when someone complains that the delivered solution doesn’t meet her needs. Next, look for groups with similar needs that you can either combine or treat as a major user class with several subclasses. Try to pare the list down to about 15 or fewer distinct user classes.

One company that developed a specialized product for about 65 corporate customers initially regarded each company as a distinct user with unique needs. Grouping their customers into just six user classes greatly simplified their requirements challenges. Donald Gause and Gerald Weinberg (1989) offer much advice about casting a wide net to identify potential users, pruning the user list, and seeking specific users to participate in the project.

Various analysis models can help you identify user classes. The external entities shown outside your system on a context diagram (see Chapter 5, “Establishing the business requirements”) are candidates for user classes. A corporate organization chart can also help you discover potential users and other stakeholders (Beatty and Chen 2012). Figure 6-2 illustrates a portion of the organization chart for Contoso Pharmaceuticals. Nearly all of the potential users for the system are likely to be found somewhere in this chart. While performing stakeholder and user analysis, study the organization chart to look for:

- Departments that participate in the business process.
- Departments that are affected by the business process.
- Departments or role names in which either direct or indirect users might be found.
- User classes that span multiple departments.
- Departments that might have an interface to external stakeholders outside the company.

Organization chart analysis reduces the likelihood that you will overlook an important class of users within that organization. It shows you where to seek potential representatives for specific user classes, as well as helping determine who the key requirements decision makers might be. You might find multiple user classes with diverse needs within a single department. Conversely, recognizing the same user class in multiple departments can simplify requirements elicitation. Studying the organization chart helps you judge how many user representatives you’ll need to work with to feel confident that you thoroughly understand the broad user community’s needs. Also try to understand what type of information the users from each department might supply based on their role in the organization and their department’s perspective on the project.
Document the user classes and their characteristics, responsibilities, and physical locations in the software requirements specification (SRS) or in a requirements plan for your project. Check that information against any information you might already have about stakeholder profiles in the vision and scope document to avoid conflicts and duplication. Include all pertinent information you have about each user class, such as its relative or absolute size and which classes are favored. This will help the team prioritize change requests and conduct impact assessments later on. Estimates of the volume and type of system transactions help the testers develop a usage profile for the system so that they can plan their verification activities. The project manager and business analyst of the Chemical Tracking System discussed in earlier chapters identified the user classes and characteristics shown in Table 6-1.

**TABLE 6-1 User classes for the Chemical Tracking System**

<table>
<thead>
<tr>
<th>Name</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chemists (favored)</td>
<td>Approximately 1,000 located in 6 buildings</td>
<td>Chemists will request chemicals from vendors and from the chemical stockroom. Each chemist will use the system several times per day, mainly for requesting chemicals and tracking chemical containers into and out of the laboratory. The chemists need to search vendor catalogs for specific chemical structures imported from the tools they use for drawing structures.</td>
</tr>
<tr>
<td>Buyers</td>
<td>5</td>
<td>Buyers in the purchasing department process chemical requests. They place and track orders with external vendors. They know little about chemistry and need simple query facilities to search vendor catalogs. Buyers will not use the system’s container-tracking features. Each buyer will use the system an average of 25 times per day.</td>
</tr>
<tr>
<td>Chemical stockroom staff</td>
<td>6 technicians, 1 supervisor</td>
<td>The chemical stockroom staff manages an inventory of more than 500,000 chemical containers. They will supply containers from three stockrooms, request new chemicals from vendors, and track the movement of all containers into and out of the stockrooms. They are the only users of the inventory-reporting feature. Because of their high transaction volume, features that are used only by the chemical stockroom staff must be automated and efficient.</td>
</tr>
<tr>
<td>Health and Safety Department staff (favored)</td>
<td>1 manager</td>
<td>The Health and Safety Department staff will use the system only to generate predefined quarterly reports that comply with federal and state chemical usage and disposal reporting regulations. The Health and Safety Department manager will request changes in the reports periodically as government regulations change. These report changes are of the highest priority, and implementation will be time critical.</td>
</tr>
</tbody>
</table>
Consider building a catalog of user classes that recur across multiple applications. Defining user classes at the enterprise level lets you reuse those user class descriptions in future projects. The next system you build might serve the needs of some new user classes, but it probably will also be used by user classes from your earlier systems. If you do include the user-class descriptions in the project’s SRS, you can incorporate entries from the reusable user-class catalog by reference and just write descriptions of any new groups that are specific to that application.

User personas

To help bring your user classes to life, consider creating a persona for each one, a description of a representative member of the user class (Cooper 2004; Leffingwell 2011). A persona is a description of a hypothetical, generic person who serves as a stand-in for a group of users having similar characteristics and needs. You can use personas to help you understand the requirements and to design the user experience to best meet the needs of specific user communities.

A persona can serve as a placeholder when the BA doesn’t have an actual user representative at hand. Rather than having progress come to a halt, the BA can envision a persona performing a particular task or try to assess what the persona’s preferences would be, thereby drafting a requirements starting point to be confirmed when an actual user is available. Persona details for a commercial customer include social and demographic characteristics and behaviors, preferences, annoyances, and similar information. Make sure the personas you create truly are representative of their user class, based on market, demographic, and ethnographic research.

Here’s an example of a persona for one user class on the Chemical Tracking System:

Fred, 41, has been a chemist at Contoso Pharmaceuticals since he received his Ph.D. 14 years ago. He doesn’t have much patience with computers. Fred usually works on two projects at a time in related chemical areas. His lab contains approximately 300 bottles of chemicals and gas cylinders. On an average day, he’ll need four new chemicals from the stockroom. Two of these will be commercial chemicals in stock, one will need to be ordered, and one will come from the supply of proprietary Contoso chemical samples. On occasion, Fred will need a hazardous chemical that requires special training for safe handling. When he buys a chemical for the first time, Fred wants the material safety data sheet emailed to him automatically. Each year, Fred will synthesize about 20 new proprietary chemicals to go into the stockroom. Fred wants a report of his chemical usage for the previous month to be generated automatically and sent to him by email so that he can monitor his chemical exposure.

As the business analyst explores the chemists’ requirements, he can think about Fred as the archetype of this user class and ask himself, “What would Fred need to do?” Working with a persona makes the requirements thought process more tangible than if you simply contemplate what a whole faceless group of people might want. Some people choose a random human face of the appropriate gender to make a persona seem even more real.
Dean Leffingwell (2011) suggests that you design the system to make it easy for the individual described in your persona to use the application. That is, you focus on meeting that one (imaginary) person’s needs. Provided you’ve created a persona that accurately represents the user class, this should help you do a good job of satisfying the needs and expectations of the whole class. As one colleague related, “On a project for servicing coin-operated vending machines, I introduced Dolly the Serviceperson and Ralph the Warehouse Supervisor. We wrote scenarios for them and they became part of the project team—virtually.”

### Connecting with user representatives

Every kind of project—corporate information systems, commercial applications, embedded systems, websites, contracted software—needs suitable representatives to provide the voice of the user. These users should be involved throughout the development life cycle, not just in an isolated requirements phase at the beginning of the project. Each user class needs someone to speak for it.

It’s easiest to gain access to actual users when you’re developing applications for deployment within your own company. If you’re developing commercial software, you might engage people from your beta-testing or early-release sites to provide requirements input much earlier in the development process. (See the “External product champions” section later in this chapter). Consider setting up focus groups of current users of your products or your competitors’ products. Instead of just guessing at what your users might want, ask some of them.

One company asked a focus group to perform certain tasks with various digital cameras and computers. The results indicated that the company’s camera software took too long to perform the most common operation because of a design decision that was made to accommodate less likely scenarios as well. The company changed their next camera to reduce customer complaints about speed.

Be sure that the focus group represents the kinds of users whose needs should drive your product development. Include both expert and less experienced customers. If your focus group represents only early adopters or blue-sky thinkers, you might end up with many sophisticated and technically difficult requirements that few customers find useful.

Figure 6-3 illustrates some typical communication pathways that connect the voice of the user to the ear of the developer. One study indicated that employing more kinds of communication links and more direct links between developers and users led to more successful projects (Keil and Carmel 1995). The most direct communication occurs when developers can talk to appropriate users themselves, which means that the developer is also performing the business analyst role. This can work on very small projects, provided the developer involved has the appropriate BA skills, but it doesn’t scale up to large projects with thousands of potential users and dozens of developers.
As in the children’s game “Telephone,” intervening layers between the user and the developer increase the chance of miscommunication and delay transmission. Some of these intervening layers add value, though, as when a skilled BA works with users or other participants to collect, evaluate, refine, and organize their input. Recognize the risks that you assume by using marketing staff, product managers, subject matter experts, or others as surrogates for the actual voice of the user. Despite the obstacles to—and the cost of—optimizing user representation, your product and your customers will suffer if you don’t talk to the people who can provide the best information.

The product champion

Many years ago I worked in a small software development group that supported the scientific research activities at a major corporation. Each of our projects included a few key members of our user community to provide the requirements. We called these people *product champions* (Wiegers 1996). The product champion approach provides an effective way to structure that all-important customer-development collaborative partnership discussed in Chapter 2.
Each product champion serves as the primary interface between members of a single user class and the project’s business analyst. Ideally, the champions will be actual users, not surrogates such as funding sponsors, marketing staff, user managers, or software developers imagining themselves to be users. Product champions gather requirements from other members of the user classes they represent and reconcile inconsistencies. Requirements development is thus a shared responsibility of the BA and selected users, although the BA should actually write the requirements documents. It’s hard enough to write good requirements if you do it for a living; it is not realistic to expect users who have never written requirements before to do a good job.

The best product champions have a clear vision of the new system. They’re enthusiastic because they see how it will benefit them and their peers. Champions should be effective communicators who are respected by their colleagues. They need a thorough understanding of the application domain and the solution’s operating environment. Great product champions are in demand for other assignments, so you’ll have to build a persuasive case for why particular individuals are critical to project success. For example, product champions can lead adoption of the application by the user community, which might be a success metric that managers will appreciate. We have found that good product champions made a huge difference in our projects, so we offer them public reward and recognition for their contributions.

Our software development teams enjoyed an additional benefit from the product champion approach. On several projects, we had excellent champions who spoke out on our behalf with their colleagues when the customers wondered why the software wasn’t done yet. “Don’t worry about it,” the champions told their peers and their managers. “I understand and agree with the software team’s approach to software engineering. The time we’re spending on requirements will help us get the system we really need and will save time in the long run.” Such collaboration helps break down the tension that can arise between customers and development teams.

The product champion approach works best if each champion is fully empowered to make binding decisions on behalf of the user class he represents. If a champion’s decisions are routinely overruled by others, his time and goodwill are being wasted. However, the champions must remember that they are not the sole customers. Problems arise when the individual filling this critical liaison role doesn’t adequately communicate with his peers and presents only his own wishes and ideas.

**External product champions**

When developing commercial software, it can be difficult to find product champions from outside your company. Companies that develop commercial products sometimes rely on internal subject matter experts or outside consultants to serve as surrogates for actual users, who might be unknown or difficult to engage. If you have a close working relationship with some major corporate customers, they might welcome the opportunity to participate in requirements elicitation. You might give external product champions economic incentives for their participation. Consider offering them discounts on the product or paying for the time they spend working with you on requirements. You still face the challenge of how to avoid hearing only the champions’ requirements and overlooking the needs of other stakeholders. If you have a diverse customer base, first identify core requirements that are common to all customers. Then define additional requirements that are specific to individual corporate customers, market segments, or user classes.
Another alternative is to hire a suitable product champion who has the right background. One company that developed a retail point-of-sale and back-office system for a particular industry hired three store managers to serve as full-time product champions. As another example, my longtime family doctor, Art, left his medical practice to become the voice-of-the-physician at a medical software company. Art’s new employer believed that it was worth the expense to hire a doctor to help the company build software that other doctors would accept. A third company hired several former employees from one of their major customers. These people provided valuable domain expertise as well as insight into the politics of the customer organization. To illustrate an alternative engagement model, one company had several corporate customers that used their invoicing systems extensively. Rather than bringing in product champions from the customers, the developing company sent BAs to the customer sites. Customers willingly dedicated some of their staff time to helping the BAs get the right requirements for the new invoicing system.

Anytime the product champion is a former or simulated user, watch out for disconnects between the champion’s perceptions and the current needs of real users. Some domains change rapidly, whereas others are more stable. Regardless, if people aren’t operating in the role anymore, they simply might have forgotten the intricacies of the daily job. The essential question is whether the product champion, no matter what her background or current job, can accurately represent the needs of today’s real users.

**Product champion expectations**

To help the product champions succeed, document what you expect your champions to do. These written expectations can help you build a case for specific individuals to fill this critical role. Table 6-2 identifies some activities that product champions might perform (Wiegers 1996). Not every champion will do all of these; use this table as a starting point to negotiate each champion’s responsibilities.

**TABLE 6-2 Possible product champion activities**

<table>
<thead>
<tr>
<th>Category</th>
<th>Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning</td>
<td>■ Refine the scope and limitations of the product.</td>
</tr>
<tr>
<td></td>
<td>■ Identify other systems with which to interact.</td>
</tr>
<tr>
<td></td>
<td>■ Evaluate the impact of the new system on business operations.</td>
</tr>
<tr>
<td></td>
<td>■ Define a transition path from current applications or manual operations.</td>
</tr>
<tr>
<td></td>
<td>■ Identify relevant standards and certification requirements.</td>
</tr>
<tr>
<td>Requirements</td>
<td>■ Collect input on requirements from other users.</td>
</tr>
<tr>
<td></td>
<td>■ Develop usage scenarios, use cases, and user stories.</td>
</tr>
<tr>
<td></td>
<td>■ Resolve conflicts between proposed requirements within the user class.</td>
</tr>
<tr>
<td></td>
<td>■ Define implementation priorities.</td>
</tr>
<tr>
<td></td>
<td>■ Provide input regarding performance and other quality requirements.</td>
</tr>
<tr>
<td></td>
<td>■ Evaluate prototypes.</td>
</tr>
<tr>
<td></td>
<td>■ Work with other decision makers to resolve conflicts among requirements from different stakeholders.</td>
</tr>
<tr>
<td></td>
<td>■ Provide specialized algorithms.</td>
</tr>
<tr>
<td>Category</td>
<td>Activities</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Validation and verification</td>
<td>■ Review requirements specifications.</td>
</tr>
<tr>
<td></td>
<td>■ Define acceptance criteria.</td>
</tr>
<tr>
<td></td>
<td>■ Develop user acceptance tests from usage scenarios.</td>
</tr>
<tr>
<td></td>
<td>■ Provide test data sets from the business.</td>
</tr>
<tr>
<td></td>
<td>■ Perform beta testing or user acceptance testing.</td>
</tr>
<tr>
<td>User aids</td>
<td>■ Write portions of user documentation and help text.</td>
</tr>
<tr>
<td></td>
<td>■ Contribute to training materials or tutorials.</td>
</tr>
<tr>
<td></td>
<td>■ Demonstrate the system to peers.</td>
</tr>
<tr>
<td>Change management</td>
<td>■ Evaluate and prioritize defect corrections and enhancement requests.</td>
</tr>
<tr>
<td></td>
<td>■ Dynamically adjust the scope of future releases or iterations.</td>
</tr>
<tr>
<td></td>
<td>■ Evaluate the impact of proposed changes on users and business processes.</td>
</tr>
<tr>
<td></td>
<td>■ Participate in making change decisions.</td>
</tr>
</tbody>
</table>

**Multiple product champions**

One person can rarely describe the needs for all users of an application. The Chemical Tracking System had four major user classes, so it needed four product champions selected from the internal user community at Contoso Pharmaceuticals. Figure 6-4 illustrates how the project manager set up a team of BAs and product champions to elicit the right requirements from the right sources. These champions were not assigned full time, but each one spent several hours per week working on the project. Three BAs worked with the four product champions to elicit, analyze, and document their requirements. (One BA worked with two product champions because the Buyer and the Health and Safety Department user classes were small and had few requirements.) One of the BAs assembled all the input into a unified SRS.

![Product champion model](image)

**FIGURE 6-4** Product champion model for the Chemical Tracking System.

We didn’t expect a single person to provide all the diverse requirements for the hundreds of chemists at Contoso. Don, the product champion for the Chemist user class, assembled a backup
team of five chemists from other parts of the company. They represented subclasses within the broad Chemist user class. This hierarchical approach engaged additional users in requirements development while avoiding the expense of massive workshops or dozens of individual interviews. Don always strove for consensus. However, he willingly made the necessary decisions when agreement wasn’t achieved so the project could move ahead. No backup team was necessary when the user class was small enough or cohesive enough that one individual truly could represent the group’s needs.¹

The voiceless user class

A business analyst at Humongous Insurance was delighted that an influential user, Rebecca, agreed to serve as product champion for the new claims processing system. Rebecca had many ideas about the system features and user interface design. Thrilled to have the guidance of an expert, the development team happily complied with her requests. After delivery, though, they were shocked to receive many complaints about how hard the system was to use.

Rebecca was a power user. She specified usability requirements that were great for experts, but the 90 percent of users who weren’t experts found the system unintuitive and difficult to learn. The BA didn’t recognize that the claims processing system had at least two user classes. The large group of non–power users was disenfranchised in the requirements and user interface design processes. Humongous paid the price in an expensive redesign. The BA should have engaged at least one more product champion to represent the large class of nonexpert users.

Selling the product champion idea

Expect to encounter resistance when you propose the idea of having product champions on your projects. “The users are too busy.” “Management wants to make the decisions.” “They’ll slow us down.” “We can’t afford it.” “They’ll run amok and scope will explode.” “I don’t know what I’m supposed to do as a product champion.” Some users won’t want to cooperate on a project that will make them change how they work or might even threaten their jobs. Managers are sometimes reluctant to delegate authority for requirements to ordinary users.

Separating business requirements from user requirements alleviates some of these discomforts. As an actual user, the product champion makes decisions at the user requirements level within the scope boundaries imposed by the business requirements. The management sponsor retains the authority to make decisions that affect the product vision, project scope, business-related priorities, schedule, or budget. Documenting and negotiating each product champion’s role and responsibilities give candidate champions a comfort level about what they’re being asked to do. Remind management that a product champion is a key contributor who can help the project achieve its business objectives.

¹ There’s an interesting coda to this story. Years after I worked on this project, a man in a class I was teaching said he had worked at the company that Contoso Pharmaceuticals had contracted to build the Chemical Tracking System. The developers found that the requirements specification we created using this product champion model provided a solid foundation for the development work. The system was delivered successfully and was used at Contoso for many years.
If you encounter resistance, point out that insufficient user involvement is a leading cause of software project failure. Remind the protesters of problems they’ve experienced on previous projects that trace back to inadequate user input. Every organization has horror stories of new systems that didn’t satisfy user needs or failed to meet unstated usability or performance expectations. You can’t afford to rebuild or discard systems that don’t measure up because no one understood the requirements. Product champions provide one way to get that all-important customer input in a timely way, not at the end of the project when customers are disappointed and developers are tired.

**Product champion traps to avoid**

The product champion model has succeeded in many environments. It works only when the product champions understand and sign up for their responsibilities, have the authority to make decisions at the user requirements level, and have time available to do the job. Watch out for the following potential problems:

- Managers override the decisions that a qualified and duly authorized product champion makes. Perhaps a manager has a wild new idea at the last minute, or thinks he knows what the users need. This behavior often results in dissatisfied users and frustrated product champions who feel that management doesn’t trust them.

- A product champion who forgets that he is representing other customers and presents only his own requirements won’t do a good job. He might be happy with the outcome, but others likely won’t be.

- A product champion who lacks a clear vision of the new system might defer decisions to the BA. If all of the BA’s ideas are fine with the champion, the champion isn’t providing much help.

- A senior user might nominate a less experienced user as champion because she doesn’t have time to do the job herself. This can lead to backseat driving from the senior user who still wishes to strongly influence the project’s direction.

Beware of users who purport to speak for a user class to which they do not belong. Rarely, an individual might actively try to block the BA from working with the ideal contacts for some reason. On the Chemical Tracking System, the product champion for the chemical stockroom staff—herself a former chemist—initially insisted on providing what she thought were the needs of the chemist user class. Unfortunately, her input about current chemist needs wasn’t accurate. It was difficult to convince her that this wasn’t her job, but the BA didn’t let her intimidate him. The project manager lined up a separate product champion for the chemists, who did a great job of collecting, evaluating, and relaying that community’s requirements.
User representation on agile projects

Frequent conversations between project team members and appropriate customers are the most effective way to resolve many requirements issues and to flesh out requirements specifics when they are needed. Written documentation, however detailed, is an incomplete substitute for these ongoing communications. A fundamental tenet of Extreme Programming, one of the early agile development methods, is the presence of a full-time, on-site customer for these discussions (Jeffries, Anderson, and Hendrickson, 2001).

Some agile development methods include a single representative of stakeholders called a product owner in the team to serve as the voice of the customer (Schwaber 2004; Cohn 2010; Leffingwell 2011). The product owner defines the product’s vision and is responsible for developing and prioritizing the contents of the product backlog. (The backlog is the prioritized list of user stories—requirements—for the product and their allocation to upcoming iterations, called sprints in the agile development method called Scrum.) The product owner therefore spans all three levels of requirements: business, user, and functional. He essentially straddles the product champion and business analyst functions, representing the customer, defining product features, prioritizing them, and so forth. Ultimately, someone does have to make decisions about exactly what capabilities to deliver in the product and when. In Scrum, that’s the product owner’s responsibility.

The ideal state of having a single product owner isn’t always practical. We know of one company that was implementing a package solution to run their insurance business. The organization was too big and complex to have one person who understood everything in enough detail to make all decisions about the implementation. Instead, the customers selected a product owner from each department to own the priorities for the functionality used by that department. The company’s CIO served as the lead product owner. The CIO understood the entire product vision, so he could ensure that the departments were on track to deliver that vision. He had responsibility for decision making when there were conflicts between department-level product owners.

The premises of the on-site customer and close customer collaboration with developers that agile methods espouse certainly are sound. In fact, we feel strongly that all development projects warrant this emphasis on user involvement. As you have seen, though, all but the smallest projects have multiple user classes, as well as numerous additional stakeholders whose interests must be represented. In many cases it’s not realistic to expect a single individual to be able to adequately understand and describe the needs of all relevant user classes, nor to make all the decisions associated with product definition. Particularly with internal corporate projects, it will generally work better to use a representative structure like the product champion model to ensure adequate user engagement.
The product owner and product champion schemes are not mutually exclusive. If the product owner is functioning in the role of a business analyst, rather than as a stakeholder representative himself, he could set up a structure with one or more product champions to see that the most appropriate sources provide input. Alternatively, the product owner could collaborate with one or more business analysts, who then work with stakeholders to understand their requirements. The product owner would then serve as the ultimate decision maker.

“On-site” customer

I once wrote programs for a research scientist who sat about 10 feet from my desk. John could provide instantaneous answers to my questions, provide feedback on user interface designs, and clarify our informally written requirements. One day John moved to a new office, around the corner on the same floor of the same building, about 100 feet away. I perceived an immediate drop in my programming productivity because of the cycle time delay in getting John’s input. I spent more time fixing problems because sometimes I went down the wrong path before I could get a course correction. There’s no substitute for having the right customers continuously available to the developers both on-site and “on-sight.” Beware, though, of too-frequent interruptions that make it hard for people to refocus their attention on their work. It can take up to 15 minutes to reimmerse yourself into the highly productive, focused state of mind called flow (DeMarco and Lister 1999).

An on-site customer doesn’t guarantee the desired outcome. My colleague Chris, a project manager, established a development team environment with minimal physical barriers and engaged two product champions. Chris offered this report: “While the close proximity seems to work for the development team, the results with product champions have been mixed. One sat in our midst and still managed to avoid us all. The new champion does a fine job of interacting with the developers and has truly enabled the rapid development of software.” There is no substitute for having the right people, in the right role, in the right place, with the right attitude.

Resolving conflicting requirements

Someone must resolve conflicting requirements from different user classes, reconcile inconsistencies, and arbitrate questions of scope that arise. The product champions or product owner can handle this in many, but likely not all, cases. Early in the project, determine who the decision makers will be for requirements issues, as discussed in Chapter 2. If it’s not clear who is responsible for making these decisions or if the authorized individuals abdicate their responsibilities, the decisions will fall to the developers or analysts by default. Most of them don’t have the necessary knowledge and perspective.
to make the best business decisions, though. Analysts sometimes defer to the loudest voice they hear or to the person highest on the food chain. Though understandable, this is not the best strategy. Decisions should be made as low in the organization’s hierarchy as possible by well-informed people who are close to the issues.

Table 6-3 identifies some requirements conflicts that can arise on projects and suggests ways to handle them. The project’s leaders need to determine who will decide what to do when such situations arise, who will make the call if agreement is not reached, and to whom significant issues must be escalated when necessary.

**TABLE 6-3** Suggestions for resolving requirements disputes

<table>
<thead>
<tr>
<th>Disagreement between</th>
<th>How to resolve</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual users</td>
<td>Product champion or product owner decides</td>
</tr>
<tr>
<td>User classes</td>
<td>Favored user class gets preference</td>
</tr>
<tr>
<td>Market segments</td>
<td>Segment with greatest impact on business success gets preference</td>
</tr>
<tr>
<td>Corporate customers</td>
<td>Business objectives dictate direction</td>
</tr>
<tr>
<td>Users and user managers</td>
<td>Product owner or product champion for the user class decides</td>
</tr>
<tr>
<td>Development and customers</td>
<td>Customers get preference, but in alignment with business objectives</td>
</tr>
<tr>
<td>Development and marketing</td>
<td>Marketing gets preference</td>
</tr>
</tbody>
</table>

**Trap**  Don’t justify doing whatever any customer demands because “The customer is always right.” We all know the customer is *not* always right (Wiegers 2011). Sometimes, a customer is unreasonable, uninformed, or in a bad mood. The customer always has a point, though, and the software team must understand and respect that point.

These negotiations don’t always turn out the way the analyst might hope. Certain customers might reject all attempts to consider reasonable alternatives and other points of view. We’ve seen cases where marketing never said no to a customer request, no matter how infeasible or expensive. The team needs to decide who will be making decisions on the project’s requirements before they confront these types of issues. Otherwise, indecision and the revisiting of previous decisions can stall the project in endless wrangling. If you’re a BA caught in this dilemma, rely on your organizational structure and processes to work through the disagreements. But, as we’ve cautioned before, there aren’t any easy solutions if you’re working with truly unreasonable people.
Next steps

- Relate Figure 6-3 to the way you hear the voice of the user in your own environment. Do you encounter any problems with your current communication links? Identify the shortest and most effective communication paths that you can use to elicit user requirements in the future.

- Identify the different user classes for your project. Which ones are favored? Which, if any, are disfavored? Who would make a good product champion for each important user class? Even if the project is already underway, the team likely would benefit from having product champions involved.

- Starting with Table 6-2, define the activities you would like your product champions to perform. Negotiate the specific contributions with each candidate product champion and his or her manager.

- Determine who the decision makers are for requirements issues on your project. How well does your current decision-making approach work? Where does it break down? Are the right people making decisions? If not, who should be doing it? Suggest processes that the decision makers should use for reaching agreement on requirements issues.
CHAPTER 21

Enhancement and replacement projects

Most of this book describes requirements development as though you are beginning a new software or system development project, sometimes called a *green-field project*. However, many organizations devote much of their effort to enhancing or replacing existing information systems or building new releases of established commercial products. Most of the practices described in this book are appropriate for enhancement and replacement projects. This chapter provides specific suggestions as to which practices are most relevant and how to use them.

An *enhancement project* is one in which new capabilities are added to an existing system. Enhancement projects might also involve correcting defects, adding new reports, and modifying functionality to comply with revised business rules or needs.

A *replacement* (or *reengineering*) project replaces an existing application with a new custom-built system, a commercial off-the-shelf (COTS) system, or a hybrid of those. Replacement projects are most commonly implemented to improve performance, cut costs (such as maintenance costs or license fees), take advantage of modern technologies, or meet regulatory requirements. If your replacement project will involve a COTS solution, the guidance presented in Chapter 22, “Packaged solution projects,” will also be helpful.

Replacement and enhancement projects face some particular requirements issues. The original developers who held all the critical information in their heads might be long gone. It’s tempting to claim that a small enhancement doesn’t warrant writing any requirements. Developers might believe that they don’t need detailed requirements if they are replacing an existing system’s functionality. The approaches described in this chapter can help you to deal with the challenges of enhancing or replacing an existing system to improve its ability to meet the organization’s current business needs.

**The case of the missing spec**

The requirements specification for the next release of a mature system often says, essentially, “The new system should do everything the old system does, except add these new features and fix those bugs.” A business analyst once received just such a specification for version 5 of a major product. To find out exactly what the current release did, she looked at the SRS for version 4. Unfortunately, it also said, in essence, “Version 4 should do everything that version 3 does, except add these new features and fix those bugs.” She followed the trail back, but every
SRS described just the differences that the new version should exhibit compared to the previous version. Nowhere was there a description of the original system. Consequently, everyone had a different understanding of the current system’s capabilities. If you’re in this situation, document the requirements for your project more thoroughly so that all the stakeholders—both present and future—understand what the system does.

Expected challenges

The presence of an existing system leads to common challenges that both enhancement and replacement projects will face, including the following:

- The changes made could degrade the performance to which users are accustomed.
- Little or no requirements documentation might be available for the existing system.
- Users who are familiar with how the system works today might not like the changes they are about to encounter.
- You might unknowingly break or omit functionality that is vital to some stakeholder group.
- Stakeholders might take this opportunity to request new functionality that seems like a good idea but isn’t really needed to meet the business objectives.

Even if there is existing documentation, it might not prove useful. For enhancement projects, the documentation might not be up to date. If the documentation doesn’t match the existing application’s reality, it is of limited use. For replacement systems, you also need to be wary of carrying forward all of the requirements, because some of the old functionality probably should not be migrated.

One of the major issues in replacement projects is validating that the reasons for the replacement are sound. There need to be justifiable business objectives for the change. When existing systems are being completely replaced, organizational processes might also have to change, which makes it harder for people to accept a new system. The change in business processes, change in the software system, and learning curve of a new system can disrupt current operations.

Requirements techniques when there is an existing system

Table 21-1 describes the most important requirements development techniques to consider when working on enhancement and replacement projects.
TABLE 21-1  Valuable requirements techniques for enhancement and replacement projects

<table>
<thead>
<tr>
<th>Technique</th>
<th>Why it’s relevant</th>
</tr>
</thead>
</table>
| Create a feature tree to show changes | ■ Show features being added.  
■ Identify features from the existing system that won't be in the new system. |
| Identify user classes | ■ Assess who is affected by the changes.  
■ Identify new user classes whose needs must be met. |
| Understand business processes | ■ Understand how the current system is intertwined with stakeholders’ daily jobs and the impacts of it changing.  
■ Define new business processes that might need to be created to align with new features or a replacement system. |
| Document business rules | ■ Record business rules that are currently embedded in code.  
■ Look for new business rules that need to be honored.  
■ Redesign the system to better handle volatile business rules that were expensive to maintain. |
| Create use cases or user stories | ■ Understand what users must be able to do with the system.  
■ Understand how users expect new features to work.  
■ Prioritize functionality for the new system. |
| Create a context diagram | ■ Identify and document external entities.  
■ Extend existing interfaces to support new features.  
■ Identify current interfaces that might need to be changed. |
| Create an ecosystem map | ■ Look for other affected systems.  
■ Look for new, modified, and obsolete interfaces between systems. |
| Create a dialog map | ■ See how new screens fit into the existing user interface.  
■ Show how the workflow screen navigation will change. |
| Create data models | ■ Verify that the existing data model is sufficient or extend it for new features.  
■ Verify that all of the data entities and attributes are still needed.  
■ Consider what data has to be migrated, converted, corrected, archived, or discarded. |
| Specify quality attributes | ■ Ensure that the new system is designed to fulfill quality expectations.  
■ Improve satisfaction of quality attributes over the existing system. |
| Create report tables | ■ Convert existing reports that are still needed.  
■ Define new reports that aren’t in the old system. |
| Build prototypes | ■ Engage users in the redevelopment process.  
■ Prototype major enhancements if there are uncertainties. |
| Inspect requirements specifications | ■ Identify broken links in the traceability chain.  
■ Determine if any previous requirements are obsolete or unnecessary in the replacement system. |

Enhancement projects provide an opportunity to try new requirements methods in a small-scale and low-risk way. The pressure to get the next release out might make you think that you don’t have time to experiment with requirements techniques, but enhancement projects let you tackle the learning curve in bite-sized chunks. When the next big project comes along, you’ll have some experience and confidence in better requirements practices.

Suppose that a customer requests that a new feature be added to a mature product. If you haven’t worked with user stories before, explore the new feature from the user-story perspective, discussing with the requester the tasks that users will perform with that feature. Practicing on this project reduces the risk compared to applying user stories for the first time on a green-field project, when your skill might mean the difference between success and high-profile failure.
Prioritizing by using business objectives

Enhancement projects are undertaken to add new capabilities to an existing application. It’s easy to get caught up in the excitement and start adding unnecessary capabilities. To combat this risk of gold-plating, trace requirements back to business objectives to ensure that the new features are needed and to select the highest-impact features to implement first. You also might need to prioritize enhancement requests against the correction of defects that had been reported against the old system.

Also be wary of letting unnecessary new functionality slip into replacement projects. The main focus of replacement projects is to migrate existing functionality. However, customers might imagine that if you are developing a new system anyway, it is easy to add lots of new capabilities right away. Many replacement projects have collapsed because of the weight of uncontrolled scope growth. You’re usually better off building a stable first release and adding more features through subsequent enhancement projects, provided the first release allows users to do their jobs.

Replacement projects often originate when stakeholders want to add functionality to an existing system that is too inflexible to support the growth or has technology limitations. However, there needs to be a clear business objective to justify implementing an expensive new system (Devine 2008). Use the anticipated cost savings from a new system (such as through reduced maintenance of an old, clunky system) plus the value of the new desired functionality to justify a system replacement project.

Also look for existing functionality that doesn’t need to be retained in a replacement system. Don’t replicate the existing system’s shortcomings or miss an opportunity to update a system to suit new business needs and processes. For example, the BA might ask users, “Do you use <a particular menu option>?” If you consistently hear “I never do that,” then maybe it isn’t needed in the replacement system. Look for usage data that shows what screens, functions, or data entities are rarely accessed in the current system. Even the existing functionality has to map to current and anticipated business objectives to warrant re-implementing it in the new system.

**Trap** Don’t let stakeholders get away with saying “I have it today, so I need it in the new system” as a default method of justifying requirements.

Mind the gap

A *gap analysis* is a comparison of functionality between an existing system and a desired new system. A gap analysis can be expressed in different ways, including use cases, user stories, or features. When enhancing an existing system, perform a gap analysis to make sure you understand why it isn’t currently meeting your business objectives.

Gap analysis for a replacement project entails understanding existing functionality and discovering the desired new functionality (see Figure 21-1). Identify user requirements for the existing system that stakeholders want to have re-implemented in the new system. Also, elicit new user requirements that the existing system does not address. Consider any change requests that were never implemented
in the existing system. Prioritize the existing user requirements and the new ones together. Prioritize closing the gaps using business objectives as described in the previous section or the other prioritization techniques presented in Chapter 16, “First things first: Setting requirement priorities.”

![Diagram](image.png)

**FIGURE 21-1** When you are replacing an existing system, some requirements will be implemented unchanged, some will be modified, some will be discarded, and some new requirements might be added.

### Maintaining performance levels

Existing systems set user expectations for performance and throughput. Stakeholders almost always have key performance indicators (KPIs) for existing processes that they will want to maintain in the new system. A key performance indicator model (KPIM) can help you identify and specify these metrics for their corresponding business processes (Beatty and Chen 2012). The KPIM helps stakeholders see that even if the new system will be different, their business outcomes will be at least as good as before.

Unless you explicitly plan to maintain them, performance levels can be compromised as systems are enhanced. Stuffing new functionality into an existing system might slow it down. One data synchronization tool had a requirement to update a master data set from the day’s transactions. It needed to run every 24 hours. In the initial release of the tool, the synchronization started at midnight and took about one hour to execute. After some enhancements to include additional attributes, merging, and synchronicity checks, the synchronization took 20 hours to execute. This was a problem, because users expected to have fully synchronized data from the night before available when they started their workday at 8:00 A.M. The maximum time to complete the synchronization was never explicitly specified, but the stakeholders assumed it could be done overnight in less than eight hours.

For replacement systems, prioritize the KPIs that are most important to maintain. Look for the business processes that trace to the most important KPIs and the requirements that enable those business processes; these are the requirements to implement first. For instance, if you’re replacing a loan application system in which loan processors can enter 10 loans per day, it might be important to maintain at least that same throughput in the new system. The functionality that allows loan processors to enter loans should be some of the earliest implemented in the new system, so the loan processors can maintain their productivity.
When old requirements don’t exist

Most older systems do not have documented—let alone accurate—requirements. In the absence of reliable documentation, teams might reverse-engineer an understanding of what the system does from the user interfaces, code, and database. We think of this as “software archaeology.” To maximize the benefit from reverse engineering, the archaeology expedition should record what it learns in the form of requirements and design descriptions. Accumulating accurate information about certain portions of the current system positions the team to enhance a system with low risk, to replace a system without missing critical functionality, and to perform future enhancements efficiently. It halts the knowledge drain, so future maintainers better understand the changes that were just made.

If updating the requirements is overly burdensome, it will fall by the wayside as busy people rush on to the next change request. Obsolete requirements aren’t helpful for future enhancements. There’s a widespread fear in the software industry that writing documentation will consume too much time; the knee-jerk reaction is to neglect all opportunities to update requirements documentation. But what’s the cost if you don’t update the requirements and a future maintainer (perhaps you!) has to regenerate that information? The answer to this question will let you make a thoughtful business decision concerning whether to revise the requirements documentation when you change or re-create the software.

When the team performs additional enhancements and maintenance over time, it can extend these fractional knowledge representations, steadily improving the system documentation. The incremental cost of recording this newly found knowledge is small compared with the cost of someone having to rediscover it later on. Implementing enhancements almost always necessitates further requirements development, so add those new requirements to an existing requirements repository, if there is one. If you’re replacing an old system, you have an opportunity to document the requirements for the new one and to keep the requirements up to date with what you learn throughout the project. Try to leave the requirements in better shape than you found them.

Which requirements should you specify?

It’s not always worth taking the time to generate a complete set of requirements for an entire production system. Many options lie between the two extremes of continuing forever with no requirements documentation and reconstructing a perfect requirements set. Knowing why you’d like to have written requirements available lets you judge whether the cost of rebuilding all—or even part—of the specification is a sound investment.

Perhaps your current system is a shapeless mass of history and mystery like the one in Figure 21-2. Imagine that you’ve been asked to implement some new functionality in region A in this figure. Begin by recording the new requirements in a structured SRS or in a requirements management tool. When you add the new functionality, you’ll have to figure out how it interfaces to or fits in with the existing system. The bridges in Figure 21-2 between region A and your current system represent these interfaces. This analysis provides insight into the white portion of the current system, region B. In addition to the requirements for region A, this insight is the new knowledge you need to capture.
Rarely do you need to document the entire existing system. Focus detailed requirements efforts on the changes needed to meet the business objectives. If you’re replacing a system, start by documenting the areas prioritized as most important to achieve the business objectives or those that pose the highest implementation risk. Any new requirements identified during the gap analysis will need to be specified at the same level of precision and using the same techniques as you would for a new system.

**Level of detail**

One of the biggest challenges is determining the appropriate level of detail at which to document requirements gleaned from the existing system. For enhancements, defining requirements for the new functionality alone might be sufficient. However, you will usually benefit from documenting all of the functionality that closely relates to the enhancement, to ensure that the change fits in seamlessly (region B in Figure 21-2). You might want to create business processes, user requirements, and/or functional requirements for those related areas. For example, let’s say you are adding a discount code feature to an existing shopping cart function, but you don’t have any documented requirements for the shopping cart. You might be tempted to write just a single user story: “As a customer, I need to be able to enter a discount code so I can get the cheapest price for the product.” However, this user story alone lacks context, so consider capturing other user stories about shopping cart operations. That information could be valuable the next time you need to modify the shopping cart function.

I worked with one team that was just beginning to develop the requirements for version 2 of a major product with embedded software. They hadn’t done a good job on the requirements for version 1, which was currently being implemented. The lead BA wondered, “Is it worth going back to improve the SRS for version 1?” The company anticipated that this product line would be a major revenue generator for at least 10 years. They also planned to reuse some of the core requirements in several spin-off products. In this case, it made sense to improve the requirements documentation for version 1 because it was the foundation for all subsequent development work in this product line. Had they been working on version 5.3 of a well-worn system that they expected to retire within a year, reconstructing a comprehensive set of requirements wouldn’t have been a wise investment.
Trace Data

Requirements trace data for existing systems will help the enhancement developer determine which components she might have to modify because of a change in a specific requirement. In an ideal world, when you’re replacing a system, the existing system would have a full set of functional requirements such that you could establish traceability between the old and new systems to avoid overlooking any requirements. However, a poorly documented old system won’t have trace information available, and establishing rigorous traceability for both existing and new systems is time consuming.

As with any new development, it’s a good practice to create a traceability matrix to link the new or changed requirements to the corresponding design elements, code, and test cases. Accumulating trace links as you perform the development work takes little effort, whereas it’s a great deal of work to regenerate the links from a completed system. For replacement systems, perform requirements tracing at a high level: make a list of features and user stories for the existing system and prioritize to determine which of those will be implemented in the new system. See Chapter 29, “Links in the requirements chain,” for more information on tracing requirements.

How to discover the requirements of an existing system

In enhancement and replacement projects, even if you don’t have existing documentation, you do have a system to work from to discover the relevant requirements. During enhancement projects, consider drawing a dialog map for the new screens you have to add, showing the navigation connections to and from existing display elements. You might write use cases or user stories that span the new and existing functionality.

In replacement system projects, you need to understand all of the desired functionality, just as you do on any new development project. Study the user interface of the existing system to identify candidate functionality for the new system. Examine existing system interfaces to determine what data is exchanged between systems today. Understand how users use the current system. If no one understands the functionality and business rules behind the user interface, someone will need to look at the code or database to understand what’s going on. Analyze any documentation that does exist—design documents, help screens, user manuals, training materials—to identify requirements.

You might not need to specify functional requirements for the existing system at all, instead creating models to fill the information void. Swimlane diagrams can describe how users do their jobs with the system today. Context diagrams, data flow diagrams, and entity-relationship diagrams are also useful. You might create user requirements, specifying them only at a high level without filling in all of the details. Another way to begin closing the information gap is to create data dictionary entries when you add new data elements to the system and modify existing definitions. The test suite might be useful as an initial source of information to recover the software requirements, because tests represent an alternative view of requirements.
Sometimes “good enough” is enough

A third-party assessment of current business analysis practices in one organization revealed that their teams did a fairly good job of writing requirements for new projects, but they failed to update the requirements as the products evolved through a series of enhancement releases. The BAs did create requirements for each enhancement project. However, they did not merge all of those revisions back into the requirements baseline. The organization’s manager couldn’t think of a measurable benefit from keeping the existing documentation 100 percent updated to reflect the implemented systems. He assumed that his requirements always reflected only 80 to 90 percent of the working software anyway, so there was little value in trying to perfect the requirements for an enhancement. This meant that future enhancement project teams would have to work with some uncertainty and close the gaps when needed, but that price was deemed acceptable.

Encouraging new system adoption

You’re bound to run into resistance when changing or replacing an existing system. People are naturally reluctant to change. Introducing a new feature that will make users’ jobs easier is a good thing. But users are accustomed to how the system works today, and you plan to modify that, which is not so good from the user’s point of view. The issue is even bigger when you’re replacing a system, because now you’re changing more than just a bit of functionality. You’re potentially changing the entire application’s look and feel, its menus, the operating environment, and possibly the user’s whole job. If you’re a business analyst, project manager, or project sponsor, you have to anticipate the resistance and plan how you will overcome it, so the users will accept the new features or system.

An existing, established system is probably stable, fully integrated with surrounding systems, and well understood by users. A new system with all the same functionality might be none of these upon its initial release. Users might fear that the new system will disrupt their normal operations while they learn how to use it. Even worse, it might not support their current operations. Users might even be afraid of losing their jobs if the system automates tasks they perform manually today. It’s not uncommon to hear users say that they will accept the new system only if it does everything the old system does—even if they don’t personally use all of that functionality at present.

To mitigate the risk of user resistance, you first need to understand the business objectives and the user requirements. If either of these misses the mark, you will lose the users’ trust quickly. During elicitation, focus on the benefits the new system or each feature will provide to the users. Help them understand the value of the proposed change to the organization as a whole. Keep in mind—even with enhancements—that just because something is new doesn’t mean it will make the user’s job easier. A poorly designed user interface can even make the system harder to use because the old features are harder to find, lost amidst a clutter of new options, or more cumbersome to access.
Our organization recently upgraded our document-repository tool to a new version to give us access to additional features and a more stable operating environment. During beta testing, I discovered that simple, common tasks such as checking out and downloading a file are now harder. In the previous version, you could check out a file in two clicks, but now it takes three or four, depending on the navigation path you choose. If our executive stakeholders thought these user interface changes were a big risk to user acceptance, they could invest in developing custom functionality to mimic the old system. Showing prototypes to users can help them get used to the new system or new features and reveal likely adoption issues early in the project.

One caveat with system replacements is that the key performance indicators for certain groups might be negatively affected, even if the system replacement provides a benefit for the organization as a whole. Let users know as soon as possible about features they might be losing or quality attributes that might degrade, so they can start to prepare for it. System adoption can involve as much emotion as logic, so expectation management is critical to lay the foundation for a successful rollout.

When you are migrating from an existing system, transition requirements are also important. Transition requirements describe the capabilities that the whole solution—not just the software application—must have to enable moving from the existing system to the new system (IIBA 2009). They can encompass data conversions, user training, organizational and business process changes, and the need to run both old and new systems in parallel for a period of time. Think about everything that will be required for stakeholders to comfortably and efficiently transition to the new way of working. Understanding transition requirements is part of assessing readiness and managing organizational change (IIBA 2009).

Can we iterate?

Enhancement projects are incremental by definition. Project teams can often adopt agile methods readily, by prioritizing enhancements using a product backlog as described in Chapter 20, “Agile projects.” However, replacement projects do not always lend themselves to incremental delivery because you need a critical mass of functionality in the new application before users can begin using it to do their jobs. It’s not practical for them to use the new system to do a small portion of their job and then have to go back to the old system to perform other functions. However, big-bang migrations are also challenging and unrealistic. It’s difficult to replace in a single step an established system that has matured over many years and numerous releases.

One approach to implementing a replacement system incrementally is to identify functionality that can be isolated and begin by building just those pieces. We once helped a customer team to replace their current fulfillment system with a new custom-developed system. Inventory management represented about 10 percent of the total functionality of the entire fulfillment system. For the most part, the people who managed inventory were separate from the people who managed other parts of the fulfillment process. The initial strategy was to move just the inventory management
functionality to a new system of its own. This was ideal functionality to isolate for the first release because it affected just a subset of users, who then would primarily work only in the new system. The one downside side to the approach is that a new software interface had to be developed so that the new inventory system could pass data to and from the existing fulfillment system.

We had no requirements documentation for the existing system. But retaining the original system and turning off its inventory management piece provided a clear boundary for the requirements effort. We primarily wrote use cases and functional requirements for the new inventory system, based on the most important functions of the existing system. We created an entity-relationship diagram and a data dictionary. We drew a context diagram for the entire existing fulfillment system to understand integration points that might be relevant when we split inventory out of it. Then we created a new context diagram to show how inventory management would exist as an external system that interacts with the truncated fulfillment system.

Not all enhancement or replacement projects will be this clean. Most of them will struggle to overcome the two biggest challenges: a lack of documentation for the existing system, and a potential battle to get users to adopt the new system or features. However, using the techniques described in this chapter can help you actively mitigate these risks.
Index

A

acceptance criteria, defined, 597
acceptance criteria, defining, 53, 347–349, 420
acceptance tests, 330, 347, 348–349
  agile projects, 146–147, 153, 161
defined, 597
project planning and, 377–379
quality attributes, 293–294
requirements and, 519
action enablers, 171–172
action plan, process improvement, 527–528
active voice, 210
activity diagrams, 153, 225, 243, 423, 597
actor, 144, 145, 147–148, 597
agile development
  acceptance criteria, 348
acceptance tests, 377, 386
adapting requirements practices for, 390–391
backlog, 387, 489
business analyst role, 71–72
change management, 389, 488–490
customer involvement, 386
defined, 597
documentation, 386
epics, user stories, and features, 388–389
estimating effort, project planning, 370–371
evolutionary prototypes, 299–300, 309
modeling on, 243–244
overview of, 381–383, 385, 387–388
priorities, setting of, 314, 387
product backlog, 387, 489
project owner, 63, 71–72, 115–116, 386, 391, 601
quality attributes, 293–294
reaching agreement on requirements, 41
requirements management, 468–470
requirements specification, 199–201, 386
use cases, 152–153
user representation, 115–116
user stories, 145–147
vision and scope in, 98–99
agreement, reaching on requirements, 38–41
allocation, requirements, 51, 373, 440–441, 532
alternative flows, use case, 152–153, 155–156, 597
ambiguity, avoiding, 205, 213–216
analysis models, 199. See also models
analysis, requirements. See also models; also priorities, setting of
defined, 597
good practices, 50–51
overview of, 15–16
risk factors, 544
troubleshooting problems, 567–569
analyst. See business analyst (BA)
application, 4
application analyst. See business analyst (BA)
architecture, 373–374
  architecture diagram, real-time projects, 445–446
defined, 597
  embedded and real-time systems projects, 440–441
requirements and, 373–374
assessment, current requirements practice, 551–557
assets, requirements engineering process, 530–533
assumption, defined, 597
assumptions, business requirements, 88, 577
assumed requirements, 140–141
assumptions, SRS document, 194, 586
atomic business rules, 174–175
attributes, requirement, 462–463. See also quality attributes
defined, 601
  requirements management tools and, 507
augmentability requirements. See modifiability requirements
author, inspection team role, 334, 336–338
availability requirements, 267–269, 274–275, 594
BA. See business analyst (BA)
backlog, 387, 460, 468–470, 489, 597
baseline, requirements, 39–41, 53, 185, 458, 459–460, 461–462, 463, 465, 597. See also change management
Beatty, Joy, 225, 322, 495
Beizer, Boris, 379
best practices. See good practices
big data, 433, 597
Bill of Responsibilities for Software Customers, Requirements, 30, 33–36
Bill of Rights for Software Customers, Requirements, 30–33
boundary values, ambiguity around, 215
Box, George E. P., 7
BPMN, 422
Brooks, Frederick, 18
Brosseau, Jim, 264
Brown, Nanette, 41
Burgess, Rebecca, 338
burndown chart, 466, 469–470
business analyst (BA). See also elicitation, requirements development; also good practices; also project planning
agile projects, 71–72
background of, 68–71
collaborative teams, creating, 72–73
decision makers, identifying, 38
defined, 598
knowledge and training, 54–55, 68–71
overview, 61
professional organizations for, xxv
reaching agreement on requirements, 38–41
roles and responsibilities, 12–13, 62–64, 459
skills required, 65–67
software requirements specification (SRS), 9
stakeholder analysis, 26–29
transitioning to agile projects, 390–391
business analytics projects
data needs, specifying, 432–435
data transformation analyses, 435–436
data, management of, 434–435
evolving nature of, 436–437
information use requirements, 431–432
overview, 427–429
prioritizing work, 430–431
requirement elicitation, overview, 429–430
business analytics system, defined, 598
business case document, 81. See also vision and scope document
business context, 90–92
business events
as scoping tool, 96
defined, 240
event-response tables, 240–242
identifying, 48–49
business intelligence. See business analytics projects
business interests, 80
business objectives, 77–79
defined, 84–85, 598
business objectives model, defined, 598
example, 86
business opportunity, 83
business process automation projects, 421–426
business process, defined, 168
business process analysis (BPA), 422
business process improvement (BPI), 422
business process management (BPM), 422
business process model and notation (BPMN), 422
business process reengineering (BPR), 422
good requirements practices, 426
modeling, 422–424
overview, 421
performance metrics, modeling, 424–426
business process flows, 225, 423, 425
business reporting. See business analytics projects
business requirements. See also vision and scope document
agile projects, scope and vision, 98–99
assumptions, and dependencies, 88
business context, 90–92
business objectives, 84–85
business opportunity, 83
business requirements section, vision and scope document, 83–88
business risks, 88
conflicting, 80–81
defined, 7–8, 78, 598
identifying and defining requirements, 78–81
judging completion with, 99
overview, 77
scope and limitations, 88–90
scope management, 97–98
scope representation techniques, 92–96
success metrics, 85–86
vision and scope document, overview, 81–88
vision and scope document, sample, 576–580
vision statement, 87–88
vs. business rules, 168
business requirements document (BRD). See software requirements specification (SRS)
business risks, 88, 577
business rules
  action enablers, 171–172
  atomic business rules, 174–175
  computations, 173–174
  constraints, 170–173
  customer input, 136
defined, 7, 10, 169, 598
discovering, 177–178
documenting, 175–177
enhancement and replacement projects, 395
facts, 170
good practices, 52
importance of, 167–169
inferences, 173
packaged solution projects, 407
requirements and, 178–180
safety requirements and, 276–277
sample, 595
taxonomy of, 169
use cases and, 156–157
business systems analyst. See business analyst (BA)

C
cardinality, 247, 598
cause-and-effect diagram, 525–526
change control. See change management
change control board (CCB)
  charter for, 481
defined, 598
good practices, 53
overview of, 480–482, 533
change management
  agile projects, 389, 488–490
  change control board, overview of, 480–482
  change control policies, 474
  change control process, 474–479, 533
  change impact analysis, 484–488, 494, 533
  customer rights and responsibilities, 32, 36
  frequency of changes, 483
good practices, 53–54
impact analysis, 53, 484–488, 494, 533
measuring change activity, 483–484
origin of changes, 483–484
outsourced projects, 419
overview, 471–472
requirements and, 519
scope management, 97–98, 472–473
tools for, 482, 506–510
troubleshooting problems, 572–574
change request, 474, 476–484
characteristics of excellent requirements, 203–207
charter, project, 81. See also vision and scope document
checklists
  change impact analysis, 485–486
defects, for requirements reviews, 338–339
defined, 530
Chen, Anthony, 225, 322, 495
Chen, Peter, 246
class diagrams, 225, 243, 248, 598
class, defined, 598
classifying business rules, 169–174
classifying customer input, 135–138
cloud solutions. See packaged solution projects
coding, project planning for, 373–377
Cohn, Mike, 388
collaborative teams. See also communication; also
elicitation, requirements development
 agile projects, 386
  business analyst role, 72–73
  customers and development, 29–30, 31, 35, 36–37
outsourced projects, 415–416, 418–419
workshops, 122–125
commercial off-the-shelf (COTS) products, defined, 598. See also packaged solution projects
commitment, to process change, 521–522
communication. See also customers; also
documenting requirements
  adoption of new systems, promoting, 401–402
  assumed and implied requirements, 140–141
business analyst role, 62–66
business analytics projects, 436–437
business process automation projects, 423–424
change control policies, 474
collaborative culture, creating, 36–37
conflicting requirements, resolution of, 116–117
elicitation activities, follow-up, 134–135
outsourced projects, 415–419
pathways for requirements, 108–109
product champions, 109–114
project planning estimates, 366–369
reaching agreement on requirements, 38–41
requirements development tools, 505–506
requirements management tools, 506–510
communications interfaces

communication. See also customers; also documenting requirements, continued
software requirement specification (SRS), good practices, 185–186
tracking requirements status, 464–466
troubleshooting problems, 564
user representatives, 108–109
writing style, requirement documentation, 208–211
communications interfaces, 197
communication protocols, requirements for, 271–272
completeness
of requirement sets, 206
of requirement statements, 204
composition, data element, 249–250
computations, business rules, 173–174
configuration requirements, COTS, 411
conflict management, 125
conflicts
resolving between stakeholder groups, 116–117
resolving between user classes, 103, 117
consistent requirements, 206
Constantine, Larry, 235
constraints
business rules, 170–173
customer input, 137
defined, 7, 10, 91, 598
design and implementation, 193, 586
quality attributes and, 291–292
real-time and embedded projects, 453
construction, requirements and, 519
context diagrams
data flow diagrams and, 227–230
defined, 598
enhancement and replacement projects, 395, 400–401
real-time projects, 442
scope representation techniques, 92–93
system external interfaces, 225
correct requirements, 204
cost. See also priorities, setting of
change impact analysis, 484–488
feasibility analysis, 50
of correcting defects, 19–20
outsourced projects, 416, 418–419
prioritizing requirements and, 315, 317, 322–326
quality attribute requirements, 268, 288–290
requirement reuse, benefits of, 351–352
requirements management, 463
requirements tools, 504–505, 511
tracking effort, 467–468
COTS (commercial off-the-shelf) products.
See packaged solution projects defined, 598
cross-functional diagrams. See swimlane diagrams
CRUD matrix, 251–252, 598
cultural differences, outsourced projects, 418–419
culture, organizational
creating respect for requirements, 36–37
process improvement fundamentals, 522–524
requirements tools and, 513
resistance to change, 521–522
current practices, assessing, 526–527, 551–557
customer input, classifying, 135–138
customers. See also communication; also documenting requirements, continued
collaborative culture, creating, 36–37
customer input, classifying, 135–138
decision makers, identifying, 38
defining, 27–29, 598
demands, 125
expected gap, 26–27
reaching agreement on requirements, 38–41
relationships with, overview, 25–26
Requirements Bill of Responsibilities for, 30, 33–36
Requirements Bill of Rights for, 30–33
stakeholders and, 27–29

cyclical complexity, 286

D

DAR (display-action-response) models, 375–377
dashboard reporting, 257–258, 431–432, 598
data analysis, requirements, 251–252. See also data requirements
business analytics projects, 432–435
defining, business analytic projects, 435–436
enhancement and replacement projects, 400
packaged solution projects, 407
data definitions, models for, 225
data dictionaries, 248–251
business analytics projects, 433
defined, 598
good practices, 50
sample, 589
SRS document, 195
use cases and, 164
documenting requirements

data field definitions, 226
data flow diagrams (DFD), 226–230
defined, 598
enhancement and replacement projects, 400–401
uses for, 225
data modeling, 245–248
enhancement and replacement projects, 395
data object relationships, models for, 225
data requirements. See also business analytics projects
COTS implementation, 412
customer input, 137
dashboard reporting, 257–258
data analysis, overview, 251–252
data dictionary, overview of, 248–251
data integrity requirements, 270–271
management and use requirements, 434–435
modeling data relationships, 245–248
overview, 245
packaged solution projects, 412
sample, 589–592
security requirements, 277–279
specifying reports, 252–256
SRS document, 195
Davis, Alan, 315
decision makers, identifying, 38
decision rule, 38, 598
decision tables, 226, 239–240, 598
decision trees, 51, 226, 239–240, 599
defect checklist for requirements reviews, 338–339
defects, cost of correcting, 19–20
degree of freedom, defined, 91
delivery dates, 372
dependencies, business requirements, 88, 577
dependencies, SRS document, 194, 586
dependency, defined, 599
deployment considerations, vision and scope document, 92, 580
deriving requirements
from business rules, 178–180
from models, 223
from nonfunctional requirements, 290
from system requirements, 440–441
from use cases, 160, 162
design, requirements and, 373–377
detail, level of requirements, 211–212, 386
development life cycle, good practices, 56
DFD. See data flow diagrams
dialog maps
defined, 599
enhancement and replacement projects, 395, 400–401
good practices, 51
overview of, 235–238
testing and, 344–346
wireframes, 299
disfavored user classes, 103–104
display-action-response (DAR) model, 375–377
document analysis, 128–129, 177
document, use of term, 8
documentation. See also data dictionary; also vision and scope document
agile projects, 386
business analyst task, 64
business rules, documenting, 175–177
document analysis, good practices, 49
elicitation activities, follow-up, 134–135
elicitation activities, notes from, 133
enhancement and replacement projects, 395, 398–401
interface specifications, 446–447
outsourced projects, requirements details, 416–417
project risks, 539–541
requirement patterns, 358–359
requirements engineering process assets, 530–533
requirements process and, 518–520
requirements repositories, 359–360, 362–364
requirements reuse, 354–355
requirements, good practices, 51–52
templates, requirements documents, 51
user documentation, 519–520
documenting requirements. See also models
agile projects, 199–201
ambiguity, avoiding, 213–216
before and after examples, 217–220
characteristics of excellent requirements, 204–207
labeling requirements, 186–188
level of detail, 211–212
overview, 181–183
representation techniques, 212–213
software requirements specification (SRS), 183–190
SRS template, 190–199
system or user perspective, 207–208
use case template, 150
documents, limitations of
documenting requirements. See also models, continued
vision and scope document template, 81–92
writing style, 208–211
documents, limitations of, 1–2, 503–504
driver, defined, 91
Dyché, Jill, 433

E
ecosystem maps, 50, 94, 225, 395, 599
educating stakeholders and developers, 44, 55, 58
efficiency requirements, 281–282, 450
effort estimates, 370–372, 467–468. See also project planning
electronic prototypes, 301–303
elicitation, requirements, 16, 119–142. See also use cases; also user stories
assumed and implied requirements, 140–141
availability requirements, 268–269
business analytics projects, 429–430
business process automation, 422–424
business rules, discovering, 177
cautions about, 139–140
completion of process, 138–139
customer input, classifying, 135–138
defined, 599
document analysis, 128–129
efficiency requirements, 282
focus groups, 124–125
follow-up activities, 134–135
framework for, 45–47
good practices, 44, 48–49
installability requirements, 270
interoperability requirements, 272
interviews, 121–122
missing requirements, identifying, 141–142, 222, 225, 227, 236, 238, 346
observations, 125–126
overview, 119–121
performance requirements, 266
planning for, 129–130
portability requirements, 284
preparing for, 130–132
quality attributes, 263–266
questionnaires, 127
reliability requirements, 274–275
reporting requirements, 253–254
reusability requirements, 284–285
risk factors, 543–544
robustness requirements, 275
safety requirements, 277
scalability requirements, 277
scope creep, managing, 473
security requirements, 277–279
system interface analysis, 127–128
tips for performing, 132–134
tools for, 505
troubleshooting problems, 565–566
usability requirements, 280
user interface analysis, 128
verifiability requirements, 287
workshops, 122–125
embedded systems projects
defined, 599
interfaces, 446–447
modeling, 441–446
overview, 439, 453–454
quality attributes, 449–453
system requirements, architecture, and allocation, 440–441
timing requirements, 447–449
end users. See users
enhancement projects
adoption of new system, 401–402
iteration and, 402–403
lack of existing documentation, 398–401
overview of, 393–394
prioritizing using business objectives, 396–397
requirements techniques, 394–395
entity, 246–247, 251–252, 599
entity-relationship diagrams
business analytics projects, 433
defined, 599
enhancement and replacement projects, 400–401
good practices, 51
modeling data relationships, 225, 245–248
entry criteria
for change control, 475, 478
for inspections, 335
environment, real-time systems, 449–453
epics, 388–389, 599
error handling, real-time systems, 450–452
estimation. See also project planning
project size and effort, 370–372
requirements effort, 366–369
evaluating packaged solutions, 408–410
evaluating process improvement efforts, 529–530
functional requirements

events
  as scoping tool, 96
  defined, 599
  event list, 96

event-response tables, 9, 226, 240–242, 443–444, 599. See also user requirements
  identifying, good practices, 48–49

evolutionary prototypes, 298–300, 342, 599. See also prototypes

excellent requirements, characteristics of, 203–207

exception handling, 152–153, 275

exceptions, use cases, 147, 151, 152–153, 159

exception, defined, 599

execution time, 447

exit criteria
  for change control, 475, 479
  for inspections, 338

expectation gap, 26–27, 102, 295

extend relationship, use cases, 155–156, 599

extensibility requirements. See modifiability requirements

extension requirements, COTS, 412

external entities, 92–93, 227–228, 271–272, 599

external events, 48–49, 92–93

external interface requirements
  customer input, 137
    defined, 7, 599
  SRS document, 196–197
  SRS document, sample, 592–593

Extreme Programming. See agile development

F

facilitation
  business analyst skills, 66
  completing elicitation sessions, 138–139
  elicitation activities, cautions about, 139–140
  elicitation activities, follow-up, 134–135
  elicitation activities, performing, 132–134
  focus groups, 124–125
  preparing for elicitation, 130–132
  workshops, 122–125

facilitator, defined, 599

facts, business rules, 170

Fagan, Michael, 333

fault
  detection, 451
  logging, 451
  prevention, 451
  recovery, 451
  tolerance, 275–276, 450–452

fault tree analysis, 452

favored user classes, 103, 117

feasibility analysis, 50

feasible requirements, 204

Feature Driven Development. See agile development

feature trees, 11, 95–96, 395, 599

features
  agile projects, 388–389
  defined, 7, 11, 599
  enhancement and replacement projects, 395–397
  example, 95, 578
  gap analysis, 396–397
  packaged solution projects, 406–410
  prioritizing, 50
  requirements reuse, 356–358
  risk management, 544
  SRS document, 194
  SRS document, sample, 586–588
  vision and scope document, 89–90

finding missing requirements, 141–142, 222, 225, 227, 236, 238, 346

fishbone diagram, 525–526

fit criteria, 267, 330

flexibility requirements. See modifiability requirements

flow diagrams, business process, 225, 423, 425

flowcharts, 153, 225, 226, 230, 236, 425, 599

flows, data, 92–93, 226–229

focus groups, 48, 108–109, 124–125

formal reviews. See inspections

function point, 370, 599

functional requirements
  architecture design, project planning and, 373–374
  business analytic projects, 435–436
  business rules and, 180
  customer input, 136
    defined, 7, 9, 599
  deriving, from business rules, 178–180
  deriving, from models, 223
  deriving, from nonfunctional requirements, 290
  deriving, from system requirements, 440–441
  deriving, from use cases, 160, 162
  enhancement and replacement projects, 396–397
  missing, 141–142, 222, 225, 227, 236, 238, 346
  prioritizing, 50, 315, 318, 319, 324
  requirement levels and types, 7–13
  reusing, 356–358
  specification of, 209–219
functional specification

functional requirements, continued
use cases and, 160, 161–163
writing, 209–219
functional specification. See software requirements specification (SRS)

G

gap analysis, 396–397, 412, 599
Gause, Donald, 105
Gilb, Tom, 187, 287, 600
glossary
good practices, 55, 199
reuse of, 353, 356, 364
goals, business. See business objectives
goals, requirements process improvement, 533–535
gold plating, 21, 600
good practices
ambiguous terms, avoiding, 213–216
analysis, 50–51
application of, 57–58
elicititation, 48–49
inspections, 333, 339–342
knowledge, 54–55
overview, 43–45
project management, 56–57
project planning, 379–380
prototypes, 310
reporting specifications, 254–255
requirement statements, documenting, 204–207
requirements development process framework, 45–47
requirements management, 53–54
requirements reuse, 360–364
specification, 51–52
validation, 52–53
writing style, requirements documentation, 208–211
Gottesdiener, Ellen, 72, 105, 122–123
government regulations. See business rules
Graham, Dorothy, 377
green-field project, 393, 600

H

hard real-time systems, 439. See also real-time
systems projects
hardware interfaces, 197
hardware requirements, 441
Hardy, Terry, 452
hazard analysis, 452
Herrmann, Debra, 452
hierarchical textual tags, 179, 187–188, 288, 587–588
high-resolution prototypes, 226
history of requirements changes, 54
Hoffman, Cecilie, 338
horizontal prototype, 297–298, 600. See also prototypes
hundred-dollar approach, prioritization, 321–322

I

identifiers, SRS documents, 186–188
IIBA (International Institute for Business Analysis), xxv
impact analysis, requirements changes, 53, 484–488, 494, 533
implied requirements, 140
in-or-out prioritization, 318
include relationships, use cases, 155–156, 600
incompleteness, in requirements documents, 188–189, 216–217
inferences, business rules, 173
initial release, scope of, 89–90
inspections, 52, 332–342, 600. See also peer reviews
installability requirements, 269–270
integration requirements, COTS, 412
integrity requirements, 270–270, 408
interfaces
analyzing, good practices, 51
architecture diagrams, 445–446
customer input, 137
dialog maps, 235–238
embedded projects, 446–447, 453
enhancement and replacement projects, 400–401
external interface requirements, 7, 10, 196–197, 592–593, 599
functional requirements, defined, 10
interface specification document, 447
mock-ups, 297–298
models for, 225–226
prototypes, 50, 299
real-time projects, 446–447, 453
SRS document, 189–190, 196–197
SRS document, sample, 592–593
system interface analysis, 127–128
user interface analysis, 128
internationalization requirements, 198
models

interoperability requirements, 271–272, 408

elicitation of requirements, 49, 121–122

skills required, 65

Ishikawa diagram, 525–526

issue, requirements, defined, 600

issue tracking, 54, 466–467

IT business analyst. See business analyst (BA)

iteration,

agile projects, 21, 56, 370, 371, 385–389, 468–470, 489

defined, 600

design, 374

requirements development, 13, 17

specifying requirements for, 46, 47

J

Joint Application Design (JAD), 49

K

Kanban. See agile development

key performance indicator model (KPI), 397, 423–426

key performance indicators (KPIs), 425, 533–535

knowledge, business analyst role, 68–71

knowledge, good practices around, 54–55

Koopman, Philip, 448, 452

Kudish, Joseph, 442–443

L

labeling requirements, 186–188

latency, 447

Lauesen, Soren, 267

Lavi, Johan, 442–443

Lawrence, Brian, 6

lean software development. See agile development

learning curve, process improvement efforts, 529–530

Leffingwell, Dean, 348

legacy systems. See also enhancement projects; also

replacement projects

business rules and, 177

requirements reuse, 357–358

levels and types of requirements, 7–13

Leveson, Nancy, 452

life cycles, development, 46–47, 330. See also agile
development; also waterfall development

listening skills, 65

localization requirements, 10, 198

Lockwood, Lucy, 235

logging, faults, 451–452

logical data model, 195

low-fidelity prototypes, 301–303

low-resolution prototypes, 226

M

maintainability requirements, 267, 282, 283

management, project. See project management

management, requirements. See requirements

management

management commitment to excellent

requirements, signs of 521–522

market requirements document (MRD), 81. See also

vision and scope document

Martin, James, 247

mean time between failures (MTBF), 267, 274

mean time to repair (MTTR), 267

measuring

change activity, 483–484

requirements management effort, 467–468

metadata, 433

metrics

business performance, 424–426

key performance indicators, 425, 533–535

process improvement, 533–535

project size, 370

requirements change activity, 483–484

requirements process improvements, 533–535

success, 78, 85–86

Miller, Roxanne, 266–267

minimum marketable feature (MMF), 389

missing requirements, identifying, 141–142, 222, 225,

227, 236, 238, 346

mitigation, risk, 539, 541–542

mock-ups, 300, 342, 600. See also prototypes

models

agile projects, 243–244

business analyst role, 67

business analytics projects, 433

business objectives models, 86, 598

business process automation, 422–424

business process model and notation (BPMN), 422

business rules, discovering, 177

context diagrams, 92–93, 598

customer comments, use of, 223–224

DAR (display-action-response) model, 375–377

data flow diagrams, 226–230, 598

data relationship modeling, 245–248

627
moderator, inspection team role

models, continued
decision tables and decision trees, 239–240, 598–599
dialog maps, 235–238, 599
ecosystem maps, 95, 599
embedded projects, 441–446
enhancement and replacement projects, 395, 400–401
entity-relationship diagrams, 245–248, 599
event-response tables, 240–242, 599
feature trees, 95–96, 599
good practices, 51
missing requirements, identifying, 141–142, 222, 225, 227, 236, 238, 346
outsourced projects, 417–418
overview of, 222–223
real-time projects, 441–446
requirements elicitation, 122, 131–132
scope representation techniques, 92–96
selection of appropriate, 225–226
simulations, good practices, 53
SRS document, 199
state tables, 232–234, 602
state-transition diagrams, 232–234, 602
swimlane diagrams, 230–231, 602
tools for drawing, 506
UML diagrams, 243
moderator, inspection team role, 334, 336, 338
modifiability requirements, 282–283, 408
modifiable requirements, 206
MoSCoW prioritization, 320–321

N
NAH (not applicable here), 362
navigation map, 235. See also dialog maps
necessary requirements, 204
negative requirements, clarifying, 216
NIH (not invented here), 362
nonfunctional requirements, 261–294. See also
constraints; also external interface
requirements; also quality attributes
agile projects, 293–294
COTS projects, 208
defined, 7, 10–11, 600
packaged solution projects, 208
real-time and embedded systems, 449–453
requirement levels and types, 7–13
requirements traceability, 497–498
risk management, 543
specifications, good practices, 52
non-human users, 104
normal flow, use cases, 152–153, 155–156, 600
numbering requirements, SRS documents, 186–188

O
object state models, 226
objectives, business
business objectives model, 86, 598
business objectives, defined, 598
completion decisions and, 99
success metrics, 85–86
vision and scope document, 84–87
observational skills, 66
observations, requirements elicitation, 125–126
on-site customer, 25, 115–116
operating environment, SRS document, 193
operational profile, 287, 409, 600
organization chart analysis, 105
organizational culture
creating respect for requirements, 36–37
process improvement fundamentals, 522–524
requirements tools and, 513
resistance to change, 521–522
organizational policies. See business rules
out-of-scope requirements, 78, 90, 97
outsourced projects
acceptance criteria, 420
acquirer-supplier interactions, 418–419
change management, 419
level of requirements detail, 416–417
overview of, 415–416

P
packaged solution projects
common challenges, 413–414
configuration requirements, 412
costs, 406, 408–409
evaluating candidates, 408–409
extension requirements, 412
identifying requirements, 406–410
implementation requirements, 411–413
integration requirements, 412
overview, 405–406
solution selection, 406, 408–409
pairwise comparisons for prioritization, 264–265, 318
paper prototypes, 301–303, 600
parking lots, 123
passaround review, 332–333
peer reviews. See also inspections
challenges, 340–342
defect checklist for requirements, 338–339
defined, 600
during elicitation, 160–161
good practices, 52
outsourced projects, 418
review process, 332–333
tips for performing, 339–340
performance. See also quality attributes
efficiency requirements, 281–282
enhancement and replacement projects, 397
packaged solution projects, 408
real-time and embedded systems, 449–453
requirements, 266, 272–273, 408, 449, 593
SRS document, 197–198
timing requirements, real-time systems, 447–449

 personas, user, 107–108
pilot, defined, 600
pilots, process improvement, 526, 528–529
plan, defined, 530
Planguage, 226, 266–267, 287–288
defined, 600
policies, company. See business rules
policy, defined, 530
portability requirements, 283–284
postconditions, use cases, 151, 156, 158–159
defined, 600
preconditions, use cases, 151, 156, 158–159, 600
predictability, timing requirements, 448
primary actor, 148
primitive data elements, 250. See also data dictionary
priorities, setting of
agile projects, 387
business analytics projects, 430–431
enhancement and replacement projects, 396–397
importance of, 313–315
prioritization, defined, 600
project, 91–92
quality attributes, 263–267
Quality Function Deployment (QFD), 322
requirements prioritization procedure, 322–327, 532
risk factors, 544
strategies and techniques for, 315–322
prioritization. See priorities, setting of
priority, as a requirement attribute, 319, 462
problem reports as source of requirements, 49
procedure, defined, 530, 600
process assets, 530–533, 600
process description, defined, 531
process flows, 225, 423, 425, 600
process improvement action plan, 527–528
process improvement. See requirements process improvement
process, defined, 600
product backlog, 387, 406, 468–470, 597
product champions, 109–114, 117, 601
product features. See features
product line, 352, 356–357
product owner, 63, 71–72, 115–116, 386, 391, 601
product requirements vs. project requirements, 14–15
product vision, 78–79, 87–88, 577, 603
product, defined, 4, 600
product-centric strategy, 16
project charter, 81. See also vision and scope
document
project management. See also good practices; also project planning; also risk management
collaborative teams, creating, 72–73
good practices for, 56–57
outsourced projects, 418–419
reaching agreement on requirements, 38–41
requirement process improvement and, 518–520
stakeholder analysis, 27–29
project manager, as business analyst, 70
project planning. See also project management
designing and coding, 373–377
estimating project size and effort, 370–372
estimating requirements effort, 366–369
good practices, 56–57, 379–380
outsourced projects, 418–419
overview of, 365–366
requirements and, 519
requirements effort, estimating, 366–369
risk management, 543, 545
scheduling, requirements and, 372
scope creep, managing, 472–473
testing, 377–379
tracking effort, 467–468
tracking requirements status, 464–466
project priorities, 91–92. See also priorities, setting of
project requirements, vs. product requirements, 14–15
project scope

project scope. See also change management; also project planning; also vision and scope document
agile projects, change management, 389
assumed and implied requirements, 140–141
change control policies, 474
completion decisions, 99
defined, 79, 602
defining for project, 13, 139–140
elicitation, good practices, 48–49
enhancement and replacement projects, 396–397
estimating effort, 370–372
good practices, 53–54
identifying and defining requirements, 78–81
outsourced projects, 419
packaged solution projects, 406–410
product vision and, 78–80
project management good practices, 56–57
requirements baseline, 459–460
requirements elicitation, 122–123
scope creep, 20–21, 472–473, 602
scope management, 97–98
scope representation techniques, 92–96
troubleshooting change management problems, 572–574
vision and scope document, overview, 81–83
vision and scope document, sample, 576–580
project tracking, requirements and, 519
proof-of-concept prototypes, 297–298, 300, 342, 601
prototypes
dashboard reporting, 258
defined, 601
electronic prototype, 302–303
enhancement and replacement projects, 395
evaluating, 306–307
evolutionary prototype, 599, 299–300
good practices, 50, 310
horizontal prototype, defined, 297, 600
mock-up, 297–298, 600
outsourced projects, 417–418
overview of, 295–297
paper prototype, 301–302, 600
proof-of-concept, 298, 601
real-time projects, 446
reporting specifications, 255
requirement validation and, 342
risks of, 307–310
throwaway prototype, 298–299, 602–603
tools for creating, 505
user interfaces, 189–190, 226
vertical prototype, defined, 298, 603
working with, 303–306
Pugh, Ken, 348

Q

QFD. See quality function deployment
quality assurance. See also testing
nonfunctional requirements, defined, 10
requirements reuse, 364
software requirements specification (SRS), 9
quality attributes. See also performance
agile projects, 293–294
availability, 267–269, 594
constraints on, 291–292
customer input, 137
defined, 7, 10, 261–263, 601
defining, overview, 267
efficiency, 281–282, 450
embedded systems, 449–453
enhancement and replacement projects, 395
identifying and prioritizing, 263–267
implementation of, 290–291
installability, 269–270
integrity, 270–271, 408
interoperability, 271–272, 408
modifiability, 282–283, 408
overview of, 261–263
packaged solution projects, 408
performance, 266, 272–273, 408, 449, 593
Planguage, 287–288
prioritizing, 264–265
real-time systems, 449–453
reliability, 274–275, 450
requirements traceability, 497–498
reusability, 284–285
robustness, 275–276, 450, 594
safety, 276–277, 452, 593
scalability, 285–286
security, 277–279, 408, 452–453, 593
SRS document, 197–198
SRS document, sample, 593–594
timing requirements, real-time systems, 447–449
trade-offs, 288–290
usability, 279–281, 453, 593
verifiability, 286–287, 453, 593
Quality Function Deployment (QFD), 322
quality of service requirements. See quality attributes
questionnaires, good practices, 49, 127

R

rank ordering, prioritization, 318
Rational Unified Process, 47
rationale, as a requirements attribute, 462, 463
reader, inspection team role, 335, 337
real-time systems projects
defined, 601
interfaces, 446–447
modeling, 441–446
overview, 439, 453–454
quality attributes, 449–453
system requirements, architecture, and
allocation, 440–441
timing requirements, 447–449
recorder, inspection team role, 335
recoverability, 275–276
reengineering project. See replacement projects
regulations, government. See business rules
relationship, 247
reliability requirements, 274–275, 450
repeating group, data elements, 251. See also data
dictionary
replacement projects
adoption of new system, 401–402
iteration and, 402–403
lack of existing documentation, 398–401
overview of, 393–394
prioritizing using business objectives, 396–397
requirements techniques, 394–395
reports. See also business analytics projects
business analytics projects, 431–432
dashboard reporting, 257–258
enhancement and replacement projects, 395
report layouts, 225
specifications for, 252–256
SRS document, 195, 591
representation techniques, 212–213
requirement, defined, 5–6, 601
requirement attributes, 462–463, 51, 54, 601
requirement pattern, defined, 601
requirements allocation procedure, 532, 601
requirements analysis. See analysis, requirements
requirements analyst. See business analyst (BA)
Requirements Bill of Responsibilities for customers,
30, 33–36
Requirements Bill of Rights for customers, 30–33
requirements development. See also analysis,
requirements; also elicitation,
requirements; also specification,
requirements; also validation,
requirements
common problems, 19–22
defined, 15, 601
overview, 15–17
process assets for, 531–532
process framework for, 45–47
requirements management, boundary between, 18
tools for, 503–506
requirements document. See software requirements
specification (SRS)
requirements elicitation. See elicitation, requirements
requirements engineer. See business analyst (BA)
requirements engineering
common problems, 19–22
defined, 15, 601
framework for, 45–47
process assets for, 530–533
requirements development, 15
requirements management, 17–19
subdisciplines of, 15
tools for, 503–514
requirements levels and types, 7–13
requirements management. See also change
management; also tracing, requirements
agile projects, 468–470
baselining, 459–460
common problems, 19–22
defined, 17–18, 458, 601
good practices, 53–54
measuring effort, 467–468
overview, 15, 17–19, 46–47, 470
process assets for, 531–533
process overview, 457–459
product backlog, 387
project planning estimates, 366–372
requirements attributes, 462–463
requirements development, boundary between, 18
requirements repositories, 359–360
resolving issues, 466–467
risk factors, 546
tools for, 503–510
tools, selecting and using, 510–513
tracking status, 464–466
troubleshooting problems, 571
version control, 460–462
requirements manager. See business analyst (BA)
requirements mapping matrix, 495
requirements practices self-assessment, 551–557
requirements prioritization procedure, 532
requirements process improvement
    action planning for, 527–528
    assessment of current practices, 526–527, 551–557
    fundamentals of, 522–524
    learning curve, 529–530
    management commitment to, 522
metrics for, 533–535
overview, 517–520
process assets, 530–533
process improvement cycle, 526–530
resistance to change, 521–522
road map for, 535
root cause analysis, 524–526
requirements review checklist, 338–339, 532
requirements specification. See specification,
    requirements; also software requirements
    specification (SRS)
requirements status tracking procedure, 532
requirements traceability matrix, 54, 495–498, 601.
    See also tracing, requirements
requirements tracing. See tracing, requirements
requirements validation. See validation, requirements
requirements, characteristics of excellent, 203–207
requirements, reuse of
    benefits of, 351–352
    common scenarios for, 356–358
    defined, 602
dimensions of, 352–355
good practices for, 360–364
quality attributes, reusability, 284–285
requirement patterns, 358–359
tools for, 359–360, 508
tracing requirements, 495
types of information to reuse, 355–356
requirements, troubleshooting problems with
    analysis issues, 567–569
    barriers to solution implementation, 560
    change management issues, 572–574
    communication issues, 564
    elicitation issues, 565–566
    overview, 559
    planning issues, 562–564
    process issues, 561–562
    product issues, 562
    requirements management issues, 571
    signs of problems, 559–560
    specification issues, 569–570
    validation issues, 570–571
response time, 266, 287–288
retrospective, 337, 601
reusability requirements, 284–285
reuse. See requirements, reuse of
reviewing requirements. See peer reviews
rework, 19, 521, 534
risk, 537, 602
risk management
    documenting project risks, 539–541
    overview, 537–539, 546
    planning for, 542
    requirements analysis, 544
    requirements elicitation, 543–544
    requirements management, 546
    requirements specification, 545
    requirements validation, 545
    risk assessment, 539
    risk avoidance, 539
    risk mitigation, 539, 541–542
    risks, business, 88, 577
    risks, technical, and requirements prioritization,
        322–323, 325–326
road map, for process improvement, 535
Robertson, James, 267
Robertson, Suzanne, 267
robustness requirements, 275–276, 450–452, 594
roles and permissions matrix, 171–172
root cause analysis, 524–526, 602
Rothman, Johanna, 326
Royce, Winston, 384
S
SaaS. See software as a service
safety requirements, 276–277, 452, 593
sample documents
    business rules, 595
    software requirements specification (SRS), 584–594
    use cases, 581–583
    vision and scope document, 576–580
Sawyer, Pete, 6
scalability requirements, 285–286, 290–291
scenarios, 149, 602
schedule. See project planning
scope creep, 20–21, 472–473
scope, project. See also change management; also
    product vision; also project planning; also
    vision and scope document
agile projects, change management, 389
change control policies, 474
completion decisions, 99
defined, 79, 602
defining for project, 13, 139–140
elicitation, good practices, 48–49
enhancement and replacement projects, 396–397
estimating effort, 370–372
good practices, 53–54
identifying and defining requirements, 78–81
outsourced projects, 419
packaged solution projects, 406–410
project management good practices, 56–57
requirements baseline, 459–460
requirements elicitation, 122–123
requirements process improvement, 519
risk management, 543–544
scope creep, defined, 602
scope management, 20–22, 97–98, 472–473
scope representation techniques, 92–96
vision and scope document, overview, 81–83, 532
vision and scope document, sample, 576–580
Scrum. See agile development
secondary actor, 148
secondary scenarios, 152–153
security
data integrity requirements, 270–271
packaged solution projects, 408
real-time and embedded systems, 452–453
requirements for, 277–279, 408, 452–453, 593
requirements reuse, 355–356
SRS document, 198
self-assessment, current requirements practices, 551–557
shall, as keyword in requirements, 9, 209
sign-off, 39–41. See also baseline, requirements
signal events
defined, 241
event-response tables, 240–242
identifying, 48–49
simulations. See also prototypes
good practices, 53
mock-ups and proofs of concept, 297–298
user interfaces, 189–190
skill development, good practices, 54–55
SMART, 266, 347
soft real-time systems, 439. See also real-time
systems projects
software as a service (SaaS) projects. See packaged
solution projects
software design, requirements and, 373–377
software development life cycle, defined, 602
software interfaces, SRS document, 197, 592–593.
See also interfaces
software process improvement. See requirements
process improvement
software requirements
defined, 5–6
deriving from system requirements, 440–441
levels and types, 7–13
Software Requirements Bill of Responsibilities for
customers, 30, 33–36
Software Requirements Bill of Rights for customers,
30–33
software requirements specification (SRS). See also
documenting requirements
audiences for, 184
defined, 9, 183, 602
labeling requirements, 186–188
lack of, on enhancement and replacement
projects, 398–401
outsourced projects, 416–417
overview, 13, 183–186, 532
product vs. project requirements, 14–15
requirements baseline, 459–460
requirements traceability matrix, 495–498
sample document, 584–594
template for, 190–199
user classes, 106
user interfaces and, 189–190, 196–197
solution ideas, customer input, 138
solution, defined, 602
Sommerville, Ian, 6
specification, requirements. See also software
requirements specification (SRS)
agile projects, 201–202
defined, 602
good practices summary chart, 44
good practices, 51–52
requirements development framework, 45–47
requirements development, 15, 17
risk factors, 545
troubleshooting problems, 569
SRS. See software requirements specification (SRS)
stakeholder, defined, 602
stakeholders. See also customers; and also users
business context, vision and scope document,
90–92
decision makers, identifying, 38
elicitation session, preparing for, 131
standards, industry

stakeholders. See also customers; also users, continued

knowledge and training, good practices, 54–55
list of potential, 28
overlooked, 22
reaching agreement on requirements, 38–41
Requirements Bill of Responsibilities for customers, 30, 33–36
Requirements Bill of Rights for customers, 30–33
requirements process improvement, 520
resistance to change, 521–522
stakeholder analysis, 27–29
standards, industry. See business rules
state diagrams, 243
state machine diagrams, 232–234, 602
state tables, 226, 232–234, 602
statechart diagrams, 443
state-transition diagrams, 51, 226, 232–234, 442–443, 594, 602
status tracking, requirements, 457–459, 464–466, 469–470, 532
story points, 325, 370, 469
storyboards, 226, 301–303
straw man models, 122, 132
structure, data, 250. See also data dictionary
subject matter expert, 62, 70–71, 110, 602
success metrics, 85–86, 577
supportability requirements. See modifiability requirements
surveys, good practices, 49
survivability, 275
swimlane diagrams
business process automation projects, 423
business process flow, 225
defined, 230, 602
enhancement and replacement projects, 400–401
overview of, 230–231
system external interfaces, 225
user task descriptions, 226
system, defined, 9–10, 439, 602
system analyst. See business analyst (BA)
system interface analysis, 127–128, 225
system requirements
allocation, 9–10, 440–441
architecture design, project planning and, 373–374
defined, 7, 9–10, 602
embedded and real-time systems projects, 440–441
partitioning of, 440–441
system requirements specification, 440
system state models, 226
system testing, requirements and, 519

taxonomy, business rules, 169
TBD (to be determined), 206, 208, 216, 221, 602
team building, 72–73
templates
change control board charter, 481, 533
change control process, 475–479
change impact analysis, 488
defined, 602
functional requirements, 207–208
interface specification document, 446–447
project risk documentation, 539–541
reporting specifications, 255–256
requirement patterns, 358–359
software requirements specification (SRS), 190–199, 532
tips for using, 82–83
use case, 146, 532
user story, 145
vision and scope document, 81–83, 532
vision statement, 87
temporal events
defined, 241
event-response tables, 241–242
identifying, 48–49
terminators, context diagrams, 92–93. See also external entities
terminology, good practices, 55, 364
testability. See verifiability
testing
acceptance criteria, 347–349
creating validation tests, 342–347
dialog maps and, 344–347
enhancement and replacement projects, 400–401
fit criteria, 267
outsourced projects, 416, 420
packaged solution projects, 408–409
project planning and, 365–366, 377–379
prototype evaluations, 306–307
requirements process improvement, 518–520
requirements reuse and, 362
software requirements specification (SRS), 9
tracing requirements to tests, 495
troubleshooting issues, 570
use cases and functional requirements, 163
use cases and user stories, 146–147
use cases and, 160–161, 346–348
validating use cases, 160–161
validation, good practices, 52–53
verifiability requirements, 286–287
textual tags, requirement labeling, 187–188
three-level scale, prioritization, 319–320
throwaway prototypes, 298–300, 602. See also prototypes
time-based events. See temporal events
timeboxed development, 98–99. See also agile development
timeboxing discussions, workshops, 124
timing requirements, on embedded and other real-time systems, 447–449
to be determined. See TBD
tools for requirements engineering
overview, 503–505
requirements development tools, 505–506
requirements management tools, 506–510
selecting and using, 510–513
traceable requirements, 206
tracing requirements
allocated requirements, 441
defined, 603
levels and types, 7–13
missing requirements, identifying, 141–142, 222, 225, 227, 236, 238, 346
motivations for, 494–495, 500–501
overview, 491–493
packaged solution projects, 407, 410
procedure for, 499–501, 533
requirements management overview, 457–459
requirements traceability matrix, 495–498
tools for, 498–499
traceability data, 400
traceability table, 495
tracking changes, 461–462, 474
tracking effort on requirements activities, 467–468
tracking requirements status, 458, 464–466, 469
training and skills development, 54–55, 68–71
transition requirements, 14, 22, 402
troubleshooting
analysis issues, 567–569
barriers to implementing solutions, 560
change management issues, 572–574
communication issues, 564
elicitation issues, 565–566
overview, 559
planning issues, 562–564
process issues, 561–562
product issues, 562
requirements management issues, 571
signs of requirements problems, 559–560
specification issues, 569–570
validation issues, 570–571
understandability requirements. See modifiability requirements
UML diagrams, 243
Unified Modeling Language (UML), 148–149, 232, 243, 445–446, 603
usability. See also quality attributes
embedded systems, 453
packaged solution projects, 408
prototype evaluations, 306–307
requirements, 279–281
SRS document, 197–198
usage-centric strategy, 16
usage scenarios, 149
use cases. See also user requirements
actors and roles, 147–148
benefits of, 164–165
business rules and, 156–157
chaining together, 156
defined, 144, 603
diagrams, 148–149
elements of, 149–150
eliciting use cases, 158–160
enhancement and replacement projects, 400–401
extend and include relationships, 155–156
functional requirements and, 161–163
identifying, 157–158
labeling conventions, 151
normal flow, alternative flows, and exceptions, 152–153
overview, 9, 143–147
pre- and postconditions, 151, 156
sample document, 581–583
setting priorities, 50
template for, 146, 150, 532
testing and, 144, 146–147, 343–344, 347
traps to avoid, 163–164
usage scenarios and, 149
use case diagrams, 148, 243, 395, 603
user stories and, 144–147, 152–153
users and actors, 147–148
validating, 160–161
user acceptance testing

user, defined, 603
user acceptance testing, 377–379
user classes, defined, 603. See also user analysis
user documentation, requirements and, 519–520
user goals. See user requirements
user interfaces
analyzing, good practices, 51
architecture diagrams, 445–446
custom input, 137
design of, requirements and, 375–377
dialog maps, 235–238
embedded projects, 446–447, 453
flow, 235
interface specification document, 447
mock-ups, 297–298
models for, 226
prototypes, 50
real-time projects, 446–447, 453
requirements analysis, 128
SRS and, 189–190, 196–197
SRS document, sample, 592–593
user interface analysis, 128
wireframe prototype, 299
user involvement in requirements, 101–116
user requirements. See also use cases; also user stories
business analytics projects, 431–432
business process automation requirements, 423–424
customer input, 136
defined, 7, 9, 603
elicitation, good practices, 48–49
packaged solution projects, 406–407
requirement levels and types, 7–13
requirements development, 16–17
stakeholder analysis, 28–29
techniques for identifying, overview, 143–144
user requirements document, 13, 400–401
user role. See actor
user stories. See also use cases; also user requirements
agile projects, 199–201, 386–389, 489
defined, 145, 603
enhancement and replacement projects, 395, 400–401
epics and, 388–389
features and, 388–389
overview, 143–147, 388–389
quality attributes, agile projects, 293–294
setting and changing priorities, 50, 314, 489
use cases and, 144–147, 152–153
user requirements, 9
user task models, 226
users. See also customers; also stakeholders
agile projects and, 115–116
classifying users, 102–104
conflicting requirements, resolution of, 116–117
customer comments, use in models, 223–224
enhancement and replacement projects, 395
importance of, 101–102
product champions, 109–114
SRS document, 193
user classes, identifying, 105–107
user observations, 125–126
user personas, 107–108
user representatives, 108–109

V

V model of software development, 330
validation, requirements. See also testing
acceptance criteria, 347–349
business analyst role, 64
defect checklist for requirements reviews, 338–339
defined, 331, 603
good practices, 44, 52–53
inspections, 332–338
outsourced projects, 420
overview of, 329–331
packaged solution projects, 408–409
peer reviews, 332–342
prototyping requirements, 342
requirements development, 15, 17, 45–47
requirements review tips and challenges, 339–342
requirements testing, 342–347
reviewing requirements, 332–342
risk factors, 545
testing requirements, 342–347
troubleshooting problems, 570
use cases, 160–161
verifiability requirements, 286–287
verifiable requirements, 205
verification, defined, 331, 603. See also validation
version control
  good practices, 53
  overview of, 460–462
  requirements management tools, 506–510
  requirements management, overview, 457–459
vertical prototype, 298, 603. See also prototypes
vision and scope document
  agile projects, 98–99
  business context, 90–92
  business requirements, 83–88
defined, 8, 81, 603
deliverables, 13
good practices, 51–52
overview, 81–83
sample document, 576–580
scope and limitations section, 88–90
template for, 81–83, 532
vision statement, 87–88, 577
vision, product, 78–79, 603
vision statement, 87–88, 577
visual representations. See models
voice of the user, 101, 108, 109
von Halle, Barbara, 177

W
walkthrough, 332–333
waterfall development, defined, 384, 603
waterfall development, limitations of, 384–385
Weinberg, Gerald, 105
Wiegers, Karl, 78, 225, 339, 366, 467
wireframe, 299, 603. See also prototypes
Withall, Stephen, 267, 358
work product, defined, 603
workshops
good practices, 49
requirements elicitation, 122–125
writing requirements documents, 203–220
writing style, requirements documentation, 207–211

Y
Young, Ralph, 61
About the authors

KARL WIEGERS is principal consultant with Process Impact, a software process consulting and education company in Portland, Oregon. His interests include requirements engineering, peer reviews, project management, and process improvement. Previously, he spent 18 years at Eastman Kodak Company as a photographic research scientist, software developer, software manager, and software process and quality improvement leader. Karl received a PhD degree in organic chemistry from the University of Illinois. When he's not on the computer, Karl enjoys wine tasting, playing guitar, writing and recording songs, and doing volunteer work.

Karl is the author of numerous books and articles on software development, chemistry, self-help, and military history. His books include the two previous editions of Software Requirements (Microsoft Press, 1999 and 2003), More About Software Requirements (Microsoft Press, 2006), Practical Project Initiation (Microsoft Press, 2007), Peer Reviews in Software (Addison-Wesley, 2002), and Creating a Software Engineering Culture (Dorset House Publishing, 1996). He is also the author of a memoir of life lessons, Pearls from Sand (Morgan James Publishing, 2011). Karl has served on the editorial board for IEEE Software magazine and as a contributing editor for Software Development magazine. He has delivered more than 300 seminars and training courses on software requirements. You can reach Karl at www.processimpact.com and www.karlwiegers.com. (Photo credit: Emily Down, Jama Software)

JOY BEATTY is a vice president at Seilevel, a professional services and training company in Austin, Texas, that helps redefine the way customers create software requirements. With 15 years of experience in business analysis, Joy evolves new methods and helps customers implement best practices that improve requirements elicitation and modeling. She assists Fortune 500 companies as they build business analysis centers of excellence. Joy has provided training to thousands of business analysts and is a Certified Business Analysis Professional (CBAP). Joy graduated from Purdue University with BS degrees in both computer science and mathematics. Joy's passions beyond requirements include rowing, swimming, and being outside with her family.

Joy is actively involved as a leader in the requirements community. She has worked with the International Institute of Business Analysis (IIBA) on A Guide to the Business Analysis Body of Knowledge (BABOK Guide). Additionally, she writes about requirements methodologies in journals, white papers, and blog posts and presents at requirements-related conferences. She also co-authored Visual Models for Software Requirements (Microsoft Press, 2012). Joy can be reached at www.seilevel.com and joy.beatty@seilevel.com.
Now that you’ve read the book...

Tell us what you think!

Was it useful?
Did it teach you what you wanted to learn?
Was there room for improvement?

Let us know at http://aka.ms/tellpress

Your feedback goes directly to the staff at Microsoft Press, and we read every one of your responses. Thanks in advance!