Microsoft

Software Requirements

Third Edition

Best practices

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

Microsoft

Software Requirements, Third Edition

Karl Wiegers and Joy Beatty

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2013942928
ISBN: 978-0-7356-7966-5

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at mspinput@microsoft.com. Please tell us what you think of this book at *http://www.microsoft.com/learning/booksurvey*.

Acquisitions Editor: Devon Musgrave
Developmental Editors: Devon Musgrave and Carol Dillingham
Project Editor: Carol Dillingham
Editorial Production: Christian Holdener, S4Carlisle Publishing Services
Copyeditor: Kathy Krause
Indexer: Maureen Johnson
Cover: Twist Creative • Seattle

[LSI]
[2013-09-06]

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

Contents

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:

https://www.microsoftpressstore.com/store/software-requirements-9780735679665

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:

https://www.microsoftpressstore.com/store/software-requirements-9780735679665

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 *http://www.MicrosoftPressStore.com/Support*.

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://www.MicrosoftPressStore.com/Support

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!

PART I

Software requirements: What, why, and who

CHAPTER 1

The essential software requirement

"Hello, Phil? This is Maria in Human Resources. We're having a problem with the personnel system you programmed for us. An employee just changed her name to Sparkle Starlight, and we can't get the system to accept the name change. Can you help?"

"She married some guy named Starlight?"

"No, she didn't get married, just changed her name," Maria replied. "That's the problem. It looks like we can change a name only if someone's marital status changes."

"Well, yeah, I never thought someone might just change her name. I don't remember you telling me about this possibility when we talked about the system," Phil said.

"I assumed you knew that people could legally change their name anytime they like," responded Maria. "We have to straighten this out by Friday or Sparkle won't be able to cash her paycheck. Can you fix the bug by then?"

"It's not a bug!" Phil retorted. "I never knew you needed this capability. I'm busy on the new performance evaluation system. I can probably fix it by the end of the month, but not by Friday. Sorry about that. Next time, tell me these things earlier and please write them down."

"What am I supposed to tell Sparkle?" demanded Maria. "She'll be upset if she can't cash her check."

"Hey, Maria, it's not my fault," Phil protested. "If you'd told me in the first place that you had to be able to change someone's name at any time, this wouldn't have happened. You can't blame me for not reading your mind."

Angry and resigned, Maria snapped, "Yeah, well, this is the kind of thing that makes me hate computers. Call me as soon as you get it fixed, will you?"

If you've ever been on the customer side of a conversation like this, you know how frustrating it is when a software system doesn't let you perform an essential task. You hate to be at the mercy of a developer who *might* get to your critical change request eventually. On the other hand, developers are frustrated to learn about functionality that a user expected only after they've implemented the system. It's also annoying for a developer to have his current project interrupted by a request to modify a system that does precisely what he was told it should do in the first place.

Many problems in the software world arise from shortcomings in the ways that people learn about, document, agree upon and modify the product's requirements. As with Phil and Maria, common problem areas are informal information gathering, implied functionality, miscommunicated assumptions, poorly specified requirements, and a casual change process. Various studies suggest that errors introduced during requirements activities account for 40 to 50 percent of all defects found in a software product (Davis 2005). Inadequate user input and shortcomings in specifying and managing customer requirements are major contributors to unsuccessful projects. Despite this evidence, many organizations still practice ineffective requirements methods.

Nowhere more than in the requirements do the interests of all the stakeholders in a project intersect. (See Chapter 2, "Requirements from the customer's perspective," for more about stakeholders.) These stakeholders include customers, users, business analysts, developers, and many others. Handled well, this intersection can lead to delighted customers and fulfilled developers. Handled poorly, it's the source of misunderstanding and friction that undermine the product's quality and business value. Because requirements are the foundation for both the software development and the project management activities, all stakeholders should commit to applying requirements practices that are known to yield superior-quality products.

But developing and managing requirements is hard! There are no simple shortcuts or magic solutions. On the plus side, so many organizations struggle with the same problems that you can look for techniques in common that apply to many different situations. This book describes dozens of such practices. The practices are presented as though you were building a brand-new system. However, most of them also apply to enhancement, replacement, and reengineering projects (see Chapter 21, "Enhancement and replacement projects") and to projects that incorporate commercial off-the-shelf (COTS) packaged solutions (see Chapter 22, "Packaged solution projects"). Project teams that build products incrementally by following an agile development process also need to understand the requirements that go into each increment (see Chapter 20, "Agile projects").

This chapter will help you to:

- Understand some key terms used in the software requirements domain.
- Distinguish *product* requirements from *project* requirements.
- Distinguish requirements *development* from requirements *management*.
- Be alert to several requirements-related problems that can arise.

Important We use the terms "system," "product," "application," and "solution" interchangeably in this book to refer to any kind of software or software-containing item that you build, whether for internal corporate use, for commercial sale, or on a contract basis.

Taking your requirements pulse

For a quick check of the current requirements practices in your organization, consider how many of the following conditions apply to your most recent project. If more than three or four of these items describe your experience, this book is for you:

- The project's business objectives, vision, and scope were never clearly defined.
- Customers were too busy to spend time working with analysts or developers on the requirements.
- Your team could not interact directly with representative users to understand their needs.
- Customers claimed that all requirements were critical, so they didn't prioritize them.
- Developers encountered ambiguities and missing information when coding, so they had to guess.
- Communications between developers and stakeholders focused on user interface displays or features, not on what users needed to accomplish with the software.
- Your customers never approved the requirements.
- Your customers approved the requirements for a release or iteration and then changed them continually.
- The project scope increased as requirements changes were accepted, but the schedule slipped because no additional resources were provided and no functionality was removed.
- Requested requirements changes got lost; no one knew the status of a particular change request.
- Customers requested certain functionality and developers built it, but no one ever uses it.
- At the end of the project, the specification was satisfied but the customer or the business objectives were not.

Software requirements defined

When a group of people begin discussing requirements, they often start with a terminology problem. Different observers might describe a single statement as being a user requirement, software requirement, business requirement, functional requirement, system requirement, product requirement, project requirement, user story, feature, or constraint. The names they use for various requirements deliverables also vary. A customer's definition of requirements might sound like a high-level product concept to the developer. The developer's notion of requirements might sound like a detailed user interface design to the user. This diversity of understanding leads to confusion and frustration.

Some interpretations of "requirement"

Many decades after the invention of computer programming, software practitioners still have raging debates about exactly what a "requirement" is. Rather than prolong those debates, in this book we simply present some definitions that we have found useful.

Consultant Brian Lawrence suggests that a *requirement* is "anything that drives design choices" (Lawrence 1997). This is not a bad colloquial definition, because many kinds of information fit in this category. And, after all, the whole point of developing requirements is to make appropriate design choices that will meet the customer's needs in the end. Another definition is that a requirement is a property that a product must have to provide value to a stakeholder. Also not bad, but not very precise. Our favorite definition, though, comes from Ian Sommerville and Pete Sawyer (1997):

> *Requirements are a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system.*

This definition acknowledges the diverse types of information that collectively are referred to as "the requirements." Requirements encompass both the user's view of the external system behavior and the developer's view of some internal characteristics. They include both the behavior of the system under specific conditions and those properties that make the system suitable—and maybe even enjoyable—for use by its intended operators.

Trap Don't assume that all your project stakeholders share a common notion of what requirements are. Establish definitions up front so that you're all talking about the same things.

The pure dictionary "requirement"

Software people do not use "requirement" in the same sense as a dictionary definition of the word: something demanded or obligatory, a need or necessity. People sometimes question whether they even need to prioritize requirements, because maybe a low-priority requirement won't ever be implemented. If it isn't truly needed, then it isn't a requirement, they claim. Perhaps, but then what would you call that piece of information? If you defer a requirement from today's project to an unspecified future release, is it still considered a requirement? Sure it is.

Software requirements include a time dimension. They could be present tense, describing the current system's capabilities. Or they could be for the near-term (high priority), mid-term (medium priority), or hypothetical (low priority) future. They could even be past tense, referring to needs that were once specified and then discarded. Don't waste time debating whether or not something is a requirement, even if you know you might never implement it for some good business reason. It is.

Levels and types of requirements

Because there are so many different types of requirements information, we need a consistent set of adjectives to modify the overloaded term "requirement." This section presents definitions we will use for some terms commonly encountered in the requirements domain (see Table 1-1).

TABLE 1-1 Some types of requirements information

Term	Definition
Business requirement	A high-level business objective of the organization that builds a product or of a customer who procures it.
Business rule	A policy, guideline, standard, or regulation that defines or constrains some aspect of the business. Not a software requirement in itself, but the origin of several types of software requirements.
Constraint	A restriction that is imposed on the choices available to the developer for the design and construction of a product.
External interface requirement	A description of a connection between a software system and a user, another software system, or a hardware device.
Feature	One or more logically related system capabilities that provide value to a user and are described by a set of functional requirements.
Functional requirement	A description of a behavior that a system will exhibit under specific conditions.
Nonfunctional requirement	A description of a property or characteristic that a system must exhibit or a constraint that it must respect.
Quality attribute	A kind of nonfunctional requirement that describes a service or performance characteristic of a product.
System requirement	A top-level requirement for a product that contains multiple subsystems, which could be all software or software and hardware.
User requirement	A goal or task that specific classes of users must be able to perform with a system, or a desired product attribute.

Software requirements include three distinct levels: business requirements, user requirements, and functional requirements. In addition, every system has an assortment of nonfunctional requirements. The model in Figure 1-1 illustrates a way to think about these diverse types of requirements. As statistician George E. P. Box famously said, "Essentially, all models are wrong, but some are useful" (Box and Draper 1987). That's certainly true of Figure 1-1. This model is not all-inclusive, but it does provide a helpful scheme for organizing the requirements knowledge you'll encounter.

The ovals in Figure 1-1 represent types of requirements information, and the rectangles indicate documents in which to store that information. The solid arrows indicate that a certain type of information typically is stored in the indicated document. (Business rules and system requirements are stored separately from software requirements, such as in a business rules catalog or a system requirements specification, respectively.) The dotted arrows indicate that one type of information is the origin of or influences another type of requirement. Data requirements are not shown explicitly in this diagram. Functions manipulate data, so data requirements can appear throughout the three levels. Chapter 7, "Requirements elicitation," contains many examples of these different types of requirements information.

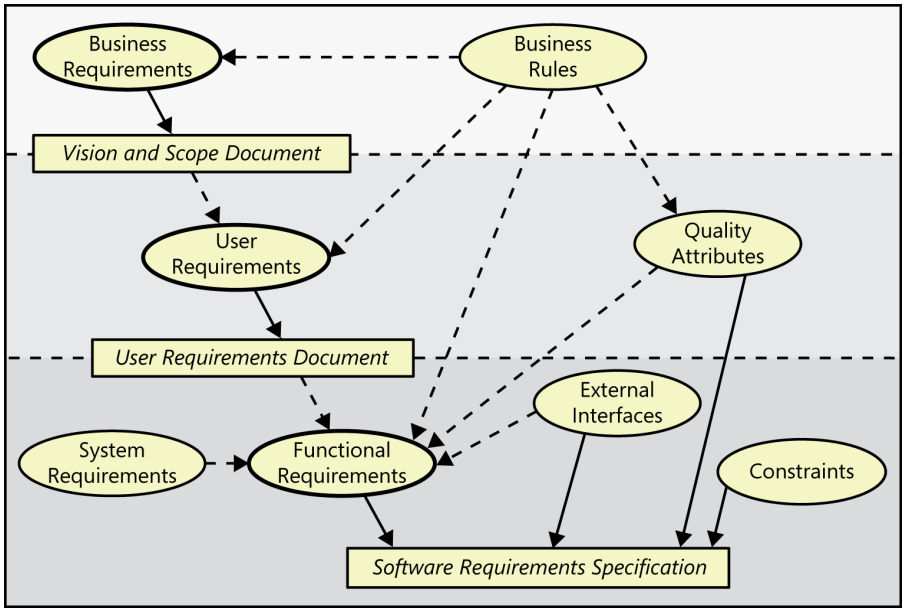

FIGURE 1-1 Relationships among several types of requirements information. Solid arrows mean "are stored in"; dotted arrows mean "are the origin of" or "influence."

Important Although we will refer to requirements "documents" throughout this book, as in Figure 1-1, those do not have to be traditional paper or electronic documents. Instead, think of them simply as containers in which to store requirements knowledge. Such a container could indeed be a traditional document, or it could be a spreadsheet, a set of diagrams, a database, a requirements management tool, or some combination of these. For convenience, we will use the term "document" to refer to any such container. We will provide templates that identify the types of information to consider storing in each such grouping, regardless of what form you store it in. What you call each deliverable is less important than having your organization agree on their names, what kinds of information go into each, and how that information is organized.

Business requirements describe *why* the organization is implementing the system—the business benefits the organization hopes to achieve. The focus is on the business objectives of the organization or the customer who requests the system. Suppose an airline wants to reduce airport counter staff costs by 25 percent. This goal might lead to the idea of building a kiosk that passengers can use to check in for their flights at the airport. Business requirements typically come from the funding sponsor for a project, the acquiring customer, the manager of the actual users, the marketing department, or a product visionary. We like to record the business requirements in a *vision and scope document.* Other strategic guiding documents sometimes used for this purpose include a project charter, business case, and market (or marketing) requirements document. Specifying business requirements is the subject of Chapter 5, "Establishing the business requirements." For the purposes of this book, we are assuming that the business need or market opportunity has already been identified.

User requirements describe goals or tasks the users must be able to perform with the product that will provide value to someone. The domain of user requirements also includes descriptions of product attributes or characteristics that are important to user satisfaction. Ways to represent user requirements include use cases (Kulak and Guiney 2004), user stories (Cohn 2004), and event-response tables. Ideally, actual user representatives will provide this information. User requirements describe *what* the user will be able to do with the system. An example of a use case is "Check in for a flight" using an airline's website or a kiosk at the airport. Written as a user story, the same user requirement might read: "As a passenger, I want to check in for a flight so I can board my airplane." It's important to remember that most projects have multiple user classes, as well as other stakeholders whose needs also must be elicited. Chapter 8, "Understanding user requirements," addresses this level of the model. Some people use the broader term "stakeholder requirements," to acknowledge the reality that various stakeholders other than direct users will provide requirements. That is certainly true, but we focus the attention at this level on understanding what actual users need to achieve with the help of the product.

Functional requirements specify the behaviors the product will exhibit under specific conditions. They describe *what* the developers must implement to enable users to accomplish their tasks (user requirements), thereby satisfying the business requirements. This alignment among the three levels of requirements is essential for project success. Functional requirements often are written in the form of the traditional "shall" statements: "The Passenger shall be able to print boarding passes for all flight segments for which he has checked in" or "If the Passenger's profile does not indicate a seating preference, the reservation system shall assign a seat."

The business analyst (BA)[1] documents functional requirements in a *software requirements specification* (SRS), which describes as fully as necessary the expected behavior of the software system. The SRS is used in development, testing, quality assurance, project management, and related project functions. People call this deliverable by many different names, including business requirements document, functional spec, requirements document, and others. An SRS could be a report generated from information stored in a requirements management tool. Because it is an industry-standard term, we will use "SRS" consistently throughout this book (ISO/IEC/IEEE 2011). See Chapter 10, "Documenting the requirements," for more information about the SRS.

System requirements describe the requirements for a product that is composed of multiple components or subsystems (ISO/IEC/IEEE 2011). A "system" in this sense is not just any information system. A system can be all software or it can include both software and hardware subsystems. People and processes are part of a system, too, so certain system functions might be allocated to human beings. Some people use the term "system requirements" to mean the detailed requirements for a software system, but that's not how we use the term in this book.

A good example of a "system" is the cashier's workstation in a supermarket. There's a bar code scanner integrated with a scale, as well as a hand-held bar code scanner. The cashier has a keyboard, a display, and a cash drawer. You'll see a card reader and PIN pad for your loyalty card and credit or debit card, and perhaps a change dispenser. You might see up to three printers for your purchase

[1] "Business analyst" refers to the project role that has primary responsibility for leading requirements-related activities on a project. The BA role also goes by many other names. See Chapter 4, "The business analyst," for more about the business analyst role.

receipt, credit card receipt, and coupons you don't care about. These hardware devices are all interacting under software control. The requirements for the system or product as a whole, then, lead the business analyst to derive specific functionality that must be allocated to one or another of those component subsystems, as well as demanding an understanding of the interfaces between them.

Business rules include corporate policies, government regulations, industry standards, and computational algorithms. As you'll see in Chapter 9, "Playing by the rules," business rules are not themselves software requirements because they have an existence beyond the boundaries of any specific software application. However, they often dictate that the system must contain functionality to comply with the pertinent rules. Sometimes, as with corporate security policies, business rules are the origin of specific quality attributes that are then implemented in functionality. Therefore, you can trace the genesis of certain functional requirements back to a particular business rule.

In addition to functional requirements, the SRS contains an assortment of nonfunctional requirements. *Quality attributes* are also known as quality factors, quality of service requirements, constraints, and the "–ilities." They describe the product's characteristics in various dimensions that are important either to users or to developers and maintainers, such as performance, safety, availability, and portability. Other classes of nonfunctional requirements describe *external interfaces* between the system and the outside world. These include connections to other software systems, hardware components, and users, as well as communication interfaces. Design and implementation *constraints* impose restrictions on the options available to the developer during construction of the product.

If they're nonfunctional, then what are they?

For many years, the requirements for a software product have been classified broadly as either functional or nonfunctional. The functional requirements are evident: they describe the observable behavior of the system under various conditions. However, many people dislike the term "nonfunctional." That adjective says what the requirements are *not*, but it doesn't say what they *are*. We are sympathetic to the problem, but we lack a perfect solution.

Other-than-functional requirements might specify not *what* the system does, but rather *how well* it does those things. They could describe important characteristics or properties of the system. These include the system's availability, usability, security, performance, and many other characteristics, as addressed in Chapter 14, "Beyond functionality." Some people consider nonfunctional requirements to be synonymous with quality attributes, but that is overly restrictive. For example, design and implementation constraints are also nonfunctional requirements, as are external interface requirements.

Still other nonfunctional requirements address the environment in which the system operates, such as platform, portability, compatibility, and constraints. Many products are also affected by compliance, regulatory, and certification requirements. There could be localization requirements for products that must take into account the cultures, languages, laws, currencies, terminology, spelling, and other characteristics of users. Though such requirements are specified in nonfunctional terms, the business analyst typically will derive numerous bits of functionality to ensure that the system possesses all the desired behaviors and properties.

In this book, we are sticking with the term "nonfunctional requirements," despite its limitations, for the lack of a suitably inclusive alternative. Rather than worry about precisely what you call these sorts of information, just make sure that they are part of your requirements elicitation and analysis activities. You can deliver a product that has all the desired functionality but that users hate because it doesn't match their (often unstated) quality expectations.

A *feature* consists of one or more logically related system capabilities that provide value to a user and are described by a set of functional requirements. A customer's list of desired product features is not equivalent to a description of the user's task-related needs. Web browser bookmarks, spelling checkers, the ability to define a custom workout program for a piece of exercise equipment, and automatic virus signature updating in an anti-malware product are examples of features. A feature can encompass multiple user requirements, each of which implies that certain functional requirements must be implemented to allow the user to perform the task described by each user requirement. Figure 1-2 illustrates a *feature tree*, an analysis model that shows how a feature can be hierarchically decomposed into a set of smaller features, which relate to specific user requirements and lead to specifying sets of functional requirements (Beatty and Chen 2012).

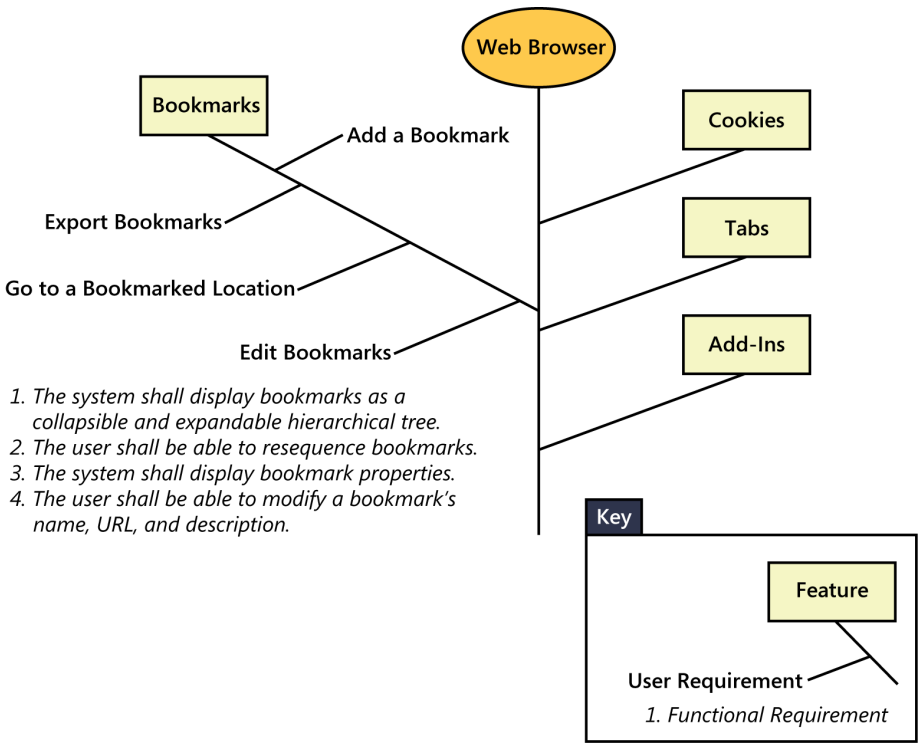

FIGURE 1-2 Relationships among features, user requirements, and functional requirements.

To illustrate some of these various kinds of requirements, consider a project to develop the next version of a text editor program. A business requirement might be "Increase non-US sales by 25 percent within 6 months." Marketing realizes that the competitive products only have English-language spelling checkers, so they decide that the new version will include a multilanguage spelling checker feature. Corresponding user requirements might include tasks such as "Select language for spelling checker," "Find spelling errors," and "Add a word to a dictionary." The spelling checker has many individual functional requirements, which deal with operations such as highlighting misspelled words, autocorrect, displaying suggested replacements, and globally replacing misspelled words with corrected words. Usability requirements specify how the software is to be localized for use with specific languages and character sets.

Working with the three levels

Figure 1-3 illustrates how various stakeholders might participate in eliciting the three levels of requirements. Different organizations use a variety of names for the roles involved in these activities; think about who performs these activities in your organization. The role names often differ depending on whether the developing organization is an internal corporate entity or a company building software for commercial use.

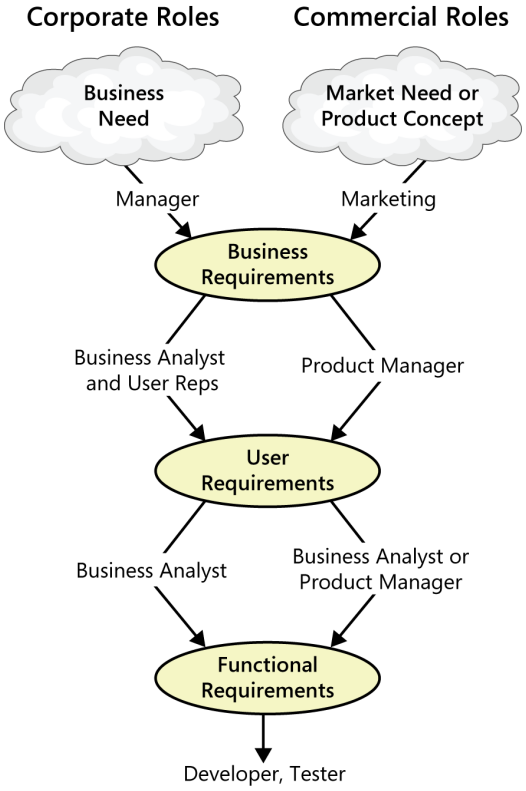

FIGURE 1-3 An example of how different stakeholders participate in requirements development.

Based on an identified business need, a market need, or an exciting new product concept, managers or marketing define the business requirements for software that will help their company operate more efficiently (for information systems) or compete successfully in the marketplace (for commercial products). In the corporate environment, a business analyst then typically works with user representatives to identify user requirements. Companies developing commercial products often identify a product manager to determine what features to include in the new product. Each user requirement and feature must align with accomplishing the business requirements. From the user requirements, the BA or product manager derives the functionality that will let users achieve their goals. Developers use the functional and nonfunctional requirements to design solutions that implement the necessary functionality, within the limits that the constraints impose. Testers determine how to verify whether the requirements were correctly implemented.

It's important to recognize the value of recording vital requirements information in a shareable form, rather than treating it as oral tradition around the project campfire. I was on a project once that had experienced a rotating cast of development teams. The primary customer was sick to tears of having each new team come along and say, "We have to talk about your requirements." His reaction to our request was, "I already gave your predecessors my requirements. Now build me a system!" Unfortunately, no one had ever documented any requirements, so every new team had to start from scratch. To proclaim that you "have the requirements" is delusional if all you really have is a pile of email and voice mail messages, sticky notes, meeting minutes, and vaguely recollected hallway conversations. The BA must practice good judgment to determine just how comprehensive to make the requirements documentation on a given project.

Figure 1-1, shown earlier in this chapter, identified three major requirements deliverables: a vision and scope document, a user requirements document, and a software requirements specification. You do not necessarily need to create three discrete requirements deliverables on each project. It often makes sense to combine some of this information, particularly on small projects. However, recognize that these three deliverables contain different information, developed at different points in the project, possibly by different people, with different purposes and target audiences.

The model in Figure 1-1 showed a simple top-down flow of requirements information. In reality, you should expect cycles and iteration among the business, user, and functional requirements. Whenever someone proposes a new feature, user requirement, or bit of functionality, the analyst must ask, "Is this in scope?" If the answer is "yes," the requirement belongs in the specification. If the answer is "no," it does not, at least not for the forthcoming release or iteration. The third possible answer is "no, but it supports the business objectives, so it ought to be." In that case, whoever controls the project scope—the project sponsor, project manager, or product owner—must decide whether to increase the current project's or iteration's scope to accommodate the new requirement. This is a business decision that has implications for the project's schedule and budget and might demand trade-offs with other capabilities. An effective change process that includes impact analysis ensures that the right people make informed business decisions about which changes to accept and that the associated costs in time, resources, or feature trade-offs are addressed.

Product vs. project requirements

So far we have been discussing requirements that describe properties of a software system to be built. Let's call those *product* requirements. Projects certainly do have other expectations and deliverables that are not a part of the software the team implements, but that are necessary to the successful completion of the project as a whole. These are *project* requirements but not *product* requirements. An SRS houses the product requirements, but it should not include design or implementation details (other than known constraints), project plans, test plans, or similar information. Separate out such items so that requirements development activities can focus on understanding what the team intends to build. Project requirements include:

- Physical resources the development team needs, such as workstations, special hardware devices, testing labs, testing tools and equipment, team rooms, and videoconferencing equipment.
- Staff training needs.
- User documentation, including training materials, tutorials, reference manuals, and release notes.
- Support documentation, such as help desk resources and field maintenance and service information for hardware devices.
- Infrastructure changes needed in the operating environment.
- Requirements and procedures for releasing the product, installing it in the operating environment, configuring it, and testing the installation.
- Requirements and procedures for transitioning from an old system to a new one, such as data migration and conversion requirements, security setup, production cutover, and training to close skills gaps; these are sometimes called *transition requirements* (IIBA 2009).
- Product certification and compliance requirements.
- Revised policies, processes, organizational structures, and similar documents.
- Sourcing, acquisition, and licensing of third-party software and hardware components.
- Beta testing, manufacturing, packaging, marketing, and distribution requirements.
- Customer service-level agreements.
- Requirements for obtaining legal protection (patents, trademarks, or copyrights) for intellectual property related to the software.

This book does not address these sorts of project requirements further. That doesn't mean that they aren't important, just that they are out of scope for our focus on software product requirements development and management. Identifying these project requirements is a shared responsibility of the BA and the project manager. They often come up while eliciting product requirements. Project requirements information is best stored in the project management plan, which should itemize all expected project activities and deliverables.

Particularly for business applications, people sometimes refer to a "solution" as encompassing both the product requirements (which are principally the responsibility of the business analyst) and the project requirements (which are principally the responsibility of the project manager). They might use the term "solution scope" to refer to "everything that has to be done to complete the project successfully." In this book, though, we are focusing on product requirements, whether your ultimate deliverable is a commercial software product, a hardware device with embedded software, a corporate information system, contracted government software, or anything else.

Requirements development and management

Confusion about requirements terminology extends even to what to call the whole discipline. Some authors call the entire domain *requirements engineering* (our preference). Others refer to it all as *requirements management*. Still others refer to these activities as a subset of the broad domain of business analysis.

We find it useful to split requirements engineering into *requirements development* (addressed in Part II of this book) and *requirements management* (addressed in Part IV), as shown in Figure 1-4. Regardless of what development life cycle your project is following—be it pure waterfall, phased, iterative, incremental, agile, or some hybrid—these are the things you need to do regarding requirements. Depending on the life cycle, you will perform these activities at different times in the project and to varying degrees of depth or detail.

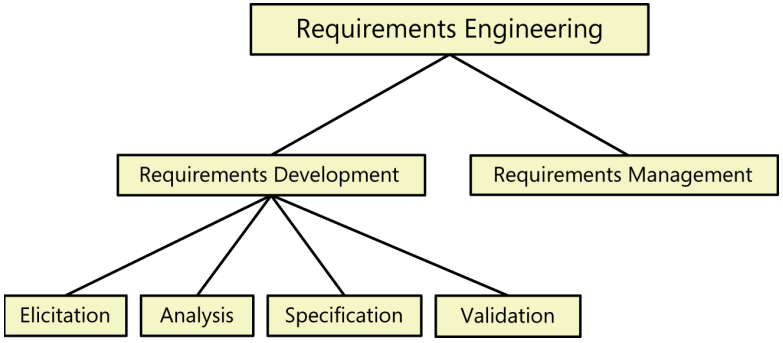

FIGURE 1-4 Subdisciplines of software requirements engineering.

Requirements development

As Figure 1-4 shows, we subdivide requirements development into *elicitation*, *analysis*, *specification*, and *validation* (Abran et al. 2004). These subdisciplines encompass all the activities involved with exploring, evaluating, documenting, and confirming the requirements for a product. Following are the essential actions in each subdiscipline.

Elicitation

Elicitation encompasses all of the activities involved with discovering requirements, such as interviews, workshops, document analysis, prototyping, and others. The key actions are:

- Identifying the product's expected user classes and other stakeholders.
- Understanding user tasks and goals and the business objectives with which those tasks align.
- Learning about the environment in which the new product will be used.
- Working with individuals who represent each user class to understand their functionality needs and their quality expectations.

Usage-centric or product-centric?

Requirements elicitation typically takes either a usage-centric or a product-centric approach, although other strategies also are possible. The usage-centric strategy emphasizes understanding and exploring user goals to derive the necessary system functionality. The product-centric approach focuses on defining features that you expect will lead to marketplace or business success. A risk with product-centric strategies is that you might implement features that don't get used much, even if they seemed like a good idea at the time. We recommend understanding business objectives and user goals first, then using that insight to determine the appropriate product features and characteristics.

Analysis

Analyzing requirements involves reaching a richer and more precise understanding of each requirement and representing sets of requirements in multiple ways. Following are the principal activities:

- Analyzing the information received from users to distinguish their task goals from functional requirements, quality expectations, business rules, suggested solutions, and other information
- Decomposing high-level requirements into an appropriate level of detail
- Deriving functional requirements from other requirements information
- Understanding the relative importance of quality attributes
- Allocating requirements to software components defined in the system architecture
- Negotiating implementation priorities
- Identifying gaps in requirements or unnecessary requirements as they relate to the defined scope

Specification

Requirements specification involves representing and storing the collected requirements knowledge in a persistent and well-organized fashion. The principal activity is:

- Translating the collected user needs into written requirements and diagrams suitable for comprehension, review, and use by their intended audiences.

Validation

Requirements validation confirms that you have the correct set of requirements information that will enable developers to build a solution that satisfies the business objectives. The central activities are:

- Reviewing the documented requirements to correct any problems before the development group accepts them.
- Developing acceptance tests and criteria to confirm that a product based on the requirements would meet customer needs and achieve the business objectives.

Iteration is a key to requirements development success. Plan for multiple cycles of exploring requirements, progressively refining high-level requirements into more precision and detail, and confirming correctness with users. This takes time and it can be frustrating. Nonetheless, it's an intrinsic aspect of dealing with the fuzzy uncertainty of defining a new software system.

Important You're never going to get perfect requirements. From a practical point of view, the goal of requirements development is to accumulate a shared understanding of requirements that is *good enough* to allow construction of the next portion of the product—be that 1 percent or 100 percent of the entire product—to proceed at an acceptable level of risk. The major risk is that of having to do excessive unplanned rework because the team didn't sufficiently understand the requirements for the next chunk of work before starting design and construction.

Requirements management

Requirements management activities include the following:

- Defining the requirements baseline, a snapshot in time that represents an agreed-upon, reviewed, and approved set of functional and nonfunctional requirements, often for a specific product release or development iteration
- Evaluating the impact of proposed requirements changes and incorporating approved changes into the project in a controlled way
- Keeping project plans current with the requirements as they evolve
- Negotiating new commitments based on the estimated impact of requirements changes

- Defining the relationships and dependencies that exist between requirements
- Tracing individual requirements to their corresponding designs, source code, and tests
- Tracking requirements status and change activity throughout the project

The object of requirements management is not to stifle change or to make it difficult. It is to anticipate and accommodate the very real changes that you can always expect so as to minimize their disruptive impact on the project.

Figure 1-5 provides another view of the boundary between requirements development and requirements management. This book describes dozens of specific practices for performing requirements elicitation, analysis, specification, validation, and management.

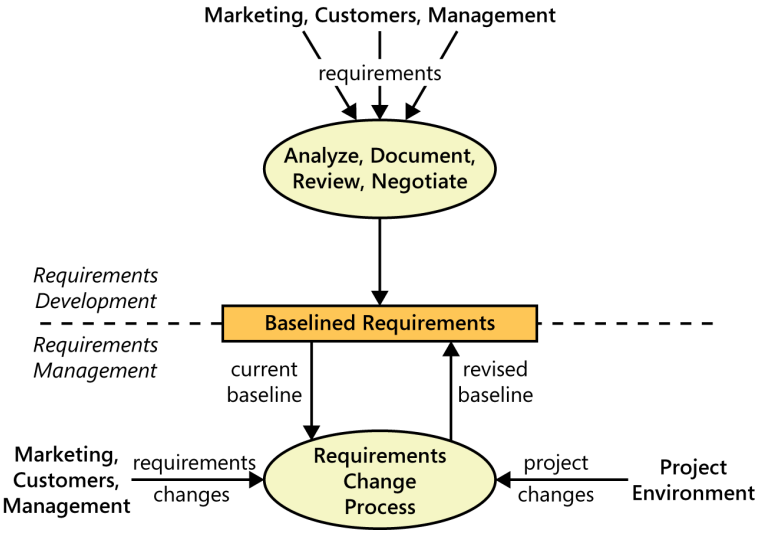

FIGURE 1-5 The boundary between requirements development and requirements management.

Every project has requirements

Frederick Brooks eloquently stated the critical role of requirements to a software project in his classic 1987 essay, "No Silver Bullet: Essence and Accidents of Software Engineering":

> *The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.*

Every software-containing system has stakeholders who rely on it. The time spent understanding their needs is a high-leverage investment in project success. If a project team does not have written representations of requirements that the stakeholders agree to, how can developers be sure to satisfy those stakeholders?

Often, it's impossible—or unnecessary—to fully specify the functional requirements before commencing design and implementation. In those cases, you can take an iterative or incremental approach, implementing one portion of the requirements at a time and obtaining customer feedback before moving on to the next cycle. This is the essence of agile development, learning just enough about requirements to do thoughtful prioritization and release planning so the team can begin delivering valuable software as quickly as possible. This isn't an excuse to write code before contemplating requirements for that next increment, though. Iterating on code is more expensive than iterating on concepts.

People sometimes balk at spending the time that it takes to write software requirements. But writing the requirements isn't the hard part. The hard part is *determining* the requirements. Writing requirements is a matter of clarifying, elaborating, and recording what you've learned. A solid understanding of a product's requirements ensures that your team works on the right problem and devises the best solution to that problem. Without knowing the requirements, you can't tell when the project is done, determine whether it has met its goals, or make trade-off decisions when scope adjustments are necessary. Instead of balking at spending time on requirements, people should instead balk at the money wasted when the project doesn't pay enough attention to requirements.

When bad requirements happen to good people

The major consequence of requirements problems is rework—doing again something that you thought was already done—late in development or after release. Rework often consumes 30 to 50 percent of your total development cost (Shull, et al. 2002; GAO 2004), and requirements errors can account for 70 to 85 percent of the rework cost (Leffingwell 1997). Some rework does add value and improves the product, but excessive rework is wasteful and frustrating. Imagine how different your life would be if you could cut the rework effort in half! Your team members could build better products faster and perhaps even go home on time. Creating better requirements is an investment, not just a cost.

It can cost far more to correct a defect that's found late in the project than to fix it shortly after its creation. Suppose it costs $1 (on a relative scale) to find and fix a requirement defect while you're still working on the requirements. If you discover that error during design instead, you have to pay the $1 to fix the requirement error, plus another $2 or $3 to redo the design that was based on the incorrect requirement. Suppose, though, that no one finds the error until a user calls with a problem. Depending on the type of system, the cost to correct a requirement defect found in operation can be $100 or more on this relative scale (Boehm 1981; Grady 1999; Haskins 2004). One of my consulting clients determined that they spent an average of $200 of labor effort to find and fix a defect in their information systems using the quality technique of software inspection, a type of peer review

(Wiegers 2002). In contrast, they spent an average of *$4,200* to fix a single defect reported by the user, an amplification factor of 21. Preventing requirements errors and catching them early clearly has a huge leveraging effect on reducing rework.

Shortcomings in requirements practices pose many risks to project success, where *success* means delivering a product that satisfies the user's functional and quality expectations at the agreed-upon cost and schedule. Chapter 32, "Software requirements and risk management," describes how to manage such risks to prevent them from derailing your project. Some of the most common requirements risks are described in the following sections.

Insufficient user involvement

Customers often don't understand why it is so essential to work hard on eliciting requirements and assuring their quality. Developers might not emphasize user involvement, perhaps because they think they already understand what the users need. In some cases it's difficult to gain access to people who will actually use the product, and user surrogates don't always understand what users really need. Insufficient user involvement leads to late-breaking requirements that generate rework and delay completion.

Another risk of insufficient user involvement, particularly when reviewing and validating the requirements, is that the business analyst might not understand and properly record the true business or customer needs. Sometimes a BA goes down the path of specifying what appears to be the "perfect" requirements, and developers implement them, but then no one uses the solution because the business problem was misunderstood. Ongoing conversations with users can help mitigate this risk, but if users don't review the requirements carefully enough, you can still have problems.

Inaccurate planning

"Here's my idea for a new product; when will you be done?" No one should answer this question until more is known about the problem being discussed. Vague, poorly understood requirements lead to overly optimistic estimates, which come back to haunt you when the inevitable overruns occur. An estimator's quick guess sounds a lot like a commitment to the listener. The top contributors to poor software cost estimation are frequent requirements changes, missing requirements, insufficient communication with users, poor specification of requirements, and insufficient requirements analysis (Davis 1995). Estimating project effort and duration based on requirements means that you need to know something about the size of your requirements and the development team's productivity. See Chapter 5 of *More about Software Requirements* (Wiegers 2006) for more about estimation based on requirements.

Creeping user requirements

As requirements evolve during development, projects often exceed their planned schedules and budgets (which are nearly always too optimistic anyway). To manage scope creep, begin with a clear statement of the project's business objectives, strategic vision, scope, limitations, and success criteria. Evaluate all proposed new features or requirements changes against this reference. Requirements *will*

change and grow. The project manager should build contingency buffers into schedules so the first new requirement that comes along doesn't derail the schedule (Wiegers 2007). Agile projects take the approach of adjusting the scope for a certain iteration to fit into a defined budget and duration for the iteration. As new requirements come along, they are placed into the backlog of pending work and allocated to future iterations based on priority. Change might be critical to success, but change always has a price.

Ambiguous requirements

One symptom of ambiguity in requirements is that a reader can interpret a requirement statement in several ways (Lawrence 1996). Another sign is that multiple readers of a requirement arrive at different understandings of what it means. Chapter 11, "Writing excellent requirements," lists many words and phrases that contribute to ambiguity by placing the burden of interpretation on the reader.

Ambiguity leads to different expectations on the part of various stakeholders. Some of them are then surprised at whatever is delivered. Ambiguous requirements cause wasted time when developers implement a solution for the wrong problem. Testers who expect the product to behave differently from what the developers built waste time resolving the differences.

One way to ferret out ambiguity is to have people who represent different perspectives inspect the requirements (Wiegers 2002). As described in Chapter 17, "Validating the requirements," informal peer reviews in which reviewers simply read the requirements on their own often don't reveal ambiguities. If different reviewers interpret a requirement in different ways but it makes sense to each of them, they won't find the ambiguity. Collaborative elicitation and validation encourages stakeholders to discuss and clarify requirements as a group in a workshop setting. Writing tests against the requirements and building prototypes are other ways to discover ambiguities.

Gold plating

Gold plating takes place when a developer adds functionality that wasn't in the requirements specification (or was deemed out of scope) but which the developer believes "the users are just going to love." If users don't care about this functionality, the time spent implementing it is wasted. Rather than simply inserting new features, developers and BAs should present stakeholders with creative ideas for their consideration. Developers should strive for leanness and simplicity, not going beyond what stakeholders request without their approval.

Customers sometimes request certain features or elaborate user interfaces that look attractive but add little value to the product. Everything you build costs time and money, so you need to maximize the delivered value. To reduce the threat of gold plating, trace each bit of functionality back to its origin and its business justification so everyone knows why it's included. Make sure that what you are specifying and developing lies within the project's scope.

Overlooked stakeholders

Most products have several groups of users who might use different subsets of features, have different frequencies of use, or have varying levels of experience. If you don't identify the important user classes for your product early on, some user needs won't be met. After identifying all user classes, make sure that each has a voice, as discussed in Chapter 6, "Finding the voice of the user." Besides obvious users, think about maintenance and field support staff who have their own requirements, both functional and nonfunctional. People who have to convert data from a legacy system will have transition requirements that don't affect the ultimate product software but that certainly influence solution success. You might have stakeholders who don't even know the project exists, such as government agencies that mandate standards that affect your system, yet you need to know about them and their influence on the project.

Benefits from a high-quality requirements process

Some people mistakenly believe that time spent discussing requirements simply delays delivery by the same duration. This assumes that there's no return on investment from requirements activities. In actuality, investing in good requirements will virtually always return more than it costs.

Sound requirements processes emphasize a collaborative approach to product development that involves stakeholders in a partnership throughout the project. Eliciting requirements lets the development team better understand its user community or market, a critical success factor. Emphasizing user tasks instead of superficially attractive features helps the team avoid writing code that no one will ever execute. Customer involvement reduces the expectation gap between what the customer really needs and what the developer delivers. You're going to get the customer input eventually; it's far cheaper to reach this understanding before you build the product than after delivery. Chapter 2 addresses the nature of the customer-development partnership.

Explicitly allocating system requirements to various software, hardware, and human subsystems emphasizes a systems approach to product engineering. An effective change control process will minimize the adverse impact of requirements changes. Documented and clear requirements greatly facilitate system testing. All of these increase your chances of delivering high-quality products that satisfy all stakeholders.

No one can promise a specific return on investment from using sound requirements practices. You can go through an analytical thought process to imagine how better requirements could help your teams, though (Wiegers 2006). The cost of better requirements includes developing new procedures and document templates, training the team, and buying tools. Your greatest investment is the time your project teams actually spend on requirements engineering tasks. The potential payoff includes:

- Fewer defects in requirements and in the delivered product.
- Reduced development rework.
- Faster development and delivery.

- Fewer unnecessary and unused features.
- Lower enhancement costs.
- Fewer miscommunications.
- Reduced scope creep.
- Reduced project chaos.
- Higher customer and team member satisfaction.
- Products that do what they're supposed to do.

Even if you can't quantify all of these benefits, they are real.

Next steps

- Write down requirements-related problems that you have encountered on your current or previous project. Identify each as a requirements development or requirements management problem. Describe the root cause of each problem and its impact on the project.
- Facilitate a discussion with your team members and other stakeholders regarding requirements-related problems from your current or previous projects, their impacts, and their root causes. Pool your ideas about changes in your current requirements practices that could address these problems. The troubleshooting guide in Appendix B might be helpful.
- Map the requirements terminology and deliverables used in your organization to that shown in this chapter to see if you're covering all the categories recommended here.
- Perform a simple assessment on just a few pages of one of your requirements documents to see where your team might have some clear improvement areas. It might be most useful to have an objective outsider perform this assessment.
- Arrange a training class on software requirements for your entire project team. Invite key customers, marketing staff, managers, developers, testers, and other stakeholders to participate. Training gives project participants a common vocabulary. It provides a shared appreciation of effective techniques and behaviors so that all team members can collaborate more effectively on their mutual challenges.

CHAPTER 2

Requirements from the customer's perspective

Gerhard, a senior manager at Contoso Pharmaceuticals, was meeting with Cynthia, the manager of Contoso's IT department. "We need to build a chemical tracking information system," Gerhard began. "The system should keep track of all the chemical containers we already have in the stockroom and in laboratories. That way, the chemists can get some chemicals from someone down the hall instead of always buying a new container. This should save us a lot of money. Also, the Health and Safety Department needs to generate government reports on chemical usage and disposal with a lot less work than it takes them today. Can you build this system in time for the compliance audit in five months?"

"I see why this project is important, Gerhard," said Cynthia. "But before I can commit to a schedule, we'll need to understand the requirements for the chemical tracking system."

Gerhard was confused. "What do you mean? I just told you my requirements."

"Actually, you described some general business objectives for the project," Cynthia explained. "That doesn't give me enough information to know what software to build or how long it might take. I'd like to have one of our business analysts work with some users to understand their needs for the system."

"The chemists are busy people," Gerhard protested. "They don't have time to nail down every detail before you can start programming. Can't your people figure out what to build?"

Cynthia replied, "If we just make our best guess at what the users need to do with the system, we can't do a good job. We're software developers, not chemists. I've learned that if we don't take the time to understand the problem, nobody is happy with the results."

"We don't have time for all that," Gerhard insisted. "I gave you my requirements. Now just build the system, please. Keep me posted on your progress."

Conversations like this take place regularly in the software world. Customers who request a new system often don't understand the importance of obtaining input from actual users of the proposed system as well as other stakeholders. Marketers with a great product concept believe that they can adequately represent the interests of prospective buyers. However, there's no substitute for eliciting requirements directly from people who will actually use the product. Some agile development methods recommend that an on-site customer representative, sometimes called a product owner, work closely with the development team. As one book about agile development said, "The project is *steered* to success by the customer and programmers working in concert" (Jeffries, Anderson, and Hendrickson 2001).

Part of the requirements problem results from confusion over the different levels of requirements described in Chapter 1, "The essential software requirement": business, user, and functional. Gerhard stated some business objectives, benefits that he expects Contoso to enjoy with the help of the new chemical tracking system. Business objectives are a core element of the business requirements. However, Gerhard can't entirely describe the user requirements because he's not an intended user of the system. Users, in turn, can describe tasks they must be able to perform with the system, but they can't state all the functional requirements that developers must implement to let them accomplish those tasks. Business analysts need to collaborate with users to reach that deeper understanding.

This chapter addresses the customer-development relationship that is so critical to software project success. We propose a Requirements Bill of Rights for Software Customers and a corresponding Requirements Bill of Responsibilities for Software Customers. These lists underscore the importance of customer—and specifically end user—involvement in requirements development. This chapter also discusses the critical issue of reaching agreement on a set of requirements planned for a specific release or development iteration. Chapter 6, "Finding the voice of the user," describes various types of customers and users and ways to engage appropriate user representatives in requirements elicitation.

Deliverable: Rejected

I heard a sad story when I visited a corporate IT department once. The developers had recently built a new information system for use within the company. They had obtained negligible user input from the beginning. The day the developers proudly unveiled their new system, the users rejected it as completely unacceptable. This came as a shock because the developers had worked hard to satisfy what they perceived to be the users' needs. So what did they do then? They fixed it. Companies always fix the system when they get the requirements wrong, yet it always costs much more than if they had engaged user representatives from the outset.

The developers hadn't planned to spend time fixing the flawed information system, of course, so the next project in the team's queue had to wait. This is a lose-lose-lose situation. The developers were chagrined, the users were unhappy because their new system wasn't available when they expected it, and the executives were upset over a lot of wasted money and the opportunity costs of delaying other projects. Extensive and ongoing customer engagement from the start could have prevented this unfortunate—but not uncommon—project outcome.

The expectation gap

Without adequate customer involvement, the inescapable outcome at the end of the project is an expectation gap, a gulf between what customers really need and what developers deliver based on what they heard at the beginning of the project (Wiegers 1996). This is shown as the dashed lines in Figure 2-1. As with the previous story, the expectation gap comes as a rude surprise to all stakeholders. In our experience, software surprises are never good news. Requirements also get out of date because of changes that occur in the business, so ongoing interactions with customers are vital.

The best way to minimize the expectation gap is to arrange frequent contact points with suitable customer representatives. These contact points can take the form of interviews, conversations, requirements reviews, user interface design walkthroughs, prototype evaluations, and—with agile development—user feedback on small increments of executable software. Each contact point affords an opportunity to close the expectation gap: what the developer builds is more closely aligned with what the customer needs.

Of course, the gap will begin to grow again immediately as development proceeds after each contact. The more frequent the contact points are, the easier it is to stay on track. As the progressively shrinking small gray triangles in Figure 2-1 illustrate, a series of such contact points will lead to a far smaller expectation gap at the end of the project and a solution that is much closer to the actual customer needs. This is why one of the guiding principles of agile development is to have ongoing conversations between developers and customers. That's an excellent principle for any project.

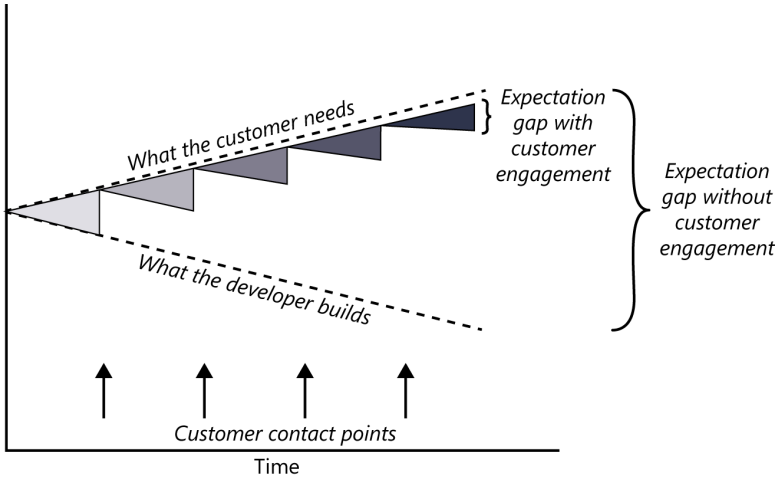

FIGURE 2-1 Frequent customer engagement reduces the expectation gap.

Who is the customer?

Before we can talk about customers, we need to discuss stakeholders. A *stakeholder* is a person, group, or organization that is actively involved in a project, is affected by its process or outcome, or can influence its process or outcome. Stakeholders can be internal or external to the project team and to the developing organization. Figure 2-2 identifies many of the potential stakeholders in these categories. Not all of these will apply to every project or situation, of course.

Stakeholder analysis is an important part of requirements development (Smith 2000; Wiegers 2007; IIBA 2009). When searching for potential stakeholders for a particular project, cast a wide net to avoid overlooking some important community. Then you can focus this candidate stakeholder list down to the core set whose input you really need, to make sure you understand all of the project's requirements and constraints so your team can deliver the right solution.

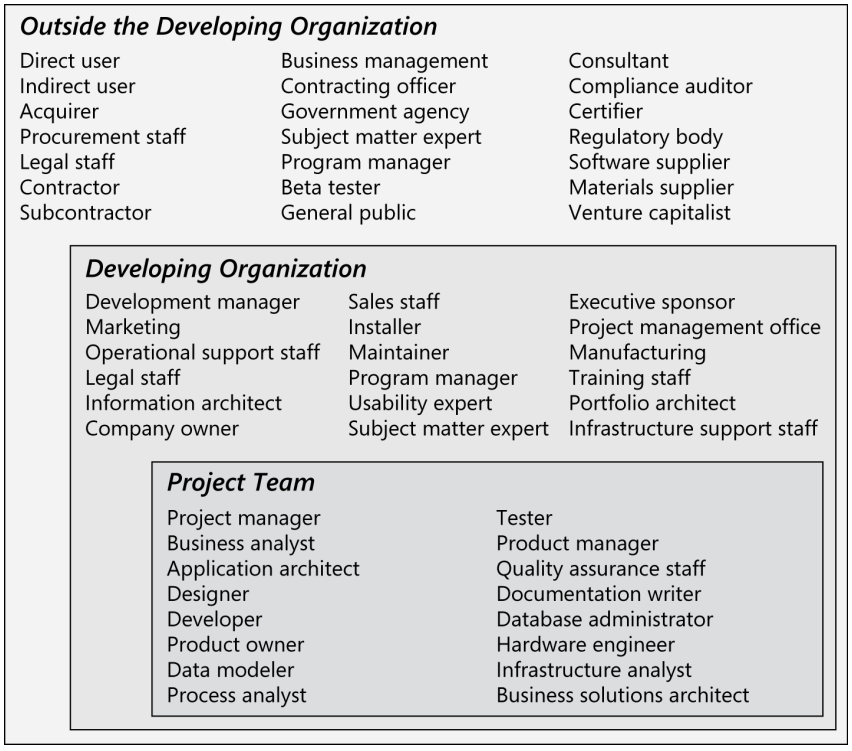

FIGURE 2-2 Potential stakeholders within the project team, within the developing organization, and outside the organization.

Customers are a subset of stakeholders. A *customer* is an individual or organization that derives either direct or indirect benefit from a product. Software customers could request, pay for, select, specify, use, or receive the output generated by a software product. The customers shown in Figure 2-2 include the direct user, indirect user, executive sponsor, procurement staff, and acquirer. Some stakeholders are not customers, such as legal staff, compliance auditors, suppliers, contractors, and venture capitalists. Gerhard, the manager we met earlier, represents an executive sponsor who is paying for the project. Customers like Gerhard provide the business requirements, which establish the guiding framework for the project and the business rationale for launching it. As discussed in Chapter 5, "Establishing the business requirements," business requirements describe the business objectives that the customer, company, or other stakeholders want to achieve. All other product requirements need to align with achieving those desired business outcomes.

User requirements should come from people who will actually use the product, either directly or indirectly. These users (often called *end users*) are a subset of customers. Direct users will operate the product hands-on. Indirect users might receive outputs from the system without touching it themselves, such as a warehouse manager who receives an automatic report of daily warehouse activities by email. Users can describe the tasks they need to perform with the product, the outputs they need, and the quality characteristics they expect the product to exhibit.

The case of the missing stakeholder

I know of a project that was almost finished with requirements elicitation when, while reviewing a process flow, the business analyst (BA) asked the stakeholder, "Are you sure we have the tax calculation steps correct in this flow?" The stakeholder replied, "Oh, I don't know. I don't own tax. That's the tax department." The team hadn't talked to anyone in the tax department over the course of working on the project for months. They had no idea that there even *was* a tax department. As soon as the BAs did meet with the tax department, they found a long list of missed requirements around the legal implications of how tax-related functions were implemented. The project was delayed several months as a result. Using an organization chart to search for all stakeholders who will be affected by a new system can avoid such unpleasantness.

Customers who provide the business requirements sometimes purport to speak for the actual users. They are often too far removed from the work to provide accurate user requirements, though. For corporate information systems, contract development, or custom application development, business requirements should come from the person who is ultimately accountable for the business value expected from the product. User requirements should come from people who will press the keys, touch the screen, or receive the outputs. If there is a serious disconnect between the acquiring customers who are paying for the project and the end users, major problems are guaranteed.

The situation is different for commercial software development, where the customer and the user often are the same person. Customer surrogates, such as marketing personnel or a product manager, typically attempt to determine what customers would find appealing. Even for commercial software, though, you should strive to engage end users in the process of developing user requirements, as Chapter 7, "Requirements elicitation," describes. If you don't, be prepared to read reviews pointing out product shortcomings that adequate user input could have avoided.

Conflicts can arise among project stakeholders. Business requirements sometimes reflect organizational strategies or budgetary constraints that aren't apparent to users. Users who are upset about having a new information system forced on them by management might not want to work with the software developers, viewing them as the harbingers of an undesired future. Such folks are sometimes called "loser groups" (Gause and Weinberg 1989). To manage such potential conflicts, try communication strategies about project objectives and constraints that can build buy-in and avoid debates and hard feelings.

The customer-development partnership

An excellent software product results from a well-executed design based on excellent requirements. Excellent requirements result from effective collaboration between developers and customers (in particular, actual users)—a partnership. A collaborative effort can work only when all parties involved know what they need to be successful and when they understand and respect what their

collaborators need to be successful. As project pressures rise, it's easy to forget that all stakeholders share a common objective: to build a product that provides adequate business value and rewards to all stakeholders. The business analyst typically is the point person who has to forge this collaborative partnership.

The Requirements Bill of Rights for Software Customers in Table 2-1 lists 10 expectations that customers can legitimately hold regarding their interactions with BAs and developers during the project's requirements engineering activities. Each of these rights implies a corresponding responsibility on the part of the BAs or software developers. The word "you" in the rights and responsibilities refers to a customer for a software development project.

Because the flip side of a right is a responsibility, Table 2-2 lists 10 responsibilities that the customer has to BAs and developers during the requirements process. You might prefer to view these as a developer's bill of rights. If these lists aren't exactly right for your organization, modify them to suit the local culture.

TABLE 2-1 Requirements Bill of Rights for Software Customers

You have the right to
1. Expect BAs to speak your language.
2. Expect BAs to learn about your business and your objectives.
3. Expect BAs to record requirements in an appropriate form.
4. Receive explanations of requirements practices and deliverables.
5. Change your requirements.
6. Expect an environment of mutual respect.
7. Hear ideas and alternatives for your requirements and for their solution.
8. Describe characteristics that will make the product easy to use.
9. Hear about ways to adjust requirements to accelerate development through reuse.
10. Receive a system that meets your functional needs and quality expectations.

TABLE 2-2 Requirements Bill of Responsibilities for Software Customers

You have the responsibility to
1. Educate BAs and developers about your business.
2. Dedicate the time that it takes to provide and clarify requirements.
3. Be specific and precise when providing input about requirements.
4. Make timely decisions about requirements when asked.
5. Respect a developer's assessment of the cost and feasibility of requirements.
6. Set realistic requirement priorities in collaboration with developers.
7. Review requirements and evaluate prototypes.
8. Establish acceptance criteria.
9. Promptly communicate changes to the requirements.
10. Respect the requirements development process.

These rights and responsibilities apply to actual customers when the software is being developed for internal corporate use, under contract, or for a known set of major customers. For mass-market product development, the rights and responsibilities are more applicable to customer surrogates such as the product manager.

As part of project planning, the key customer and development stakeholders should review these two lists and negotiate to reach a meeting of the minds. Make sure the participants in requirements development understand and accept their responsibilities. This understanding can reduce friction later, when one party expects something that the other is not willing or able to provide.

Trap Don't assume that the project participants instinctively know how to collaborate on requirements development. Take the time to discuss how those involved can work together most effectively. It's a good idea to write down how you decide to approach and manage requirements issues on the project. This will serve as a valuable communication tool throughout the project.

Requirements Bill of Rights for Software Customers

Following are 10 rights that customers can expect when it comes to requirements issues.

Right #1: To expect BAs to speak your language

Requirements discussions should center on your business needs and tasks, using business vocabulary. Consider conveying business terminology to the BAs with a glossary of terms. You shouldn't have to wade through technical jargon when talking with BAs.

Right #2: To expect BAs to learn about your business and your objectives

By interacting with you to elicit requirements, the BAs can better understand your business tasks and how the system fits into your world. This will help developers create a solution that meets your needs. Invite BAs and developers to observe what you and your colleagues do on the job. If the new system is replacing an existing one, the BAs should use the current system as you use it. This will show them how it fits into your workflow and where it can be improved. Don't just assume that the BA will already know all about your business operations and terminology (see Responsibility #1).

Right #3: To expect BAs to record requirements in an appropriate form

The BA will sort through all the information that stakeholders provide and ask follow-up questions to distinguish user requirements from business rules, functional requirements, quality goals, and other items. The ultimate deliverable from this analysis is a refined set of requirements stored in some appropriate form, such as a software requirements specification document or a requirements management tool. This set of requirements constitutes the agreement among the stakeholders about

the functions, qualities, and constraints of the product to be built. Requirements should be written and organized in a way that you find easy to understand. Your review of these specifications and other requirements representations, such as visual analysis models, helps to ensure that they accurately represent your needs.

Right #4: To receive explanations of requirements practices and deliverables

Various practices can make requirements development and management both effective and efficient, and requirements knowledge can be represented in a variety of forms. The BA should explain the practices he's recommending and explain what information goes into each deliverable. For instance, the BA might create some diagrams to complement textual requirements. These diagrams might be unfamiliar to you, and they can be complex, but the notations shouldn't be difficult to understand. The BA should explain the purpose of each diagram, what the symbols mean, and how to examine the diagram for errors. If the BA doesn't offer such explanations, feel free to ask for them.

Right #5: To change your requirements

It's not realistic for BAs or developers to expect you to think of all your requirements up front or to expect those requirements to remain static throughout the development cycle. You have the right to make changes in the requirements as the business evolves, as the team gathers more input from stakeholders, or as you think more carefully about what you need. However, change always has a price. Sometimes adding a new function demands trade-offs with other functions or with the project's schedule or budget. An important part of the BA's responsibility is to assess, manage, and communicate change impacts. Work with the BA on your project to agree on a simple but effective process for handling changes.

Right #6: To expect an environment of mutual respect

The relationship between customers and developers sometimes becomes adversarial. Requirements discussions can be frustrating if the participants don't understand each other. Working together can open the eyes of the participants to the problems each group faces. Customers who participate in requirements development have the right to expect BAs and developers to treat them with respect and to appreciate the time they are investing in the project's success. Similarly, customers should demonstrate respect for the development team members as everyone collaborates toward their mutual objective of a successful project. Everyone's on the same side here.

Right #7: To hear ideas and alternatives for your requirements and for their solution

Let the BA know about ways that your existing systems don't fit well with your business processes to make sure that a new system doesn't automate ineffective or obsolete processes. That is, you want to avoid "paving the cow paths." A BA can often suggest improvements in your business processes. A creative BA also adds value by proposing new capabilities that customers haven't even envisioned.

Right #8: To describe characteristics that will make the product easy to use

You can expect BAs to ask you about characteristics of the software that go beyond your functional needs. These characteristics, or quality attributes, make the software easier or more pleasant to use, which lets users accomplish their tasks more efficiently. Users sometimes request that the product be *user-friendly* or *robust*, but such terms are too subjective to help the developers. Instead, the analyst should inquire about the specific characteristics that mean "user-friendly" or "robust" to you. Tell the BA about which aspects of your current applications seem "user-friendly" to you and which do not. If you don't discuss these characteristics with the BA, you'll be lucky if the product comes out as you hope.

Right #9: To hear about ways to adjust requirements to accelerate development through reuse

Requirements are often somewhat flexible. The BA might know of existing software components or requirements that come close to addressing some need you described. In such a case, the BA should suggest ways of modifying your requirements or avoiding unnecessary customizations so developers can reuse those components. Adjusting your requirements when sensible reuse opportunities are available saves time and money. Some requirements flexibility is essential if you want to incorporate commercial off-the-shelf (COTS) packages into the product, because they will rarely have precisely the characteristics you want.

Right #10: To receive a system that meets your functional needs and quality expectations

This is the ultimate customer right, *but* it can happen only if you clearly communicate all the information that will let developers build the right product, if developers communicate options and constraints to you, and if the parties reach agreement. Be sure to state all your assumptions and expectations; otherwise, the developers likely can't address them properly. Customers sometimes don't articulate points that they believe are common knowledge. However, validating a shared understanding across the project team is just as important as expressing something new.

Requirements Bill of Responsibilities for Software Customers

Because the counterpart to a right is a responsibility, following are 10 responsibilities that customer representatives have when it comes to defining and managing the requirements for their projects.

Responsibility #1: To educate BAs and developers about your business

The development team depends on you to educate them about your business concepts and to define business jargon. The intent is not to transform BAs into business experts but to help them understand your problems and objectives. BAs aren't likely to be aware of knowledge that you and your peers take for granted.

Responsibility #2: To dedicate the time that it takes to provide and clarify requirements

Customers are busy people; those who are involved in requirements work are often among the busiest. Nonetheless, you have a responsibility to dedicate time to workshops, interviews, and other requirements elicitation and validation activities. Sometimes the BA might think she understands a point you made, only to realize later that she needs further clarification. Please be patient with this iterative approach to developing and refining the requirements; it's the nature of complex human communication and a key to software success. The total time required is less when there is focused effort for several hours than when the time is spent in bits and pieces strung out over weeks.

Responsibility #3: To be specific and precise when providing input about requirements

It's tempting to leave the requirements vague and fuzzy because pinning down details is tedious and time consuming (or because someone wants to evade being held accountable for his decisions). At some point, though, someone must resolve the ambiguities and imprecisions. You're the best person to make those decisions. Otherwise, you're relying on the BA or developers to guess correctly. It's fine to temporarily include *to be determined* (TBD) markers in the requirements to indicate that additional exploration or information is needed. Sometimes, though, TBD is used because a specific requirement is difficult to resolve and no one wants to tackle it. Try to clarify the intent of each requirement so that the BA can express it accurately. This is the best way to ensure that the product will meet your needs.

Responsibility #4: To make timely decisions about requirements when asked

Just as a contractor does while building your fabulous dream home, the BA will ask you to make many decisions. These include resolving conflicting requests received from multiple customers, choosing between incompatible quality attributes, and evaluating the accuracy of information. Customers who are authorized to make such decisions must do so promptly when asked. Developers often can't proceed with confidence until you render your decision, so time spent waiting for an answer can delay progress. When the demands for your time start to feel onerous, remember that the system is being built for you. Business analysts are often skilled at helping people think through making decisions, so ask for their help if you get stuck.

Responsibility #5: To respect a developer's assessment of the cost and feasibility of requirements

All software functions have a cost. Developers are in the best position to estimate those costs. Some features might not be technically feasible or might be surprisingly expensive to implement. Certain requirements might demand unattainable performance in the operating environment or require access to data that isn't available to the system. The developer can be the bearer of bad news about feasibility or cost. You should respect that judgment, even if it means you might not get something you asked for in exactly the form you envisioned. Sometimes, you can rewrite requirements in a way that makes them attainable or cheaper. For example, asking for an action to take place "instantaneously" isn't feasible, but a more precise timing requirement ("within 50 milliseconds") might be achievable.

Responsibility #6: To set realistic requirement priorities in collaboration with developers

Few projects have the time and resources to implement every bit of functionality all customers want. Determining which capabilities are essential, which are useful, and which the customers can live without is an important part of requirements analysis. You have a lead role in setting requirement priorities. Developers can provide information about the cost and risk of each requirement or user story to help determine final priorities. When you establish realistic priorities, you help the developers deliver the maximum value at the lowest cost and at the right time. Collaborative prioritization is key for agile projects, so the developers can begin delivering useful software as quickly as possible.

Respect the development team's judgment as to how much of the requested functionality they can complete within the available time and resource constraints. If everything you want doesn't fit in the project box, the decision makers will have to reduce project scope based on priorities, extend the schedule, or provide additional funds or people. Simply declaring every requirement as high priority is neither realistic nor collaborative.

Responsibility #7: To review requirements and evaluate prototypes

As you'll see in Chapter 17, "Validating the requirements," peer reviews of requirements are among the most powerful software quality activities available. Having customers participate in reviews is a key way to evaluate whether the requirements demonstrate the desired characteristics of being complete, correct, and necessary. A review is also an opportunity for customer representatives to assess how well the BA's work is meeting the project's needs. Busy customers often are reluctant to devote time to a requirements review, but it's well worth their time. The BA should make requirements available to you for review in manageable chunks throughout the requirements elicitation process, not in a massive tome dumped on your desk when the requirements are "done."

It's hard to develop a good mental picture of how software will work from written requirements alone. To better understand your needs and explore the best ways to satisfy them, BAs or developers sometimes build prototypes of the intended product. Your feedback on these preliminary, partial, or exploratory implementations provides valuable information to the developers.

Responsibility #8: To establish acceptance criteria

How do developers know when they're done? How can they tell if the software they built will meet the expectations of the various customer communities? As a customer, one of your responsibilities is to establish acceptance criteria, predefined conditions that the product must satisfy to be judged acceptable. Such criteria include acceptance tests, which assess whether the product lets users perform certain of their important business operations correctly. Other acceptance criteria might address the estimated remaining defect levels, the performance of certain actions in the operating environment, or the ability to satisfy external certification requirements. Agile projects rely heavily on acceptance tests, instead of written requirements, to flesh out the details of user stories. Testers can judge whether a specified requirement was implemented correctly, but they don't always know exactly what *you* will consider an acceptable outcome.