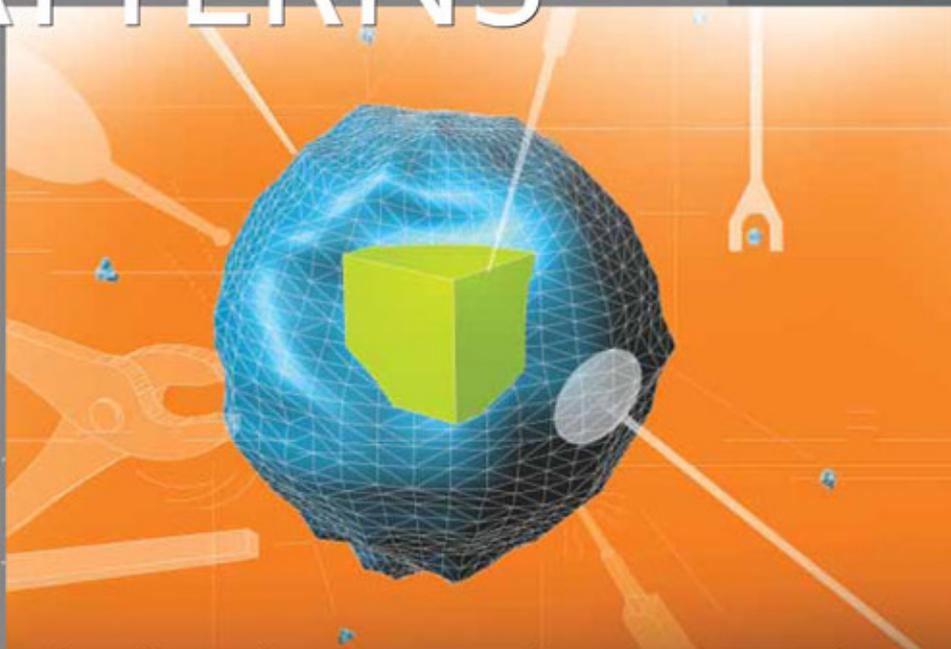


Microsoft

BEST PRACTICES

# SOFTWARE REQUIREMENT PATTERNS



Stephen Withall

Foreword by Karl E. Wieggers  
Author of *Software Requirements*

# Software Requirement Patterns

*Stephen Withall*

**PUBLISHED BY**

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

Copyright © 2007 by Stephen Withall

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Library of Congress Control Number: 2007926327

Printed and bound in the United States of America.

Second Printing: July 2014

Distributed in Canada by H.B. Fenn and Company Ltd.

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

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at [www.microsoft.com/mspress](http://www.microsoft.com/mspress). Send comments to [mspinput@microsoft.com](mailto:mspinput@microsoft.com).

Microsoft, Microsoft Press, Excel, Internet Explorer, Outlook, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other product and company names mentioned herein may be the trademarks of their respective owners.

The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, Microsoft Corporation, nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

**Acquisitions Editor:** Ben Ryan  
**Developmental Editor:** Devon Musgrave  
**Project Editor:** Maria Gargiulo  
**Illustrations:** Stephen Withall  
**Editorial Production:** ICC Macmillan Inc.

This product is printed digitally on demand.

# Contents at a Glance

Part I	<b>Setting the Scene</b>	
1	Synopsis of “Crash Course in Specifying Requirements”	3
2	Synopsis of “The Contents of a Requirements Specification”	11
3	Requirement Pattern Concepts	19
4	Using and Producing Requirement Patterns	39
Part II	<b>Requirement Pattern Catalog</b>	
5	Fundamental Requirement Patterns	51
6	Information Requirement Patterns	85
7	Data Entity Requirement Patterns	119
8	User Function Requirement Patterns	155
9	Performance Requirement Patterns	191
10	Flexibility Requirement Patterns	239
11	Access Control Requirement Patterns	281
12	Commercial Requirement Patterns	325
	Glossary	341
	References	349



# Table of Contents

<i>Foreword</i> .....	<i>ix</i>
<i>Preface</i> .....	<i>xi</i>

## Part I **Setting the Scene**

<b>1 Synopsis of “Crash Course in Specifying Requirements” .....</b>	<b>3</b>
1.1 What Are Requirements?.....	4
1.2 Where Do Requirements Fit in the Grand Scheme? .....	5
1.3 A Few General Principles .....	6
1.4 A Traditional Requirements Process.....	7
1.5 Agile Requirements Processes.....	8
An Extreme Requirements Process .....	9
An Incremental Requirements Process.....	10
<b>2 Synopsis of “The Contents of a Requirements Specification” .....</b>	<b>11</b>
2.1 Introduction Section.....	12
System Purpose .....	12
Document Purpose .....	12
Requirement Format.....	13
Glossary.....	14
References.....	14
Document History.....	15
2.2 Context Section.....	15
Scope.....	15
Major Assumptions .....	16
Major Exclusions .....	16
Key Business Entities .....	16
Infrastructures .....	17

 **What do you think of this book? We want to hear from you!**

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

[www.microsoft.com/learning/booksurvey/](http://www.microsoft.com/learning/booksurvey/)

2.3 Functional Area Sections . . . . . 17  
2.4 Major Nonfunctional Capabilities Section . . . . . 18

**3 Requirement Pattern Concepts . . . . . 19**

3.1 Introduction to Requirement Patterns . . . . . 19  
3.2 The Anatomy of a Requirement Pattern . . . . . 21  
    Basic Details . . . . . 22  
    Applicability . . . . . 23  
    Discussion . . . . . 24  
    Content . . . . . 24  
    Template(s) . . . . . 24  
    Example(s) . . . . . 26  
    Extra Requirements . . . . . 26  
    Considerations for Development . . . . . 28  
    Considerations for Testing . . . . . 29  
3.3 Domains . . . . . 29  
    Domains and Infrastructures . . . . . 30  
3.4 Requirement Pattern Groups . . . . . 31  
3.5 Relationships Between Requirement Patterns . . . . . 32  
    Requirement Pattern Classifications . . . . . 33  
    Refinement Requirements . . . . . 35  
    Divertive Requirement Patterns . . . . . 36  
    Requirement Patterns and Diversity of Approaches . . . . . 36  
    Use Cases for Requirement Patterns . . . . . 37  
    Business Rules and Requirement Patterns . . . . . 38

**4 Using and Producing Requirement Patterns . . . . . 39**

4.1 When and How to Use Requirement Patterns . . . . . 39  
4.2 Tailoring Requirement Patterns . . . . . 41  
4.3 Writing New Requirement Patterns . . . . . 42  
    How to Find Candidate Requirement Patterns . . . . . 43  
    How to Write a Requirement Pattern . . . . . 45

**Part II Requirement Pattern Catalog**

**5 Fundamental Requirement Patterns . . . . . 51**

5.1 Inter-System Interface Requirement Pattern . . . . . 51  
5.2 Inter-System Interaction Requirement Pattern . . . . . 62

5.3	Technology Requirement Pattern	65
5.4	Comply-with-Standard Requirement Pattern	71
5.5	Refer-to-Requirements Requirement Pattern	79
5.6	Documentation Requirement Pattern	81
<b>6</b>	<b>Information Requirement Patterns</b>	<b>85</b>
6.1	Data Type Requirement Pattern	86
6.2	Data Structure Requirement Pattern	94
6.3	ID Requirement Pattern	97
6.4	Calculation Formula Requirement Pattern	102
6.5	Data Longevity Requirement Pattern	107
6.6	Data Archiving Requirement Pattern	110
<b>7</b>	<b>Data Entity Requirement Patterns</b>	<b>119</b>
7.1	Living Entity Requirement Pattern	129
7.2	Transaction Requirement Pattern	133
7.3	Configuration Requirement Pattern	138
7.4	Chronicle Requirement Pattern	144
7.5	Information Storage Infrastructure	154
	Implementation Requirements	154
<b>8</b>	<b>User Function Requirement Patterns</b>	<b>155</b>
8.1	Inquiry Requirement Pattern	156
8.2	Report Requirement Pattern	161
8.3	Accessibility Requirement Pattern	168
8.4	User Interface Infrastructure	187
8.5	Reporting Infrastructure	189
<b>9</b>	<b>Performance Requirement Patterns</b>	<b>191</b>
9.1	Response Time Requirement Pattern	195
9.2	Throughput Requirement Pattern	204
9.3	Dynamic Capacity Requirement Pattern	212
9.4	Static Capacity Requirement Pattern	215
9.5	Availability Requirement Pattern	217
<b>10</b>	<b>Flexibility Requirement Patterns</b>	<b>239</b>
10.1	Scalability Requirement Pattern	241
10.2	Extendability Requirement Pattern	246
10.3	Unparochialness Requirement Pattern	254

10.4 Multiness Requirement Pattern . . . . .	261
10.5 Multi-Lingual Requirement Pattern. . . . .	272
10.6 Installability Requirement Pattern . . . . .	274
<b>11 Access Control Requirement Patterns . . . . .</b>	<b>281</b>
11.1 User Registration Requirement Pattern . . . . .	284
11.2 User Authentication Requirement Pattern . . . . .	295
11.3 User Authorization Requirement Patterns . . . . .	305
11.4 Specific Authorization Requirement Pattern . . . . .	308
11.5 Configurable Authorization Requirement Pattern . . . . .	313
11.6 Approval Requirement Pattern . . . . .	318
<b>12 Commercial Requirement Patterns . . . . .</b>	<b>325</b>
12.1 Multi-Organization Unit Requirement Pattern . . . . .	325
12.2 Fee/Tax Requirement Pattern . . . . .	330
<b>Glossary . . . . .</b>	<b>341</b>
<b>References . . . . .</b>	<b>349</b>
<b>Index . . . . .</b>	<b>351</b>



**What do you think of this book? We want to hear from you!**

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

[www.microsoft.com/learning/booksurvey/](http://www.microsoft.com/learning/booksurvey/)

# Foreword

Requirements development is hard! Requirements analysts often are not adequately trained or experienced, so they do the best they can without necessarily knowing how to write high-quality requirements. Analysts struggle with questions such as “Where do I start?,” “How do I know when I’m done?,” “How detailed should my requirements be?,” “Have I missed any requirements?,” and “Have I overlooked any critical information in the requirements I’ve written?” Unfortunately, there’s no formulaic approach to the communication-intensive challenge of understanding and specifying requirements.

Stephen Withall’s *Software Requirement Patterns* can help any analyst write better requirements. These patterns provide a way to embody comprehensive and structured knowledge about different types of requirements. Requirements development is a journey of exploration, not just a simple collection or transcription process. The patterns Steve presents can help analysts ask the right questions to properly understand and specify requirements of many types in an appropriate level of detail. From the perspective of “know your audience,” the patterns include guidance to assist the developers and testers who must take the requirements to the next development stages. People learn from examples, and they work more efficiently with the help of templates rather than blank pages. To this end, Steve’s requirement patterns provide both templates and examples.

These requirements patterns are applicable to a wide variety of projects and products. You can apply the concepts in the book to develop new requirement patterns specific to your own industry, application domain, or product line. Too many projects begin specifying requirements from scratch, but the requirement patterns let organizations effectively reuse requirements knowledge captured on previous projects.

This book communicates a wealth of wisdom and insight for writing stellar requirements. Through the patterns, Steve points out the value of using a consistent style when writing requirements, which can enhance every analyst’s capabilities. Even if you don’t apply the patterns rigorously, the book contains hundreds of practical tips for specifying better requirements. Use the book as a reference: read the relevant patterns, try them, and absorb the ideas and advice Steve presents. Internalizing those patterns that fit your situation will make them a routine aspect of how you explore, analyze, document, and use software requirements.

Requirement patterns just might represent the next generation of software requirements thinking. Stephen Withall’s *Software Requirement Patterns* will likely remain the definitive treatise on requirement patterns for years to come.

Karl Wieggers  
April 2007



# Preface

## The Purpose of This Book

*There is nothing new under the sun. It has all been done before.*

–Sherlock Holmes: A Study in Scarlet,  
Arthur Conan Doyle

The purpose of this book is to help you decide and define what a new software system needs to do and to suggest what extra features to add to make it a *good* system—or even an *excellent* one. It saves you effort and enables you to be more precise, by providing detailed guidance on how to specify individual requirements. Requirement patterns are encapsulated expertise, conveniently prepackaged for reuse. The book contains 37 requirement patterns, each of which describes an approach to tackling a particular type of situation that crops up repeatedly in all kinds of systems, but focusing on commercial business software. Only a fraction of any system is specific to its business area; the bulk occurs over and over again no matter what your system is for. These patterns cover more than half of all requirements in some systems—a lot more if we add the extra requirements the patterns suggest.

If you're wary of the word “requirement” here, don't be; it doesn't mean you have to be embroiled in paperwork. This book is suitable for use by business analysts using a traditional analysis approach and by software architects and engineers who use agile methods. You can use requirement patterns to help you identify and define what a system needs to do even if you don't write formal requirements as a result.

The requirements for a software system specify the problem it needs to solve—its purpose and goals. If they're omitted or done badly—which is, unfortunately, all too frequently the case—a system is unlikely to be a perfect fit, no matter how well it's implemented. A disturbing proportion of computer systems are judged to be inadequate; many are not even delivered; more are late or over budget. Studies consistently show that the single biggest cause is poorly defined requirements: not properly nailing down a system's purpose and what it must do. Even a modest contribution to improving requirements offers the prospect of saving businesses part of a huge sum of wasted investment.

To build good systems more often, improvements are needed all along the chain. Serious efforts have been (and continue to be) made in nearly all of them. But most fundamental of all is what the requirements themselves actually *say* (and, just as importantly, *fail to say*). That's been neglected, but it's what this book concentrates on. If I want to define a requirement of a specific type, what do I need to write? How do I go about it? What extra requirements should I consider writing? What else should I worry about? This book identifies many areas (big and small) for which requirements are frequently inadequate, imprecise, or missed altogether—and suggests what you can do about it. The patterns themselves aim to be down-to-earth and practical—primarily distilled from personal experience—things I wish I'd known all along.

This is primarily a *reference work*, to be pulled out whenever you want help in a particular situation—to explain what a requirement needs to convey, raise questions to ask, point out potential pitfalls, suggest extra requirements, provide example requirements, and generally provide practical advice. You can start using the requirement patterns without having read the book through.

This book contains lots of example requirements—over 400—many of which are suitable for applying unchanged to any system and others that are a useful starting point for a requirement to suit the reader’s needs. These examples are the heart of the book. It was from the study of real-life requirements that the requirement patterns in this book were identified. Omissions, ambiguities, and other weaknesses in these real requirements fed much of what the requirement patterns have to say.

This book also provides guidance on how to write other kinds of information that belong in a requirements specification—such as system scope, assumptions, a glossary, document history, and references—and how to structure a requirements specification.

### What This Book is Not

This is not a book about the process of specifying requirements or about analysis techniques or requirements management. There are other good books that explain all those things, and this book can be used as a reference alongside them. This book can, however, be used perfectly well by itself; it includes a “crash course in specifying requirements” for readers with no previous experience.

This book doesn’t advocate any particular methodology, approach, or specification software tool. It provides relevant advice no matter which way you choose to work. It isn’t prescriptive: it doesn’t say, “You must do it this way.” It steers clear of jargon and avoids introducing its own terminology as far as possible.

You won’t agree with everything in this book, and you won’t need to act on all the suggestions made by any requirement pattern. But if the time it saves you, when writing requirements or later, is worth more than the purchase price, it has earned its keep. I hope that these patterns prove useful one way or another, by containing enough useful and thought-provoking material to lead you to produce better systems.

## Who Will Benefit from Using This Book

The primary audience of this book is **anyone involved in deciding what a new software system needs to do**. This is the business of specifying the requirements for a software system, even if you don’t like the word “requirement” or you don’t end up writing a full requirements specification. For convenience, we refer to any person who specifies requirements as an **analyst**; they could be a business analyst, a systems analyst, a systems architect, or a software engineer; they could be a business-oriented or technical person. They might have previous experience with specifying requirements, or they might not. They can be divided into those who use traditional analysis processes and those who use more agile methods:

- a. **Business analysts**, or anyone fulfilling that role. This book makes no assumptions about how much the reader knows: it’s suitable for both junior and experienced business analysts as well as for business executives and software engineers who have never specified requirements before. Requirement patterns can be put into practice quickly.
- b. **Software architects and engineers** on any system for which requirements have *not* been written—because the gap must be filled, and it will be one way or another. This book’s advice

is equally relevant no matter who decides what a system needs to do. Its advice is of just as much value to any organization that does not have dedicated analysts, and particularly those that take an *agile* approach to development. Agile methods place little (if any) emphasis on writing requirements specifications, but still the functionality of the system must be identified—and the requirement patterns in the book can help just as well here as when using a traditional approach. In *extreme programming*, in particular, requirement patterns can help you write user stories, interpret user stories, and formulate “rules” for good practices for developers to follow. Software architects and engineers who are familiar with *design patterns* should be particularly comfortable using *requirement patterns*.

Secondary audiences are:

- c. Anyone asked to **review** a requirements specification, which covers a wide range of technical, managerial, and sales people as well as a new system’s user community. This book can help reviewers judge a specification’s quality and completeness, and discover omissions.
- d. **Software developers** who must implement requirements. Each requirement pattern contains a “Considerations for Development” section to assist developers.
- e. **Software testers** who must test how well the delivered system satisfies its requirements. Each requirement pattern contains a “Considerations for Testing” section for testers with suggestions on how to test requirements of that type.
- f. **Project managers** who manage a system’s requirements, changes to them, and a project to implement them.

Job titles of people who will find this book valuable include business analyst, systems analyst, business systems analyst, software architect, systems architect, software engineer, testing engineer, product manager, project manager, project office manager, and chief technical officer.

## Benefits the Reader Will Gain

You, dear reader, will be able to improve your skills and productivity in the following ways from reading this book (and from using it as a reference):

- 1. You will be able to define better requirements—with more detail, precision, and clarity, and with less ambiguity.
- 2. You will be able to write requirements more quickly and with less effort, by taking advantage of the effort already put into the book (reuse!).
- 3. You will recognize extra topics that requirements should specify, further improving their results and making them more complete.
- 4. You will be better able to organize a requirements specification and to write general sections (such as the glossary).

As a result you, your colleagues, and the organization you work for will see further benefits:

- 5. It is easier to estimate the effort needed to build a specified system.
- 6. Development and testing teams will find it easier to understand the requirements.
- 7. The resulting system will better reflect the organization’s needs, potentially yielding considerable extra return on the investment in it. What price can you put on avoiding a big mistake?

8. Fundamental mistakes, misunderstandings, and omissions will be spotted earlier—with potentially huge savings, given that fixing a defect during the design phase costs roughly ten times more than during requirements, and during development ten times more again.

## Skills and Experience Needed by the Reader

This book can be used with no previous experience of specifying requirements. Chapter 1 is a “crash course” containing the bare minimum that a novice reader needs to get started. A good general book on requirements engineering (such as those cited at the beginning of Chapter 1) is a better introduction, and readers who have read them or who are already experienced business analysts are likely to get more from this book. Software engineers using agile methods can use the book in isolation. Anyone responsible for *reviewing* a requirements specification needs no previous knowledge or skills in order to use this book to help them.

This book is accessible to a nontechnical reader. It focuses on writing textual requirements in natural language that can be read by anyone. It is free of arcane diagram formats, deep theory, and jargon. You can read it without knowing UML (Unified Modeling Language) or any other formal technique.

## The Structure of This Book

This book is divided into two parts:

- **Part I: Setting the Scene** These four explanatory chapters open with Chapter 1, “Crash Course In Specifying Requirements,” written for someone who is inexperienced at specifying requirements—but everyone should read it, because it states a few principles that are important to the rest of the book. Chapter 2, “The Contents of a Requirements Specification,” describes the types of material, in addition to requirements, that belong in a requirements specification. The versions of Chapters 1 and 2 printed in the book are merely synopses of much longer, “full” versions that can be downloaded from the associated Web page (as described in the “Supporting Resources” section that follows). Chapter 3, “Requirement Pattern Concepts,” explains what requirement patterns are all about: the basics, what each pattern contains, how they’re organized (into domains), and related concepts. Chapter 4, “Using and Producing Requirement Patterns,” explains how to use requirement patterns and to write your own.
- **Part II: Requirement Pattern Catalog** These are sets of patterns for types of requirements that occur repeatedly, to be used as a reference. It opens with a snapshot of the requirement patterns in this book and then has eight chapters (5 through 12) containing the requirement patterns themselves.

Bringing up the rear are a glossary of terms and acronyms used and encountered in the book, plus a list of references.

I advise that you read through Part I to understand what’s going on. If Chapters 1 and 2 in the book don’t tell you enough, refer to the Web page for the full versions. You don’t need to devour Part II systematically: familiarize yourself with the patterns that it contains (unless you’re an analyst keen for advancement!), and refer to it whenever you encounter a situation in which one of the patterns will help.

## Supporting Resources

You can download the following documents from the book’s companion Web page at <http://aka.ms/623989/files>

1. The full version of Chapter 1, “Crash Course in Specifying Requirements.”
2. The full version of Chapter 2, “The Contents of a Requirements Specification.”
3. “Example Requirements,” a complete set of all the examples in the book, plus the requirement templates for all the requirement patterns, to make it easy to copy and paste an individual template or example to use as a starting point when writing a requirement of your own. This document also includes a requirement pattern template, to use if you want to write your own patterns.
4. A “Ready Reference” suitable for printing, containing a diagram of all the requirement patterns plus a list of all the requirement patterns and the “applicability” of each one (to make it easy to figure out which pattern to use when).

The first two are available both as Adobe PDF (Portable Document Format) documents and Microsoft XML Paper Specification (XPS) documents. The last two are available as Microsoft Word documents. To download these documents, you will need about 6 MB of disk space. For system requirements for viewing these files, see the companion Web page.

## Acknowledgments

I greatly appreciate the diligent and generous contributions of a number of people, without whose assistance this book would have been much the poorer—or wouldn’t even have been completed at all. First, special thanks to Trish Reader for encouragement all the way through, sound business analysis advice, and feedback on various drafts.

I am deeply indebted to all my reviewers, especially those who heroically read and commented cover to cover: to Roxanne Miller, for her deep understanding of what all business analysts will look for in this book, and for keeping me (relatively) honest on analysis techniques; and to Lydia Ash, for her expertise on testing but also countless invaluable suggestions on almost everything. I appreciate the feedback and suggestions of Robert Posener for scrutinizing the text with the all-seeing eye of the consummate project manager; Craig Malone on development methodologies (especially agile matters); Marc Munro for his database expertise on the information and data entity patterns; security guru Eric Fitzgerald on access control; and accessibility experts Annuska Perkins, Norm Hodne, Ramkumar Subramanian, and Laura Ruby. Finally, thanks to Shanno Sanders for perceptive insights on the overall direction of the book. Sometimes I have rashly persisted in disregarding their advice, for which I assume full responsibility—as I do for all errors that remain.

I am grateful to Karl Wieggers for contributing such a generous Foreword, and for the early encouragement that was the nudge I needed to write this book.

I’d like to thank everyone at Microsoft Press, especially acquisitions editor Ben Ryan for his faith in the concept, and editors Devon Musgrave and Maria Gargiulo for their never-failing patience, their good-natured reactions to even the quirkiest of my ideas, and their painstaking copy editing.

Finally, this book could never have been written at all if not for the innumerable people who have contributed to my professional experience over the years. The most valuable have been those at the two extremes of the spectrum: the excellent, from whom I've learned so much about how to specify and develop good systems; and the inept, whose creativity in finding ways to do things wrong is an education in itself. Thanks to you all.

### **Microsoft Press Support**

Every effort has been made to ensure the accuracy of this book. Microsoft Press provides corrections for books through the World Wide Web at the following address:

*<http://www.microsoft.com/mspress/support/>*

To connect directly to the Microsoft Press Knowledge Base and enter a query regarding a question or issue that you may have, go to:

*<http://www.microsoft.com/mspress/support/search.asp>*

If you have comments, questions, or ideas regarding this book, please send them to Microsoft Press using either of the following methods:

Postal Mail:

*Microsoft Press*

*Attn: Software Requirement Patterns Editor*

*One Microsoft Way*

*Redmond, WA 98052-6399*

E-Mail:

*[mspinput@microsoft.com](mailto:mspinput@microsoft.com)*



# Requirement Pattern Concepts

In this chapter:	
<b>3.1 Introduction to Requirement Patterns</b> .....	<b>19</b>
<b>3.2 The Anatomy of a Requirement Pattern</b> .....	<b>21</b>
<b>3.3 Domains</b> .....	<b>29</b>
<b>3.4 Requirement Pattern Groups</b> .....	<b>31</b>
<b>3.5 Relationships Between Requirement Patterns</b> .....	<b>32</b>

## 3.1 Introduction to Requirement Patterns

In all but trivial systems you'll have requirements that are similar in nature to one another or that crop up in most systems—and probably lots of them. For example, you might have a number of inquiry functions, each with its own requirement. When specifying a business system, a significant proportion of the requirements fall into a relatively small number of types. It's worthwhile to make an effort to specify all the requirements of one type in a consistent manner. To do this, we introduce the notion of a requirement pattern, to allow us to describe how each requirement that uses it should be defined.

*Requirement pattern: an approach to specifying a particular type of requirement.*

A requirement pattern is applied at the level of an *individual requirement*, to guide the specifying of a single requirement at a time. For example, if you have a requirement for a certain report, you can engage the report requirement pattern to help you specify it. Once you've written the requirement (and any extra requirements it suggests), the pattern's job is done, and you can put it away and move on. But when a software designer or developer comes to decide how to implement the requirement, the pattern is available to give them some hints relevant to their job, if they wish. A tester can similarly use the pattern for ideas on how to test it.

What are the benefits of using requirement patterns? First, they provide guidance: suggesting the information to include, giving advice, warning of common pitfalls, and suggesting other matters you ought to consider. Second, they save time: you don't have to write each requirement from scratch, because the pattern gives you a suitable starting point, a foundation to build on. Third, patterns promote consistency across requirements of the same type. Of these, guidance is of the greatest value. Saving specification time and increasing consistency are nice, but sound guidance that leads to much better requirements can avoid immense trouble and work later on.

The guidance provided by a requirement pattern usually goes deeper than just "say this..." It can give background insight into the problem at hand. It can raise questions you ought to ask yourself. In some cases it can lead you to write a requirement (or several) very different from the one you first envisaged. Answering a big question often raises a number of smaller questions. A requirement pattern is a response to a big question and aims to give both the big answer and the smaller questions.

Some requirement patterns either demand or invite the specification of extra requirements: both **follow-on requirements** that expand on the original requirement, and systemwide **pervasive requirements** to support the pattern itself (for instance, for an underlying feature needed by all requirements of this type). It is therefore useful to be aware of which patterns you have used (perhaps by keeping a simple list), so you can run through them asking yourself whether each one needs its own extra supporting requirements and whether you have defined them. This topic is explained in more detail in the “Extra Requirements” section later in this chapter.

Patterns can vary in their level of detail (their preciseness) and their value. Some types of requirements can be defined in great detail, and instances of them vary little. Others have something worthwhile in common but vary to such an extent that we can't prescribe what they ought to say. These variations are natural. To justify itself, a pattern simply needs to be of value; it doesn't have to do everything a pattern *could possibly* do. On the other hand, just because we encounter a particular type of requirement repeatedly does not mean a pattern for it would automatically have value. If it's hard to encapsulate what the requirements have in common, it's hard to provide guidance on how to specify requirements of this type.

Where do requirement patterns come from? This book defines patterns for a range of common types of requirements, which can be used as they stand. Other patterns may become publicly available in due course. Or you can write your own—see the “Writing New Requirement Patterns” section in Chapter 4, “Using and Producing Requirement Patterns,” for guidance. You can also modify existing patterns to suit your own preferences and circumstances.

### Pattern Ecosystems

“Each pattern describes a problem which occurs over and over again in our environment.” So says Christopher Alexander, the godfather of the technical use of patterns (as quoted by Gamma, Helm, Johnson, and Vlissides, 1995). In a complex environment, there are many niches for patterns to fill, within which different species of patterns can live together harmoniously.

Individual requirements reside low down in the food chain. Designs live high up the food chain, feeding on the requirements (or, in their absence, on whatever unhealthy carrion they can find). In information technology, different species of patterns can coexist at various scales—big or small—and on both sides of the problem/solution divide. They all have their place: it just depends on what you seek guidance about.

Requirement-related patterns have been suggested at the large scale—for *sets* of requirements (by Robertson and Robertson, 2006) and requirements for a *whole system* (by Ferdinandi, 2002). Conceptually, both of these levels are valid.

Martin Fowler's **analysis patterns** are worth mentioning, too. They live on the other side of the fence from requirement patterns, in the design domain next door. Each one serves to guide the solution of a specific application problem. Analysis patterns are one step higher in the food chain than requirement patterns, and design patterns can feed upon them in turn. (Martin Fowler's analysis patterns can be found at <http://www.martinfowler.com> or in his book [1996].) Tony Morgan presents a few handy **business rule patterns** in his *Business Rules and Information Systems* (2002). (A reminder: see “References” at the back of this book for details on the publications mentioned.)

Patterns to apply at the level of individual requirements are especially useful because of the atomic nature of requirements. That is, requirements have a lot more in common with one another than aspects of design do. This is not to say that requirement patterns are in any

way better than design patterns—certainly not. It merely means that they are easier to apply because we’re always applying requirement patterns to requirement objects that are conveniently self-contained. Also, choosing a requirement pattern delivers a concrete requirement instance to use as our starting point.

Because various species of patterns coexist, this book uses the term “requirement pattern” throughout, rather than just “pattern.” For the same reason, it’s a good idea to always explicitly qualify the word “pattern” with the sort of pattern you’re talking about.

## 3.2 The Anatomy of a Requirement Pattern

This section describes what a requirement pattern contains, how it is organized, and why. Bear in mind that it talks about *patterns* and how to write them, not the requirements that might result from a pattern. A requirement pattern is considerably more substantial than a requirement. Writing a requirement pattern is correspondingly a lot more involved, too. Indeed, writing a requirement pattern deserves much thought. To be useful as a guide to writing requirements, it needs to take into account all the situations and variations that are likely to be encountered in the type of requirement for which it’s written.

A requirement pattern needs to say **when to use** the pattern and **how to write requirements** based on it. It can also give hints on **how to implement** and **how to test** requirements of this type. To convey these sorts of information, each requirement pattern contains the following sections:

1. **Basic details** The pattern manifestation, owning domain, related patterns (if any), anticipated frequency of use, pattern classifications, and pattern author.
2. **Applicability** In what situations can the pattern be applied? And when can it not be applied?
3. **Discussion** How do we write a requirement of this type? What does a requirement of this type need to consider?
4. **Content** What must a requirement of this type say? What extra things *might* it say? This is the main substance of the pattern.
5. **Template(s)** A starting point for writing a requirement of this type—or more than one if there are distinct alternative ways.
6. **Example(s)** One or more representative requirement written using this pattern.
7. **Extra requirements** What sorts of requirements often follow on from a requirement of this type? And what pervasive systemwide requirements might define something for all requirements of this type?
8. **Considerations for development** Hints for software designers and engineers on how to implement a requirement of this type.
9. **Considerations for testing** What do we need to bear in mind when deciding how to test this type of requirement?

Preceding all of this we have the **pattern name**, which appears in the title of the whole pattern. Each requirement pattern needs a unique name so that it can be referred to unambiguously. A pattern name should be meaningful—to clearly capture the nature of the pattern. A pattern’s name should also be as concise as possible—preferably a single word, but not more than three. It’s also recommended that each pattern name be a noun-phrase. The name of a pattern for functional requirements reflects the function name (for example, “inquiry” for the pattern that guides how to specify an inquiry function).

The number of sections in each pattern has been deliberately kept to a minimum to make them as easy to read and to follow as possible.

## Basic Details

The “Basic Details” section of a requirement pattern is simply a convenient vehicle for all those items that can be answered briefly (rather than clogging up our patterns with lots of tiny subsections). In fact, the items lumped together here are a mixed bag. The items indicated by an asterisk (\*) are omitted from all the patterns in this book, either because they are obvious from the context (the chapter they’re in) or because they’re the same in every case.

<p>Pattern manifestation*:</p>	<p>There can be more than one manifestation of a particular requirement pattern, which means different variants or versions. This item tells us which manifestation we have before us, to distinguish it from others. The first manifestation of a pattern is referred to as the “Standard” one. The manifestation can convey one or more of the following things:</p> <ol style="list-style-type: none"> <li>a. The pattern’s version number.</li> <li>b. The date the pattern was last changed.</li> <li>c. The requirements approach (or author).</li> <li>d. The customer organization (company name).</li> <li>e. The requirements specification language (for example, English).</li> </ol> <p>(See the “Requirement Patterns and Diversity of Approaches” section later in this chapter and “Tailoring Requirement Patterns” in Chapter 4 for more.)</p> <p>“Pattern manifestation” is omitted from all the patterns in this book. All should be regarded as being “Standard.”</p>
<p>Belongs to domain*:</p>	<p>Every requirement pattern belongs in a domain, and this states which one.</p> <p>Many requirements for which patterns are used need some sort of supporting infrastructure. For example, requirements for reports imply the existence of an underlying reporting infrastructure for producing those reports. Requirements should be defined for the infrastructure itself. (See the next section, “Domains and Infrastructures,” for more information.)</p> <p>One purpose of identifying whether a particular pattern implies the presence of supporting software is to prompt you to consider whether the requirements for that software have been adequately specified.</p> <p>“Belongs to domain” is omitted from all the patterns in this book because it is clear from the chapter in which the pattern resides. This item becomes essential if the domain cannot be determined from its context, such as a single pattern that resides in a stand-alone document.</p>
<p>Related patterns:</p>	<p>This lists any other requirement patterns that apply to related situations. If it would be helpful, this item can also say a little about <i>how</i> the patterns relate to each other.</p>

---

Anticipated frequency:	<p>How many times is this pattern likely to be used in a typical requirements specification? For rarely used patterns, this is best stated as an <i>absolute number</i>, or an absolute number for each occurrence of a parent requirement. For more commonly used patterns, expressing it as a <i>percentage</i> of the requirements is more useful and easier. Often the frequency is best expressed as a range (of absolute numbers or percentages). This item is also at liberty to point out circumstances in which the frequency might fall outside the normal range.</p> <p>For most patterns, this value is indicative only; it might vary considerably from one system to another. Don't lose sleep if your number of requirements falls outside the suggested range. The frequencies stated for the requirement patterns in this book derive from a diverse set of real-world requirements specifications. Sometimes the actual numbers encountered have been adjusted to create a broader range, to bear in mind factors not present in the specifications studied.</p>
Pattern classifications:	<p>Each requirement pattern can be classified in multiple ways, and this item lists all that apply to the main type of requirement covered by the pattern. No attempt is made to apply these classifications to any of the <i>extra requirements</i> described in the pattern. See the "Requirement Pattern Classifications" section later in this chapter for more information about how to use classifications.</p> <p>Classification lists are given in a standard format of "Name: Value", separated by semicolons. For example:</p> <p>Functional: Yes; Performance: Yes</p> <p>This format is concise, readable, and easy to follow. It allows new classifications to be added without changing the standard structure of requirement patterns.</p>
Pattern author*:	<p>Knowing who wrote a pattern can help you decide whether you want to use it. For patterns written in-house, it tells you who to go to for help.</p> <p>For manifestations other than the first, this identifies both the original author and who <i>tailored</i> it.</p> <p>The pattern author is omitted from all the requirement patterns in this book but should be included in all patterns whose authorship isn't obvious.</p>

---

## Applicability

The "Applicability" section describes the situations in which the requirement pattern can be applied. It should be clear and precise. Conciseness helps too, to let the reader form as quickly as possible a picture of when to use this pattern. It's advisable for the first sentence to capture the essence of the matter, and for the rest to clarify and expand it—just like a requirement, in fact. All such statements in this book begin with, "Use the «*Pattern name*» to...."

Normally, a requirement pattern is applicable in only one clear situation: two different situations usually demand two different patterns. That's not to say that all the requirements that use a pattern will look the same—far from it, because there may be considerable variations between them. Some might have optional extra clauses, and they might vary greatly in their size and complexity. However, they will all share the same underlying nature.

The “Applicability” section can also state situations in which the pattern should not be used, if there's any danger of the pattern being misapplied. If there are no situations of this kind, this statement is omitted. All such statements in this book begin with, “Do not use the «*Pattern name*» to....”

## Discussion

The “Discussion” section of a requirement pattern tells you how to write a requirement of this type. It explains everything it can that's likely to help someone who wants to specify a requirement of this type. It generally opens with an overview, to set the scene. It can describe a process to follow, if figuring what to write in such a requirement isn't straightforward. It can raise topics to which to give special thought. It can mention potential pitfalls. The quantity of this discussion material can vary enormously from one requirement pattern to another: from one paragraph to many pages; it all depends on the nature of the requirement and what there can usefully be said about it.

## Content

The “Content” section is a detailed list of each item of information that a requirement of this type must convey. Each content item begins with a name by which to refer to the item, followed by an indication of whether it's optional, and then general descriptive material. It is presented like this:

1. **Item 1 name** Item 1 description.
2. **Item 2 name (optional)** Item 2 description.
3. ...

It's useful for the description to *justify* the item, to explain its purpose if it's not obvious: if the writer of a requirement understands why it's needed, they are more likely to write it (and to write it well). The description can also suggest what to say about the item in the requirement, offer advice, and generally say anything that might help. The order in which the elements of the content are described is implicitly the best order for them to appear in a requirement.

## Template(s)

The aim of a requirement template is to allow you to copy it into a requirement description to use as a starting point. A template is a fill-in-the-blanks definition for a requirement that is deemed to be typical of the type.

The “Content” section of a pattern can describe various optional topics a requirement might need to address but that aren't relevant in all requirements. When deciding which of these topics to include in a template, our guide is efficiency: to minimize the effort in using the template. If a topic is likely to be needed in only a small percentage of requirements, it's best left out of the template. But we must bear in mind that it's much easier to delete an unwanted item than to type in an item for which we have no starting point. A useful rule of thumb is to include a starting point for a topic in the template if at least 20 percent of requirements are likely to possess it.

Be warned that templates for documents or parts of documents are dangerous, because they can lull you into avoiding the thinking you really need to do, or they give you the impression that all the thinking has already been done. Another pitfall is that you end up saying what the template writer felt should be said when they had a different situation in mind. Nevertheless, if taken with a suitably large pinch of salt, using a template can save a little time when writing a requirement.

Each template is shown within a table like the one below, which is in a form suitable for direct copying into a requirement:

Summary	Definition
The format of the summary	The format of the definition

Additional explanatory notes can follow the table.

Here's an example of a template, from the inquiry requirement pattern, that demonstrates the main aspects:

Summary	Definition
« <i>Inquiry name</i> » inquiry	There shall be an [« <i>Inquiry name</i> »] inquiry that shows « <i>Information to show</i> ». Its purpose is « <i>Business intent</i> ». For each « <i>Entity name</i> », the inquiry shall show the following: <ul style="list-style-type: none"> <li>■ «<i>Information item 1</i>»</li> <li>■ ...</li> </ul>

Both the summary and the definition can contain **placeholders** for variable information, which are indicated by being enclosed in double-angled brackets and in italics—for example, «*Entity name*» or «*Description*». Each placeholder must have the same name as that used in the list of content items in the “Content” section (or sufficiently similar that they can be readily matched up). The summary format typically comprises two parts:

1. A fixed word or brief phrase related to the name of the pattern.
2. A brief description to distinguish this requirement from all others of this type.

A template can contain **optional parts**: items of information that are not needed in all cases. This is indicated by surrounding each option part in square brackets: [like this]. This is indicative only; it doesn't mean that everything *not* in square brackets is always essential. Conversely, an optional item might be essential in a particular situation for which you're writing a requirement.

When a requirement can contain a list of items, a sequence number is added to the name of each one (as with '«*Information item 1*»' in the example above), to allow the template to show more than one. An ellipsis (...) indicates that the list continues and might contain as many items as are needed.

A requirement pattern can contain several alternative templates, each tailored to a particular situation. For example, there might be one for a simple case, one containing every possible item, and one or more in between.

## Example(s)

Each requirement pattern contains at least one—usually more—example requirement definitions that demonstrate use of the pattern in practice. For instance, a typical requirement that uses the pattern may be very simple, but some might need to say more; in such a case, we might give examples of each.

Each example is shown within a box like the one below, containing information exactly as they would appear in real requirements of which they are representative:

Summary	Definition
Summary for the example	The definition of the example

Anything that follows the box (like this) is explanatory material that is not part of the example requirement itself. There ought to be no need for notes to make clear the meaning of any requirement, because requirements should be self-explanatory, but notes can be used to point out an aspect of a requirement that renders it a useful example.

Example requirements need not be consistent with one another. Each one is present to demonstrate a representative situation. Spanning a range of possibilities often demands requirements from different sorts of system and sometimes requirements that conflict with one another. All examples are intended to be *good* examples; there are no examples of *what not to do*. Examples are also intended to be realistic, which means not simplified when being added to a requirement pattern. Sometimes this involves the inclusion of extra clauses that might make an example look a little long-winded; this is preferable to giving an example that says less than one would want in practice.

Some requirement patterns contain real-world examples that can be copied directly into requirements specifications and then modified as you see fit. For example, the comply-with-standard requirement pattern has examples for a range of frequently used, general-purpose standards. A list of examples, then, can serve as a body of knowledge to be tapped at will and not just as representatives that show you what such requirements might look like.

## Extra Requirements

In many situations, one requirement isn't sufficient to say all that must be said: you might (or, in some cases, *always*) need to specify additional requirements to spell out the implications properly. The “Extra Requirements” section in a requirement pattern explains what sorts of extra requirements should be considered and in what circumstances. It provides guidance on what to do beyond simply specifying the main requirement. What other things should you think about? What more needs to be said? If there's nothing further to specify, a requirement pattern's “Extra Requirements” section can be empty.

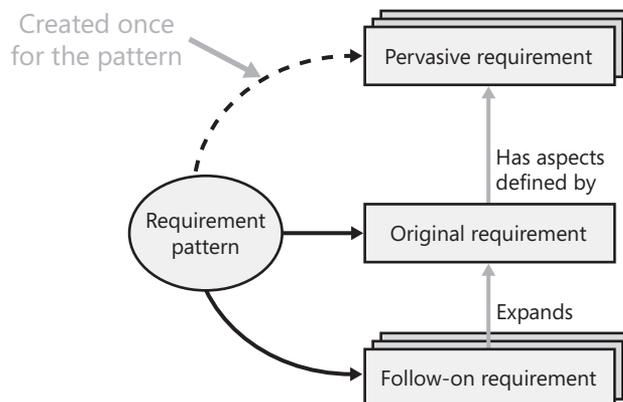
For example, a requirement that mandates compliance with a particular standard (see the comply-with-standard requirement pattern) is rarely sufficient. Just what does that standard say? Which parts are relevant to our system? What must our system do to satisfy this standard? We need detailed requirements that reflect the answers to these questions. The “Extra Requirements” section of the comply-with-standard pattern is the place that points out what further work needs to be done.

There are two types of extra requirements:

- **Type 1: Follow-on requirements** These come after the original requirement and define additional things about it. They expand the original requirement. For ease of reading, follow-on requirements should come immediately after the original requirement.
  - **Type 2: Pervasive requirements** These are defined once for the whole system and define things that apply implicitly to all requirements of this type. Usually there is only one pervasive requirement for a particular aspect (for example, “Every page on every report shall show the company logo”), but sometimes there are more, with each applying in clearly defined circumstances (for example, “Every page on every report to an agent shall show the agent’s company logo,” in addition to the previous example). The use of pervasive requirements of this sort means that each original requirement has fewer topics to cover and can be simpler. They save repetition, and as a result they avoid the chance of inconsistency in their areas.
- Pervasive requirements can also be “catch-alls” that define implicit functions for all instances of this pattern. For example, “All data shall be displayable on some inquiry or another.”

An extra requirement could itself be written with the assistance of a requirement pattern. It can have extra requirements of its own.

Figure 3-1 shows how use of a pattern can result in an original requirement plus two sets of extra requirements: follow-on requirements that add details about the original requirement, and pervasive requirements that define common aspects shared by all original requirements of this type.



**Figure 3-1** Pervasive and follow-on relationships between requirements

You should group related pervasive requirements together, either before all the original requirements to which they relate or after them all. The pervasive requirements for a requirement pattern might look as though they belong in the specification for an infrastructure supporting the domain in which the pattern lives, but actually they should be kept separate. The infrastructure’s requirements define what the infrastructure *can* do; the pervasive requirements define what *our system* needs (because another system might use that same infrastructure differently). That’s fine, although if both systems use the same instance of software that implements the infrastructure, it can impose extra functional demands on that software. In such cases, you can place the pervasive requirements in a separate “common” requirements specification that both systems refer to.

Take care to alert all readers to the presence of pervasive requirements—especially developers, because pervasive requirements often have profound design and development implications. Imagine how you'd feel upon discovering that some characteristic you demanded for *every* user function was possessed by *none* of them. So,

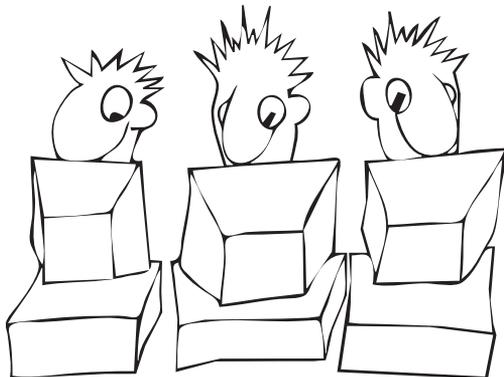
1. Don't rely on readers reading the whole requirements specification. A developer might read only those parts that look relevant to them, plus the introduction.
2. Don't bury important pervasive requirements where they might be missed (such as at the end of the requirements specification).
3. Make references to relevant pervasive requirements from elsewhere as you see fit.
4. Explain in the requirements specification's introduction the significance of pervasive requirements and the importance of not missing them.
5. Consider putting all the pervasive requirements in one place and pointing all developers unequivocally to it.
6. Consider highlighting each pervasive requirement in some way, such as a clear statement at the end of the requirement's definition. For example, "This is a pervasive requirement" or "This requirement applies across the whole system" or "This requirement applies to all user functions in the system".

An "Extra Requirements" section can contain its own example requirements to demonstrate what each kind of extra requirement might look like. If so, follow-on and pervasive requirements should be kept separate and clearly labeled so that they won't get mixed up. Example pervasive requirements are often suitable for direct copying into a requirements specification. In rare cases, the number of pervasive requirements in a pattern could run to a dozen or more.

In extreme cases, the follow-on requirements resulting from a single original requirement could involve more work than all the other requirements put together. For example, complying with a demanding standard (for, say, safety) might be a massive undertaking and much harder than building a simple system that has the functionality you need.

## Considerations for Development

The "Considerations for Development" section is intended to assist a developer who needs to design and implement software to satisfy a requirement of this type. It gives them hints and suggestions and points out things not to forget. The "Considerations for Development" section should be written in the language of developers.



The best way to look at this section is as the sort of guidance a very experienced developer would give to a junior developer. If a grizzled, seen-it-all engineer were asking a wet-behind-the-ears graduate to implement a requirement of this type and giving them advice on how to tackle it, what would they say? The amount to be said varies greatly from one requirement pattern to another. In some cases, the requirement is self-explanatory; in others, there are various pitfalls to point out.

This section can also point out things that a development representative could look out for when *reviewing* requirements. Is a requirement being unreasonable? If it's likely to be impractical to implement, press to have the requirement changed.

## Considerations for Testing

A requirement pattern is a useful place to explain how to test requirements of its type. This section is aimed at testers. It is written primarily with user acceptance testing in mind, because that's the sort of testing that can be done most closely against the requirements. But it can be used for other sorts of testing, too.

Since requirements vary considerably in their nature, they vary as much in the ways they need to be tested. Each “Considerations for Testing” section aims to convey three sorts of information:

1. Points to look out for when reviewing a requirement of this type. If a requirement is likely to be difficult to test, suggest how it can be reframed to make testing easier.
2. Overall guidance on how to approach the testing of this type of requirement.
3. Notes on matters to bear in mind and (where possible) tips on how to deal with them.

Universal requirement patterns can discuss testing only in general terms—because a pattern knows nothing about a particular organization's testing practices, the testing tools it uses, the nature of the environment in which the system will run, or the nature of the system.

An organization may well find it worthwhile to tailor—or rewrite—the “Considerations for Testing” sections in its patterns to suit the ways it does testing (taking into account, in particular, whatever tools it uses for testing) and the expertise and culture of those responsible for testing. Indeed, by taking into account the organization's individual situation, it's possible to write sections that are far more than *considerations* for testing; they can become *instructions* on how to test requirements of the requisite type. If you aim to do this, you may find it more useful to leave the “Considerations for Testing” section alone and add (or replace it with) your own “Testing Instructions” section. This could be augmented by additional artifacts to use when testing this type of requirement, such as tailored forms for writing test cases.

## 3.3 Domains

There is much to be gained by organizing our requirement patterns rather than presenting a monolithic list of them. We do this by assigning every requirement pattern to a **domain**. Each domain has a theme, which all its patterns share, but the nature of the theme can vary greatly from one domain to another. The domains used in this book—each with its own chapter in Part II, “Requirement Pattern Catalog”—are **Fundamental**, for things that any kind of system might need; **Information**, for several aspects of the storage and manipulation of information (data); **Data entity**, on how to treat specific kinds of data; **User function**, for a couple of common types of functions, plus accessibility; **Performance**; **Flexibility**; **Access control**; and **Commercial**, for

business-oriented matters. This shouldn't be regarded as a definitive list: new domains might be needed if further requirement patterns are written. For example, if requirement patterns were to be written for a particular industry, they would deserve their own domain (or possibly more than one). The applicability of a domain can range from very broad to very narrow: from nearly all systems, through systems in one industry, to just a couple of systems in a single company.

When you begin specifying a system, you can look through all the requirement pattern domains (in this book, plus any extras you have from elsewhere) and decide which ones are relevant to your system. If it's a noncommercial system, you might decide to drop the Commercial domain. The set of requirement patterns that are available for use in your system depends on the domains you decide are relevant. Regard only the patterns in your chosen domains as available. Conversely, if you want to use some other pattern, it can force you to add a domain you hadn't previously recognized, which can alert you to extra topics you need to address (such as an infrastructure it depends upon). Identifying the available patterns is more useful if you have patterns in specialized domains; the patterns in this book are too general for drawing up a list of relevant domains to make much of a difference.

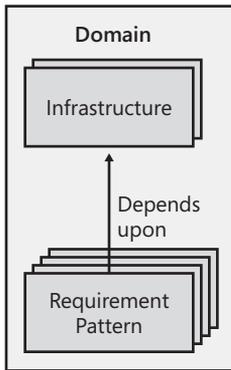
Each domain needs an introduction to explain its theme. It can then describe features that are common to all its patterns. The size of an introduction could be anywhere from one short paragraph to many pages; it depends solely on how much there is to say. The domain also needs to describe any infrastructure(s) its patterns depend upon (or, more properly, the requirements produced by these patterns), as discussed in the next section.

## Domains and Infrastructures

Some types of requirements depend upon infrastructures, as discussed in the “Infrastructures” section of Chapter 2, “The Contents of a Requirements Specification.” A requirement pattern gives us the opportunity to identify any infrastructure(s) its type of requirement depends upon, which saves having to figure them out for individual requirements. We can go further and discuss each infrastructure—things to bear in mind when you specify requirements for the infrastructures your system needs. It's not possible to go into detail or specify actual requirements because they will vary considerably according to the demands of each organization and each system. They're called **infrastructure overviews** to make this clear.

Rather than expect that each requirement pattern describes any infrastructure it needs, we pass the burden of explanation up to the domain the pattern belongs to. This is because each infrastructure tends to be needed by more than one pattern in the domain. To avoid repetition, each type of infrastructure is described in only one domain. Each chapter of patterns in this book contains a subsection for each infrastructure in its domain. This book discusses three infrastructures: information storage (in Chapter 7, “Data Entity Requirement Patterns”), user interface, and reporting (both in Chapter 8, “User Interface Requirement Patterns”). The key concepts relate to each other as shown in Figure 3-2.

A requirement pattern is free to use infrastructures in other domains. It's always good practice to avoid mutual dependencies, so if anything in one domain depends on another domain, nothing in the latter domain should depend on the former—that is, if you can avoid it. An infrastructure can also depend upon another infrastructure.



**Figure 3-2** Relationships between domains, requirement patterns, and infrastructures

What should such an infrastructure overview say? Its role is to give guidance and advice to someone who's going to specify requirements for an infrastructure of this kind for a particular system, to suggest *topics* for requirements to cover. At a minimum, it should state what a calling system expects from the infrastructure: what it's there for, and its main functions. For problems for which there are obvious alternative solutions, the overview should avoid making judgments.

Each infrastructure overview is divided into the following subsections:

1. **Purpose** An explanation of why the infrastructure exists, the role it plays.
2. **Invocation requirements** Suggestions for sorts of requirements that define how a system will interact with the infrastructure—for those functions that the infrastructure must make available to a system—plus any other capabilities systems might want from it (such as access control). The needed functions can be regarded collectively as the *interface* (or *interfaces*) the infrastructure needs to make available to callers.
3. **Implementation requirements** Some ideas on the other features the infrastructure needs in order to stand on its own feet (for example, inquiry, maintenance and configuration functions). These are brief and merely hint at the likely main areas of functionality to think about when defining the infrastructure itself.

For example, for a reporting infrastructure, our invocation requirements might be very simple: just a function that lets our system request the running of a chosen report. The implementation requirements, on the other hand, would be far more extensive, addressing the complexities of various ways of delivering a report to a user, other ways of requesting reports, designing reports, and so on. (These topics are discussed further in the “Reporting Infrastructure” section in Chapter 8.) To take the analogy of building a house, one of the infrastructures we'd want is an electricity supply. In this case, the invocation requirements would cover how many sockets we want in each room, and the implementation requirements would deal with the parts you can't see, such as the connection to the power grid and adherence to building quality regulations.

### 3.4 Requirement Pattern Groups

When several requirement patterns have features in common, we can create a **requirement pattern group** and use it as a place to describe all of their common aspects, to save repeating them in each individual pattern. A requirement pattern group is not a requirement pattern: you can't create requirements of this type. But the definition of a group can contain any of the following sections

that appear in the definition of a requirement pattern: “Extra Requirements,” “Consideration for Development,” and “Considerations for Testing.” The rule is to include whichever of these sections in which something valuable can be said and to omit the rest. Whenever one of these sections is present in the group, it’s worth including a note in the equivalent section in each pattern based on the group as a reminder to refer to it.

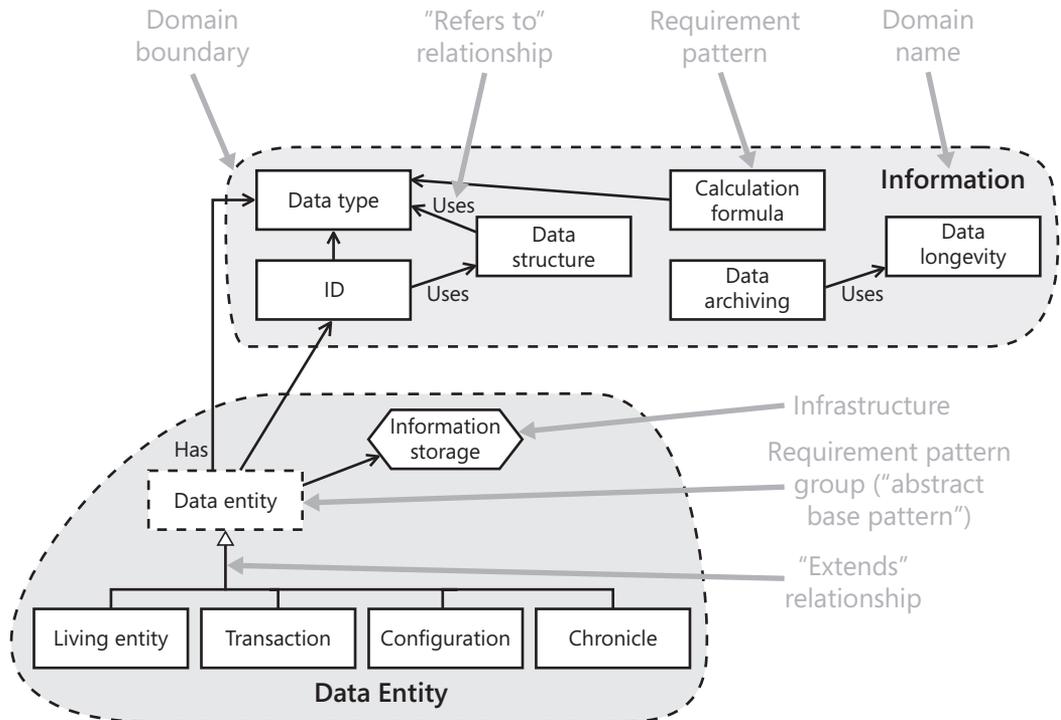
The difference between a *domain* and a *requirement pattern group* is that the patterns in a domain share a common **theme**, whereas those in a group have **detailed features** in common. The patterns in a group don’t need to belong to the same domain. (For those familiar with Java programming, the relationship between requirement patterns and domains is akin to that between classes and packages: every class belongs to a package just as each pattern belongs to a domain. Also, a requirement pattern can build upon a pattern belonging to a different domain, just as a Java class can extend a class in a different package.)

### 3.5 Relationships Between Requirement Patterns

When you use a requirement pattern, it generally says everything you need to know to create a requirement of that type. But a pattern might refer to other patterns for several reasons. Two fundamental types of relationship between requirement patterns exist:

1. **Refers to** A requirement pattern can mention another pattern somewhere in its definition. There are several reasons why a requirement pattern might refer to another:
  - a. A requirement defines something that contains (**has**) something else defined by another requirement.
  - b. A requirement that’s an instance of one pattern **uses** information defined in requirements that are instances of a second pattern. For example, a requirement that defines a data structure might use a value of a kind defined by a data type requirement.
  - c. A requirement might suggest the creation of an **extra requirement** of a type for which a pattern is available.
  - d. A **divertive** pattern might persuade you to create a requirement using a different pattern. (See the “Divertive Requirement Patterns” section later in this chapter.)
  - e. A requirement pattern could refer to another pattern that contains relevant discursive information on a particular topic.
2. **Extends** A requirement pattern builds upon (or is a specialization of) another pattern. In object-oriented terms, this is an *inheritance* relationship. Instead of extending another pattern, a requirement pattern can build upon (extend) a requirement pattern group. (In object-oriented terms, the group acts like an abstract base class for the pattern.) A requirement pattern is not allowed to extend more than one pattern or group.

We can draw a collection of patterns and infrastructures and the relationships between them in a diagram. Figure 3-3 is an example that shows two domains, with elements of the notation annotated. Inheritance is the most important type of relationship. For simplicity, every other type of relationship is shown as “Refers to,” though its role can be indicated by a label on the link. When showing several domains on one diagram, it can become impractical to show all the relationships. In such a case, show all relationships *within* each domain and all *extends* relationships, but omit *refers to* relationships between domains as you see fit. For readers familiar with object-oriented concepts (or UML), this is akin to a class diagram showing relationships between classes.



**Figure 3-3** Annotated sample requirement pattern diagram

In Figure 3-3, “Data entity” is not a pattern. It’s a requirement pattern group: a place for describing the common features of the four patterns that build on it. Any descriptive information that applies to all patterns in a group should be given for the group, rather than repeated for each pattern. Also, by convention, labels for relationships between requirement patterns are placed nearer the subject of the relationship, rather than the object. So, it is a Data entity that *has* an ID (rather than vice versa). The hexagonal shape of “Information storage” denotes it as an *infrastructure*.

See the beginning of Part II of this book for a diagram of this sort for all the requirement patterns in this book. Each of the eight domain chapters contains a diagram of this sort in its introduction, with annotations giving a brief explanation of each pattern, pattern group, and infrastructure.

## Requirement Pattern Classifications

Requirements can be classified in various ways (for example, by dividing them into functional and nonfunctional ones). Using requirement patterns has the advantage that if we classify the patterns, we automatically classify the requirements that use those patterns. Classifications tell us a little about the nature of the requirements that result from using each requirement pattern.

Other ways of using these classifications include finding requirements according to their classification and producing statistics. People like statistics (some people, at least, and they tend to be senior executives it’s worth our while keeping happy). Statistics on the requirements for a system can be useful in a variety of ways. They can give a rough picture of the scale and complexity of the system. To do this, we need to tag each requirement with whatever values are needed for all the sorts of

statistics we want. (Requirements management tools typically do this by letting you define extra requirement “attributes.” Then you enter the value of each attribute for each requirement. It’s a tedious business.) Using requirement patterns can save some of this effort, because all requirements created using a pattern have attributes in common. They need be defined only once, when the pattern is written. This information is recorded in the “Pattern classifications” section of each pattern.

Once requirements are tagged in this way, it’s also possible to search on the classifications to find all the requirements that match your criteria. How you transport this classification information from the patterns to your requirements depends on how you store the requirements. (This is left as an exercise for the reader!) A straightforward way is to copy the requirements into a spreadsheet, add a column identifying the requirement pattern used by each requirement (if any), and add a column for each classification. (Sorting on the pattern name makes it easier to apply classification values many at a time.)

The requirement patterns in this book contain classifications according to a small number of basic classification schemes that are defined below. You can define extra classification schemes of your own and classify patterns according to them. If you do, write proper definitions in a similar manner to those below, and make them available in the same place as the requirement patterns that refer to them.

Classifications can be defined that assist anyone who uses requirements, including developers. As a result, it’s not necessary for everyone to understand every classification. For this reason, each classification has its primary audience explicitly stated. If you’re not part of this audience, don’t worry if you can’t follow what it’s for or if you’re not interested in it.

Requirement pattern classifications need to be properly and precisely defined, or else any statistics based on them can’t be regarded as reliable. Each classification needs the following defined for it:

Name:	A self-explanatory, unique name for the classification.
Audience:	An explanation of who is likely to be most interested in this classification: who it’s aimed at.
Purpose:	A description of what the classification is intended to be used for.
Allowed values:	A definition of the values that a pattern may have for this classification, and explanations of their meanings. The most common way is to define a list of individual values. Numeric or alphabetic or other kinds of values may be used, provided it’s clear to the reader what each value means.
Default value:	This is the value assumed for this classification if it’s not present (explicitly stated) in a pattern. This saves cluttering patterns with explicit mentions of classifications that are meaningful for relatively few patterns (that is, the few that have a significant value for it).

There are three requirement pattern classifications used in this book, which we now describe using this format.

### “Functional” Classification

Name:	<b>Functional</b>
Audience:	Anyone interested in picking out the functionality of the system, or the number of functions.
Purpose:	This indicates whether a requirement of this type defines a function that must be provided by the system.

Allowed values:	<b>Yes</b>	Every requirement of this type is a functional requirement.
	<b>Maybe</b>	Some requirements of this type are functional requirements; some are not. If you're writing a pattern, use this value with care. Ask yourself whether your pattern is well-defined; perhaps it ought to be split into two, one for the functional part and one for the nonfunctional.
	<b>No</b>	No requirement of this type is itself a functional requirement.
Default value:	<b>No</b>	

### "Pervasive" Classification

Name:	<b>Pervasive</b>
Audience:	Software developers
Purpose:	This indicates whether a requirement of this type is pervasive (that is, applies systemwide). Its intent is to bring to the attention of developers requirements that may apply to them no matter which part of the system they're developing.
Allowed values:	<b>Yes</b> Every requirement of this type is pervasive.
	<b>Maybe</b> Some requirements of this type are pervasive; some are not.
	<b>No</b> No requirement of this type is pervasive.
Default value:	<b>No</b>

### "Affects Database" Classification

Name:	<b>Affects database</b>
Audience:	Database administrator (and software developers, too)
Purpose:	This indicates whether a requirement of this type has an impact on the design of the system's database. Its intent is to highlight those requirements that are of most interest to whoever is responsible for designing the database.
Allowed values:	<b>Yes</b> Every requirement of this type affects the database.
	<b>Maybe</b> Some requirements of this type affect the database; some do not.
	<b>No</b> No requirement of this type itself directly affects the database (though this doesn't necessarily mean a database administrator will have no interest in it).
Default value:	<b>Maybe</b>

## Refinement Requirements

It is good practice to keep the size of each requirement's definition within moderate bounds; one that runs to ten paragraphs is way too long. A requirement pattern may identify several pieces of information, although a typical requirement of its type might possess only one or two. From time to time you might have a requirement that possesses more and, as a result, is unreasonably large. In this situation, it makes sense to split the requirement into two or more requirements.

The way to do this is to retain the initial requirement but to cut out parts and make them into additional requirements, which are *refinements* of the main requirement. Each refinement requirement should specify one extra aspect. And each refinement should identify the requirement it builds on. For readability, the main requirement should be immediately followed in the specification

by its refinements. This practice makes sense whether or not you're using a requirement pattern—but when you *are*, you can regard the pattern as applying to both the main requirement and the refinements. If a requirement pattern suggests that several pieces of information be present, this is satisfied if all are present in one of the requirements or another. A second reason for splitting a requirement is if different parts of it have different priorities.

Depending on the nature of the system you're specifying, up to a quarter of all requirements could be refinements of other requirements. If you use a very fine level of requirements granularity, you'll increase the number of requirements and with it the percentage of requirements that are refinements.

## Divertive Requirement Patterns

Usually when you apply a requirement pattern, the result is a requirement that matches what you asked for. However, a pattern could be sneakier than this: it could try to lead you away from the obvious and towards an alternative, better way of formulating what you want. It explains the difficulties that the obvious way causes (usually for developers), and it provides advice on how to avoid these problems by using requirements that are stated in different terms but that aim to achieve the same underlying aim. A divertive requirement pattern can either explain the alternative itself or it can divert you to a different requirement pattern entirely—or both.

Requirement patterns can be much more valuable than just saying, “If you want to require X, this is what to write....” They can be like having an expert sitting at your shoulder saying, “Hang on! If you specify it like that, you're asking for trouble. Let me explain. Why don't you try this instead?” Several of the performance patterns are divertive, because the most obvious ways to specify performance are often a nightmare to satisfy. (For example, “The system shall be available 24×7” gives developers little idea of what they must do to achieve it.)

## Requirement Patterns and Diversity of Approaches

There's no single right or best way of formulating or expressing requirements. **For a given system, there's no single *perfect* set of requirements to define it.** Different requirements approaches might break down a problem in different ways, resulting in requirements that vary in their level of granularity and the way they're expressed. The term “requirements approach” as used in this section simply means a way of going about the specifying of requirements in general or certain types of requirements in particular. Each approach could have its own set of requirement patterns. They might simply be the approach's own distinct manifestations of recognized standard patterns, or they might be patterns specific to the approach.

We can accommodate a diversity of requirements approaches—to let the proponents of each approach create whatever requirement patterns they wish, including their own manifestations of existing patterns. And by recognizing different approaches explicitly, we make the available choices clearer to analysts.

Nevertheless, the greater agreement there is on standard patterns (and the fewer different manifestations of patterns), the better. It's perhaps a testament to the excellence of the choices for design patterns made by Gamma et al. that there's been no apparent call for variants (although lots of extras have appeared), and thus accommodating diverse sets hasn't been necessary. In allowing for multiple requirements approaches, I'm heading off potential criticism of the requirement

patterns given here. I can cite it as proof that there's no single "right" set of requirement patterns. If anyone doesn't like them, they can devise their own alternatives without demanding to replace the ones here. There's room for many schools of thought.

To avoid potential confusion, don't mix material for multiple requirements approaches in the same requirement pattern. It is clearer to have multiple manifestations of the pattern and to pick only one to use on your system.

When creating a new pattern (or manifestation of an existing pattern) for a particular requirements approach, state the approach it relates to in the "Manifestation" section of the pattern. Note that when a manifestation is created for a different approach, it takes on a life of its own and might go through a succession of versions independent of the manifestation for the original "standard" approach.

Where two sets of requirement patterns exist that cover the same ground, there are two ways they can be organized:

1. One domain specification can contain both sets of requirement patterns. (A "domain specification" is a document, or part of a document, that contains its requirement patterns and a section about each of its infrastructures. Each of the eight domain chapters that follows is a domain specification.)
2. There can be two manifestations of the domain specification, each containing one set of the requirement patterns.

The second way is easier and less confusing to use (and thus less liable to have an analyst apply the wrong manifestation of a pattern). The second way also allows each manifestation of the infrastructure specification to be tailored to the methodology its patterns use, which is useful if the infrastructure's own requirements use these patterns (which they might).

## Use Cases for Requirement Patterns

It is perfectly possible to write use cases for some requirement patterns, those that result in requirements that demand the presence of a well-defined function (or, indeed, more than one function). For example, for the inquiry requirement pattern we could write a use case that shows the steps in a typical inquiry. Use cases for requirement patterns are always *generalizations*, an official UML concept that means they are written to apply in any circumstance that fits, using an "is-a-kind-of" relationship.

One requirement pattern might demand the presence of more than one function for each of the resulting requirement. For example, the configuration requirement pattern implies the presence of create, read (inquire), update, and delete functions (commonly called CRUD) for each item of configuration data. We can write a use case for each of these functions, and four use cases will suffice for all configuration, rather than attempting to write four for each type of configuration (or, more likely, writing use cases for a few but not the rest).

It makes sense to write requirement pattern use cases to suit your particular environment. To attempt to write universally applicable use cases risks them being so high-level as to be of no practical value. For example, a universal "create configuration data" use case has little to say—perhaps just an actor entering the data and the system storing it in a data store. But if we have a browser-based user interface with remote users and a Web server that is outside our system's

scope, the use cases will look very different. Also, the use cases' preconditions might insist the actor be logged in and authorized to access this type of configuration data—to satisfy particular security requirements. (All this illustrates how hard it is to write detailed use cases without bringing in elements of the *solution*, even though use cases are meant to reflect only the *problem* to be solved.) No use cases have been written for the requirement patterns in this book.

## Business Rules and Requirement Patterns

A **business rule** is a definition of one aspect of how a business operates. For example, a business rule could define how a particular business should act in a given situation (such as when a customer credit card payment is rejected) or a constraint (such as a policy of not selling to anyone under sixteen). A **business rules approach** to building systems recognizes the importance of business rules, with a view to making it easier to understand and change how the business works. In an ideal world, you'd be able to modify your business rules and all affected systems would instantly jump in line. It's a very attractive prospect. There exist **business rules products** to help you do this. They act a bit like a guru who knows your business inside-out and of whom you can ask questions, but there's a lot more to it than that. That's not to say you need a specialized product to adopt a business rules approach. Two places to go for more information are <http://www.brcommunity.com> and <http://www.businessrulesgroup.org>.

Quite a few types of requirements reflect business rules, including several covered by requirement patterns in this book. So why not say which ones they are? The trouble is that there isn't a single agreed-upon set of business rule types. There are many, and it would be arbitrary to pick one. (The same argument applies when you consider mentioning in an individual requirement how it maps to a business rule.) One could create a requirement pattern classification for a selected business rule scheme to indicate how each pattern relates to a business rule. My excuse for not doing so in this book is not wishing to offend any scheme I left out.

It's also worth pointing out that adopting a particular business rule classification scheme might not happen until *after* the requirements have been written. Consider the case where you invite tenders from three prospective suppliers. The first might use one business rules scheme. The second might use a different one. And the third might not think in business rules terms at all.

Nevertheless, if your organization has made a commitment to using a particular business rule scheme, you can write requirements in a way that's friendly to that scheme (and, if you wish, mention the type of business rule that each requirement reflects, where applicable). If you're committed to using a business rule product, you can treat it as an *infrastructure* that your system must interface to. Then, just like for any infrastructure, you can specify requirements for what you need from it, and use those requirements as your basis for choosing the most suitable product.

# Index

All page numbers that begin with “F-” refer to the full versions of Chapters 1 and 2. These are not in the printed book, but are available for download from <http://www.microsoft.com/mspress/companion/9780735623989>.

## A

- acceptance of requirements, F-21–F-22
- acceptance tests, 9, F-25
- access control
  - chronicle, 149, 150–151
  - configuration, 140
  - for data archiving, 115
  - described, 281
  - infrastructure considerations, 283
  - integral to a software system, 282–283
  - making more friendly, 306
  - for organization units, 329
  - recording changes in, 317
  - for reporting, 190
  - underlying mechanism for, 318
- access control domain, 29–30, 281
- access control extensions, 234–235
- Access Controller, F-57
- access control requirement pattern, 281–323
- access only when logged in requirement, 310
- access privileges for users, 285
- access rights. *See* privileges
- access rules, applying, 308–309
- accessibility
  - authentication alternatives, 298
  - as a source of nonfunctional system aspects, F-70
  - specific needs for, 168, 169, 172, 174–175, 178–182, 183–184
- accessibility requirement pattern, 168–186
- accountants, as programmers, 332
- accounting entries, 260
- accounting system interface, 56
- ACID properties, 123, 154
- acknowledgement of receipt, 58, 59
- acronyms, use of, F-47–F-48
- actions
  - approving, 318
  - authorizing within functions, 308, 310
  - not approving own, 322
  - pending, storage of, 129
- active attack, 60, 61
- active user sessions, terminating, 300
- active users, number of, 149
- activity peaks, 213
- activity triggers, 209
- actors, F-13, F-69
- adapters, 53, 54, 250
- address formats, unparochialness and, 259
- “Affects Database” pattern classification, 35
- agile manifesto, 8, F-22
- agile outlook, adopting, F-22
- agile requirements processes, 8–10, F-22–F-28
- alarm monitor interface, 56
- alarms, 214, 233
- alphabetic characters, 87
- alphabetic mnemonic, F-37
- alphanumeric characters, 87
- alternative templates, 25
- ambiguities of fee calculation, 336–337
- Americans with Disabilities Act, 170
- analysis paralysis, F-4
- analysis patterns, 20
- analyst, F-8, F-16
- anecdotal checking, 261
- animated graphics, 188
- Apache Web server, 69
- Applicability section, 21, 23–24, 45
- approval, 321–322, 323
- approval mechanisms, 322
- approval requirement pattern, 281, 282, 318–323
- “approved” version, determining, F-52–F-53
- architecture, infrastructures contributing to, F-64
- archiving
  - chronicles, 150
  - data. *See* data archiving
- artifacts in agile development, F-23
- assistive technology, 172, 176, 178–179, 185
- assumptions, 16, F-59, F-60
- atomic changes, 123
- atomic nature of requirements, 20–21
- attachments, email, 123
- attributes, defining, 34
- audience, identifying, 13, 34, F-33
- audio alert, visual cue for, 182
- audit trails, 121, 190
- Australian Privacy Act of 1988, 75
- authentication
  - biometric, 178
  - defined, 295
  - information for users, 285
  - mechanisms, 296
  - strength of, 308, 311

## authors

- for each reference, F-50
- listing in the document history, F-52
- of a requirement pattern, 23
- authority, ability to delegate, 307
- authorization
  - chronicling changes, 317
  - example requirements, 309–313
- authorization checks, 317, 318
- authorization inquiries, 316–317
- automatic refreshing of inquiries, 157, 159, 177, 188
- availability
  - of an interface, 57, 58
  - lack of. *See* downtime
  - scalability and, 242, 244
- availability requirement pattern, 217–238
- availability window, 208, 219, 221, 222–223

**B**

- back door access, preventing, 293
- background processing, responsibility for, 282
- backups. *See also* data archiving
  - database, 110
  - information, 123
  - reports, 190
  - system, 229
- Basic details section of requirement pattern, 21, 22–23, 45
- basis of a fee or tax, 332–333
- binary numbers, 87
- binding part. *See* formal part
- biometric authentication, 178
- biometric reader, 285
- blanket bans, 309, 312–313
- blanket permission, 309
- blocking users, 298, 302–303
- Boolean operators, 88
- brackets, double-angled and square, 25, 46
- break locked session, 301
- bureau service, 329
- business entities, identifying, 16, F-61
- business hours, availability during, 222
- business intent of inquiry, 156–157
- business motivation behind a system, F-31
- business purposes, data types for, 86
- business rule patterns, 20
- business rules, 38
- business significance, 147
- business systems, data categories in, 120–122
- business volume, growth in, 241

**C**

- calculations, 102–107
- calculation formula, subcalculations, 103, 106
- calculation formula requirement pattern, 102–107

- calendar date, 88, 259
- calendar time period, 336–337
- candidate glossary terms, 14, F-45
- candidate infrastructures, building a list of, F-64
- candidate multinesses, 264–265
- candidate patterns, multiple variants of, 44
- candidate requirement pattern, 43
- candidate solutions, F-10
- capacity, patterns for, 47
- capitalization, F-41, F-48
- capped data line in reports, 167
- card number format, 92
- case of alphabetic characters, 87
- case-sensitive identifiers, 98
- cash withdrawal limit, 142
- catch-alls, pervasive requirements as, 27
- categorization of data entities, 119–120
- certification authority (CA) interface, 56
- change history for living entities, 131
- changes, listing in the document history, F-52
- changing passwords, 287, 289
- characters, 87, 88
- character sets, allowing, 87
- charging fees over time, 333
- charts in reports, 164
- check digits, 89
- chronicles, 121, 144–153
  - chronicle entries for reports, 190
- chronicle requirement pattern, 144–153
- circumstances of approval, 319
- classes of users, 285
- classification lists, format of, 23
- classification of requirement patterns, 23, 33–35
- codes of conduct, as standards, 71
- coding standards, compliance with, 76
- cognitive abilities (accessibility), 171, 174, 182
- collaboration over contracts, F-23
- collaborative approach, 8, F-17, F-18, F-19
- color, use of, 180, 183
- commenting, of source code, 83
- commercial domain, 29–30, 325
- commercial requirement pattern, 325–340
- commercial systems
  - availability of, 217
  - detailed requirements specification for, 79
  - importance of transactions in, 134
  - performance types in, 191
- common data for living entities, 131
- common inquiry characteristics, 159, 161
- common requirements, 10, F-26
- communication links, F-56
- communication medium, 60
- communications, disaster recovery, 237
- communications mechanism, interfaces involving, 58
- company coding standards, compliance with, 76
- company financial information, restricting access to, 311

- company standards, 71
  - company Web style guidelines, compliance with, 76
  - company-specific requirement patterns, 43
  - compatible technology, 70
  - completeness
    - of chronicles, 145
    - of requirements, 40
  - complex transactions, 134–135, 138
  - compliance
    - with accessibility standards, 169, 170, 175, 184
    - with data longevity regulations, 107
    - demands for a standard, 73
  - comply-with-standard requirement pattern, 26, 71–79
  - component status inquiry, 159
  - components of a system, 15, F-55
  - compound data item, 95
  - compound data types, 88
  - comprehensive inquiries, 159, 160
  - computer literacy, accessibility and, 171, 174
  - conciseness of identifiers, 97
  - conciseness of requirement IDs, 13, F-37
  - confidentiality of reports, 166
  - configurable authorization, 47, 281, 282, 313–318
  - configurable authorization requirement
    - pattern, 313–318
  - configuration, 120
    - of drivers, 248, 251–252, 253
    - multiness requirements and, 263, 264
  - configuration entities, 138, 141, 142
  - configuration files, 123
  - configuration requirement pattern, 37, 138–144
  - configuration values
    - changing, 139–140, 143, 144, 149
    - content requirements, 141
    - defining, 140
    - examples of, 142
    - flexibility of, 138
    - hard-coded, avoidance of, 138–139, 143
    - representative, 141
    - storing, 140
    - systemwide, 138, 139
    - templates for, 142
  - Considerations for development section, 21, 28–29, 40, 48
  - Considerations for testing section, 21, 29, 40, 48
  - consistency
    - of data, 123
    - of glossary terms, F-49
    - maintaining, 14, F-48
    - promoted by requirement patterns, 19
  - consolidation of fees/taxes, 338
  - constraints, F-54
  - contact details, 96
  - Content section, 21, 24, 47
  - context diagrams
    - advantages of, F-57
    - comparing for old and new systems, F-58
    - defined, F-54–F-55
    - in every requirements specification, 52
    - kinds of information on, 15–16, F-55–F-56
    - notable points about, F-57
    - samples, 52, 53
  - context of data types, 92
  - Context section of a requirements specification, 15–17, F-30, F-53–F-68
  - continuity of IDs, 99, 101
  - contradictions, resolving, F-43
  - core database, 124
  - core terms in a glossary, F-48
  - corporate restructuring, 245
  - cosmetic script, 181
  - cost-benefit analysis, 45
  - countries
    - differences between, 258–260
    - installing systems in different, 255
    - ISO 3166 standard, 76
  - CPU cycle rate, throughput targets and, 207
  - cross-instance access, 270
  - cross-instance conversion, 270
  - CRUD operations, 154
  - cultural differences, unparochialness and, 259
  - currency details
    - configuration values for, 142
    - ISO 4, 217
    - multiness in, 267
    - number types for, 87
    - unparochialness and, 258
  - customers
    - putting in control, F-5
    - using requirements, F-3
  - customer agreement, F-26
  - customer capacity, 216
  - customer details, storing, 130
  - customer number, 100
  - customer orders, archiving of, 113
  - customer password format requirement, 288
  - customer scalability, 243
  - customer service, accessible, 184
  - customer support for multiness instances, 268
- D**
- data. *See also* information
    - backdated, 109
    - backing up, 123
    - configurable authorization to, 314
    - derived, 121
    - disaster recovery, 237
    - historic, 121
    - restricting access to, 308, 310–311
    - segregated, 261–262
    - unaddable, 262
  - data access, recording unsuccessful, 148

- data archiving requirement pattern, 110–117
- data consistency, 123
- data description for archiving, 112
- data display, types of, 86
- data durability, 123
- data entities. *See also* living entities
  - categorization of, 119–120
  - naming, 122, 130
- data entity domain, 29
- data entity requirement pattern, 119–154
- data entry function, testing user registration as, 295
- data format error response time, 200
- data husbandry, 85
- data longevity requirement pattern, 107–110
- data loss, 123, 124
- data modification, archiving and, 115
- data protection, 123, 287, 290–292
- data recovery, 123, 124
- data storage, 107, 108. *See also* data archiving
- data stores, 122, 126–127, F-56
- data structure, 136
- data structure requirement pattern, 94–96
- data type requirement pattern, 86–94
- data types
  - in calculation formulas, 103
  - for different organizations, 260
- data warehouse, 164
- database
  - ACID properties of, 123, 154
  - backing up of, 110
  - CRUD operations, 154
  - estimating disk space needed for, 216
  - recording events in, 151
  - specifying a widely used, 68
  - as storage mechanism, 120, 154
- dates, 88, 89–90
  - of each version released, F-52
  - event, 151
  - report, 166
  - systemwide aspects of, 93
  - unparochialness and, 259
- date display formats, 90
- day one considerations for user
  - registration, 287, 294
- Daylight Saving Time, 90
- de-authentication of users, 298, 299–300
- default language of a system, 274
- default multitenancy instances, 268
- definitions
  - in a glossary, 14, F-44, F-45, F-46
  - of requirements. *See* requirement definitions
- delays, warning users about, 203
- deliberate shutdowns, 231
- delivery mechanisms for reports, 190
- denial-by-default rule, 305–306, 308, 310
- dependencies, 16, 45, F-59
- de-registration of users, 287, 290
- derived data, 121
- descriptive placeholders, 46
- design patterns, 20, 21, 42
- designers, constraining, F-9
- destination of archived data, 112–113
- detailed requirements, 4, F-3, F-27
- details in a requirement definition, F-42
- developers
  - constraining, F-9
  - guidance to, 47, 48
  - using requirements, F-3
  - writing in the language of, 28–29
- development
  - approaches, 5–6, F-6
  - facilities, 69
  - splitting between multiple teams, 54
  - technology, 66, 70
- devolution of information, 135
- dexterity, accessibility and, 171, 174, 182
- dialects, 259, 272
- diminishing returns, availability increases and, 220
- disabilities, people with, 168, 169
- disaster recovery systems, 237
- disclaimers
  - regarding the glossary, F-44
  - stating, 13, F-33
- discounts, specifying, 334
- Discussion section in requirement pattern, 21, 24
- disk space needed, estimating, 216
- display format, 91, 99
- display of data, 86
- distinct user experiences, 261, 267–269, 271
- distinctive requirement IDs, 13, F-37
- distributed office scalability, 243
- divertive performance patterns, 192
- divertive requirement pattern, 32, 36
- Document history section of requirements
  - specification, 15, F-51–F-53
- document library scalability, 243
- document management system, 154
- document purpose, F-30
- Document purpose section, 12–13, F-32–F-33
- document templates, 123
- documentary efficiency, 9
- documentation
  - accessibility of, 175, 184
  - of interfaces, 57, 61
  - purpose of, F-24
  - unparochial, 258
- documentation requirement pattern, 81–84
- domains
  - assigning requirement patterns to, 29–31
  - compared to requirement pattern groups, 32
  - creating new, 44–45
  - infrastructures in, 30
  - as a source of candidate infrastructures, F-64
  - specifying, 22

- domain expert, F-15
- domain specification, 37
- downtime
  - business impact of, 221, 223–224
  - causes of, 221, 225
  - levels of, 218
  - reducing, 225–235
  - scheduled, 219
  - tolerated, 222
- draft requirements specification, writing, 8, F-18–F-19
- drivers
  - configuration information for, 248, 249–250
  - extendability requirements and, 247, 249
  - installation requirements for, 251–252, 253
  - switching between, 253
  - system requirements for, 250–251
  - third-party, development requirements, 252
  - upgrading, 253
- driving entity, 215–216
- durability of data, 123
- duration start trigger, 108
- dynamic capacity, 47
- dynamic capacity requirement pattern, 212–215
- dynamic users, counting, 213
- dynamic Web functions, 222

## E

- effort estimation, F-26
- ejecting users, 214
- electronic communication medium for approval, 320
- email, 91, 123
- emergency extended access, 235
- emergency remote access, 235
- employee ID, 100
- employee roles, 260
- end of report line, 167
- end-to-end response time, 196–197
- English as a second language, accessibility and, 171
- entity life history diagram. *See* state transition diagram
- errors, 148, 233, 236. *See also* failures
- error messages, 83, 236, 237
- essence of a requirement, F-42
- European Union, data protection provisions, 290
- events, 144–145, 148–149, 333
- event date, recording, 151
- event record, 151, 152
- event severity levels. *See* severity level
- evolution of requirements, F-38
- example requirements, 45–46
- examples in a requirement definition, F-42
- Examples section in a requirement pattern, 21, 26
- excluded entities, determining, 216
- exclusions, 16, F-61
- executive summaries, avoiding acronyms in, F-48
- explanatory text, displaying, 181
- explicit interaction requirements, 62

- extendability, 239
  - of fees, 336
  - of an interface, 57
  - multiness requirements and, 262
  - requirement pattern, 52, 73, 246–253
  - types of, 264
- extendability requirement pattern, 52, 73, 246–253
- extended access, 234–235
- extends requirement pattern relationship, 32, 33
- external dependencies, F-54
- external interfaces, standards affecting, 72
- external requirements specification, 79
- external sources, allocating IDs from, 99
- external systems, 127, 260
- external users, 212
- extra requirements, 27, 47
- Extra requirements section
  - considering during requirements definition, 39
  - example requirements in, 28
  - in a requirement pattern, 21, 26–28
  - writing, 47–48
- extreme programming (XP), 9, F-6, F-24
- extreme requirements process, 9–10, F-24–F-26

## F

- failures, 231–237
- faithful nature of chronicles, 145
- federation of systems, 54
- fee rates, determinants of, 333
- feedback on requirements specification, dealing with, F-21
- fee/tax requirement pattern, 330–340
- financial data, storage of, 109
- financial flows, diagramming, 331–332
- financial transaction, 137–138, 158
- financial year, 259
- flat files, 123, 124
- flexibility, 135
  - of configuration values, 138
  - described, 239
  - domain, 29–30
  - of identifiers, 98
  - importance of, 239
  - performance requirements and, 240
  - as a source of nonfunctional system aspects, F-70
  - testing, 240
- flexibility requirement pattern, 239–279
- follow-on requirements, 20, 27, 28, 47, 132
- forgotten passwords, handling, 297, 298–299
- formal part of a requirements specification, 6–7, F-11–F-12
- formal statement of a requirement, F-42
- formats
  - for documentation, 82
  - for requirement IDs, F-37

framework, compared to infrastructure, F-64  
 freezing requirement IDs, F-38  
 frequencies for requirement patterns, 23  
 full year, storing dates for, 90  
 functions  
   authorizing access to, 308, 310, 311  
   availability goals related to, 225  
   compared to use cases, F-13  
   controlling access to, 314  
   controlling use of, 305  
   recording, 145  
   testing access to, 318  
   user. *See* user function requirement pattern  
   for viewing chronicles, 152  
 functional areas, 41, F-68–F-69  
 functional area sections in a requirements specification,  
   12, 17–18, F-30, F-68–F-70  
 “Functional” requirement pattern classification, 34–35  
 functional requirements, 4, F-3  
 functional testing of user authentication, 304  
 fundamental domain, 29, 51  
 fundamental requirement patterns, 51–84

**G**

gaps in a domain, 45  
 Gather information requirements process step, 8, F-17  
 generalizations, use cases as, 37  
 general-purpose glossary, writing, F-49  
 geographic distribution of users, 209, 259  
 geographic variations for calculations, 103  
 glossary, 14, F-44–F-49  
 government regulations as standards, 71  
 graceful degrading of response time, 199  
 graphics, 180–181, 188. *See also* images  
 Greenwich Mean Time, 90  
 groupable requirement IDs, 13, F-37  
 guidance, provided by requirement patterns, 19

**H**

hand strain, accessibility and, 171, 174, 182  
 happen time for a transaction, 136  
 hard-coded configuration values, avoidance of,  
   138–139, 143  
 hard-coding language-specific text, banning, 258  
 hardware  
   availability issues related to, 226  
   power of, 193  
   replication of, 231  
   scalability requirements and, 244, 245  
 hardware setup  
   response time and, 199  
   for throughput, 205, 206–207  
 Have specification reviewed step, 8, F-19–F-20  
 hearing, accessibility and, 171, 174, 181–182  
 hexadecimal numbers, 87

high load caveat, response time and, 199  
 high-level requirements, 4, F-3  
 HIPAA standard requirements, 75  
 historic data, 110, 121  
 housekeeping. *See* maintenance

**I**

IBM, accessibility guidelines, 186  
 icons, consistent meanings of, 182  
 identification details for users, 285  
 identifier. *See also* IDs  
   unique for each requirement, 13, F-33  
 identity card issuance, 201  
 ID requirement pattern, 97–101  
 IDs, allocation of, 99, 263  
 IEEE (Institute of Electrical and Electronics Engineers)  
   technical standards, 74  
 IETF (Internet Engineering Task Force), 74  
 images, 123, 180–181, 188. *See also* graphics  
 implementation, 40, F-43  
 implicit interactions, 62  
 inaccessible data, filtering, 311  
 inaccessible functions, hiding or disabling, 306  
 inactive information, removal of, 217  
 inactivity time-out, 301  
 incorrect password entry, recording, 149  
 incremental approach to requirements, 3, 10, F-2,  
   F-6, F-26, F-28  
 independent reviews, approach to, 8, F-19  
 indexes of data archives, 114  
 industry-specific codes of practice, 71  
 industry-specific patterns, 43  
 industry-specific regulations, 71  
 industry-specific standards, 75  
 inflexibility  
   system, 135, 138  
   temporal, 256  
 informal elements in a requirement definition, 13,  
   F-42–F-43  
 informal part of a requirements specification, 6–7,  
   F-11–F-12  
 information. *See also* data  
   backing up, 123  
   changes, 122  
   described, 85  
   devolution of, 135  
   domain, 29  
   entry, multi-part, 122, 124–125  
   flowing in an interaction, 64  
   form of, 86  
   gathering, F-17  
   gathering from people, 8  
   infrastructure, 154  
   inquiries for, 123  
   integrity, 122, 123

- restricting access to, 305
- retrieval mechanisms, 154
- standards, 71
- storage, 120, 124, 154
- types of, 123
- information requirement pattern, 85–117
- infrastructures
  - defined, 17, F-63
  - disentangling from business functions, F-67
  - extendability requirements and, 250
  - fewer distinct actors, F-69
  - identifying, 30
  - improving the boundary of each, F-67
  - for information storage, 120, 154
  - for a new domain, 45
  - numbered list of, F-65
  - overviews, 30, 31
  - process for identifying and specifying, F-64–F-68
  - relationships with domains and requirement patterns, 30, 31
  - reporting, 155, 189–190
  - in a requirement pattern diagram, 33
  - requirements, 27, F-67, F-68
  - requiring technical analysis, F-67
  - treating a business rule product as, 38
  - user interface, 155, 187–189
  - using from other domains, 30
- Infrastructures section in a requirements specification, 17, F-54, F-63–F-68
- inheritance relationship between requirement patterns, 32, 33
- initial customer capacity, 216
- input, 171. *See also* throughput
- input interpretation by multiness instances, 268
- inquiry
  - automatic refreshing of, 157, 159, 177, 188
  - dynamic use, 214
  - for information, 123
  - for living entities, 131
  - multiness requirements and, 263
  - from offline storage, 117
  - versus report, 162
  - response time, 200
  - scalability requirements and, 245
  - static capacity, 217
- inquiry requirement pattern, 156–161
- inspection of a requirements specification, F-19
- installability requirement pattern, 274–279
- installation requirements for drivers, 251–252
- installing
  - driver software, 251–252
  - scalability requirements and, 245
- instances, giving users access to, 270
- Institute of Electrical and Electronics Engineers (IEEE)
  - technical standards, 74
- integers, data type, 87
- integrity, 122, 123, 126, 140
- intent of a requirement, F-42
- interactions
  - across an interface, 57, 62, 63
  - describing an activity across the interface, 52
  - with inquiries, 157
  - specifying types of, 63, 64
- interactive documentation, 84
- interactive tutorial, 83–84
- interest calculations, 104–106
- interfaces
  - defining, 15–16, 54, 55, F-55
  - describing within other systems, F-57
  - development considerations, 62
  - documentation of, 61
  - examples, 56, F-58
  - indeterminate number of, 53
  - influencing the design of, 63
  - of an infrastructure, 31
  - interactions in requirements, 63
  - to multiple systems for the same purpose, 52, 54
  - ownership of, 63
  - passport control for, 58
  - positioning in a context diagram, F-57
  - security requirements for, 60–61
  - specifying between systems, 52
  - standards defining, 79
  - switching to new versions of, 59
  - templates, 56, 64
  - testing, 62, 65
  - upgrading, 59–61
  - as weak links, 52
- interface adapters, 53–54
- Interface Developer's Guide, writing, 61
- interface ID, 55, 64
- interface name, 55, 64
- interface ownership, 54–55
- interface resilience requirements, 58
- internal interactions, 52
- internal users, 212, F-57
- international dimension, unparochialness and, 258–260
- International Organization for Standardization (ISO), 74
- International Standard Book Number (ISBN), F-51
- Internet Engineering Task Force (IETF), 74
- Internet Explorer Web browser, 68
- Internet-based retail system, F-35–F-36
- inter-system interaction requirement pattern, 47, 62–65
- inter-system interface requirement pattern, 51–62
- inter-system interfaces, 15–16, 65, 206, F-55
- interview approach to gathering information, F-17, F-18
- introduction of the glossary, F-44
- Introduction section of a requirements specification, 12–15, F-30, F-31–F-53
- invalid interactions, testing for an interface, 65

invisible ID scheme, 101  
 invocation requirements  
   information infrastructure, 154  
   for an infrastructure, F-67, F-68  
   in an infrastructure overview, 31  
   reporting infrastructure, 189–190  
   user interface infrastructure, 187–188  
 “is-a-kind-of” relationship, 37  
 ISBN (International Standard Book Number), F-51  
 ISO (International Organization for Standardization), 74  
 ISO 639 standard for natural languages, 76, 272  
 ISO 3166 standard for countries and regions, 76  
 ISO 4217 standard for currencies, 76, 142  
 isolated changes, 123  
 iterative development, F-6

**J**

Java programming language, 69  
 Javadoc, 81  
 Joint Application Development (JAD), F-17  
 Joint Requirements Planning (JRP), F-17  
 justification  
   for a requirement’s existence, F-42–F-43  
   for the requirement’s form, F-42  
   stating for every exclusion, F-60

**K**

Key Business Entities section of a requirements  
 specification, 16–17, F-54, F-61–F-63  
 keyboards, accessibility and, 171, 174, 182

**L**

lack of confidence in requirements, F-4  
 languages  
   accessibility and, 171  
   displaying user interface in more than one  
   alternative, 272  
   of documentation, 82  
   ISO 639, 76, 272  
   multiness requirements and, 262  
   unparochialness and, 258–259  
 language-specific resources, substituting for missing, 274  
 language-specific text, banning hard-coding of, 258  
 lapsed time period, 336–337  
 layouts, consistent for reports, 166  
 levels of registration, 286  
 liability to pay a fee, 333  
 life support mechanisms for a system, 17, F-63  
 life-critical systems, 217  
 lifespan of a data entity, 120, 129  
 list of allowed values, 88  
 living entity  
   event in life of, 120, 133  
   lifespan of, 120, 129  
   multiness of, 264

  naming, 122, 130  
   testing considerations, 133, 144  
 living entity requirement pattern, 129–133  
 load on a system, reducing, 214, 215, 245  
 loan approval decision rule ID, 100  
 local context diagram, F-56  
 local currency, configuration values for, 142  
 local scope, F-46  
 locale, 272  
 logging in, 295  
 logging messages, 58  
 logging out, 298, 299–300  
 logical data types, 86, 264–265  
 logical pattern for requirement IDs, F-37  
 logical remove only, 131  
 logical view of a system, F-56  
 login function, coding, 304  
 logs, 121, 123  
 longevity of transactions, 136  
 loopholes in access control, 307  
 lowest-priority requirements, including, F-40

**M**

machines, distributing loads across, 214, 215, 245  
 machine shutdown, user access and, 230  
 maintenance, 190, 225, 228–229  
 Major assumptions section in a requirements  
 specification, 16, F-54, F-59–F-60  
 Major exclusions section in a requirements  
 specification, 16, F-54, F-60–F-61  
 Major nonfunctional capabilities section in a  
 requirements specification, 18, F-30, F-70–F-71  
 manifestations  
   of a domain specification, 37  
   of an infrastructure, F-65  
   of a requirement pattern, 22, 42  
 manual dexterity, accessibility and, 171, 174, 182  
 marketing campaigns, multiness in, 267  
 masking wait times, 203  
 maximum acceptable response time, 198–199  
 meaningfulness of identifiers, 97  
 medium for documentation, 82  
 memorability of identifiers, 97  
 memory leaks, testing for, 238  
 messages, recording, 58, 59  
 metrics, attaching to a requirement pattern, 40  
 Microsoft Windows operating system, 69  
 missing resources, substituting for, 268  
 mistakes, correcting for fees/taxes, 339  
 modification rules for transactions, 137–138  
 modularity, extendability and, 247  
 monetary amounts, 87  
 month basis for interest calculation, 105  
 morsel sizes for different development approaches,  
   5–6, F-6  
 multibyte characters, allowing, 87

- multilevel numbers as version numbers, F-52
- multilingual organization, 262
- multilingual requirement pattern, 272–279
- multi-locale, 272
- multimedia resources, 123, 176
- multiness requirement pattern, 261–272
- multi-organization unit, 262, 328
- multi-organization unit requirement pattern, 325–330
- multi-part information entry, 122, 124–125
- multiple data stores, 122, 126–127
- multiple instances of a particular interface, 52
- multiple parts, IDs comprising, 97, 99
- multiple sites, 254, 255
- multi-release deliveries, F-40

## N

- names, data structure for personal, 96
- naming
  - configuration values, 141
  - data entities, 122, 130
  - data types, 90
  - identifiers, 98
  - inquiries, 156
  - transactions, 135
- natural availability level, 220
- natural languages, ISO 639 standard, 76, 272
- navigation among inquiries, 157, 160
- negative numbers, 87
- new users, processes for, 287, 293–294
- nonbinding part. *See* informal part
- nonfunctional requirements, 4, 69, F-3, F-40
- Nonfunctional requirements section of requirements specification, F-71
- noninterference archiving, 110
- non-repudiation, 282
- normal opening times, 217
- notification methods
  - customer preferences for, 133
  - for driver requirements, 251
  - for report recipients, 190
  - for system failures, 233, 234
- novice users, accessibility and, 171, 174
- numeric display formats, unparochialness and, 258

## O

- Object Management Group (OMG), 74
- objectives of requirements, F-36
- objects of a system, F-10
- occurrence. *See* events
- offline transactions, 117
- off-the-shelf products, implementing infrastructures, F-68
- OMG (Object Management Group), 74
- online data storage, 107, 108
- online documentation, testing, 84
- open issues in a requirements specification, F-54
- operating systems, 69, 185

- operational rules, 308, 312
- opportunistic approach to capturing requirement patterns, 43–44
- optional parts of a requirement template, 25
- Oracle database, 68
- order events, storing, 147
- order ID, 100
- organization units, 264, 326, 327
- organizational construct, 325
- organizational structure, 260, 327
- organizations, system building for different, 255, 260
- origin
  - of a term, F-47
  - of transactions, 206
- outages, 225, 232–237. *See also* failures
- output, 171. *See also* throughput
- outsourcing, tailoring documentation for, 84
- over time, charging a fee, 333
- overall processing state, specifying, 143
- overflows, highlighting on reports, 167
- owner
  - of a glossary, F-49
  - of transactions, 136
- owner entity name, 98
- ownership shyness, F-50

## P

- page count on a report, 164, 167
- page throw levels on a report, 165
- paper type for a report, 167
- parameter values, 138, 195
- parent entity, 130
- parochial data type definition, 89
- parochialness, 256, 257. *See also* unparochialness
- participants in the requirements process, F-15
- passive attack on an interface, 60–61
- passive reviewers of a requirements specification, F-19
- passwords
  - changing, 287, 289
  - criteria for acceptable, 288–289
  - generating initial, 287
  - system-generated, 293
- password entry, recording incorrect, 149
- password format, criteria for, 287, 288–289
- password guesser utility, 289
- pattern author, 23
- “Pattern classifications” in a requirement pattern, 23
- pattern manifestation, 22
- pattern name, 21
- patterns, 20, 43. *See also* requirement patterns
- peak customer capacity, 213
- peak minute, calculating throughput for, 208
- peak period, 213
- pending actions, 128, 129
- people over processes, agile exhortation, F-23
- percentages, as data type 87, 218

performance

- common issues with, 192–195
- defined, 191–192
- degrading of, 225, 244
- flexibility and, 240
- as a source of nonfunctional system aspects, F-70
- transactions and, 137

performance domain, 29–30, 191–238

performance requirement patterns, 191–238

performance targets, 192, 193–194

PERL scripts, 69

permissions. *See* privileges

personal contact details, data structure for, 96

personal name details, data structure for, 96

“Pervasive” requirement pattern classification, 35

pervasive requirements

- alerting readers to the presence of, 28
- defined, 20, 27
- grouping related, 27
- for inquiries, 159, 161
- for living entities, 131–132
- for reports, 166–167, 168
- writing, 47

post codes, unparochialness and, 259

postmortem, 42

power users, accessibility concerns and, 171, 174

preciseness of requirement patterns, 20

precision

- of calculations, 103
- of a glossary definition, 14, F-46

preemptive corrections, 225

preferences of users, 285

Prepare step in requirements process, 7, F-16–F-17

presentation unit, 176

preset thresholds, passing, 145

previous screen, returning to, 125

pricing changes, 129

primary programming language, 69

primary reviewers of requirements specification, F-19

principles, regarding requirements specification, 6–7, F-9

priorities, F-40–F-41

prioritizing requirements, 14

priority of a requirement, F-34, F-36, F-39–F-41, F-43

privacy requirements for reports, 166

privileges, 305

privilege types inquiry, 317

problems

- distinguishing from solutions, 9
- specifying, 6, F-9

procedures manual, documentation requirement, 83

processing abilities, accessibility concerns and, 171, 174, 182

processing load, 215

product mailing list, customers joining, 133

product restarts, need for, 229

production line, non-interference with, 139–140

programming language, 69, 188

progress bar, displaying, 204

project manager, F-3, F-15

proof of existence archiving, 111

prospective installations, scope of, 255

protection of data, 123

prototypes, developing, F-14

pseudo-localization, testing, 271

punctuation, unparochialness and, 259

## Q

quality

- defining, 192
- shutdowns and, 231
- as a source of nonfunctional system aspects, F-70
- upgrade frequency and, 229

quality standards, 71, F-49

quantity limitations, removing, 241

## R

range, expressing frequency as, 23

rationale of a requirement, F-42

readability of a requirement, F-34

rearchiving, 114

recent orders inquiry, 159

recorded event inquiry, 152

recovery of data, 123

refactoring, 239

reference numbering scheme, F-49

referenced requirements, 80–81

referenced specification, 79, 80

references

- for calculation formulas, 103
- details required, F-50–F-51
- to other glossaries, F-44
- withholding, F-49

References section of a requirements specification, 14–15, F-49–F-51

Refers to requirement pattern relationship, 32, 33

refer-to-requirements requirement pattern, 79–81

refinement requirements, 35–36

reflective monitoring of throughput, 210

refreshing user interface, 157, 159, 177, 188

regions, installing systems in varied geographic, 255

registering users, 284

registration, process of, 285

regulations

- accessibility, 169, 170, 184
- data longevity, 107, 109

rejection, during approval process, 320

“Related patterns” in section in requirement pattern, 22

relationships between requirements, F-43

relative volumes for throughput, 205, 206

releases, as independent of priority levels, F-40

relevance of glossary terms, 14, F-45

reliability

- of chronicles, 145
- system, 231

- reloading of archived data, 116–117
- remote access
  - facilities, 234–235
  - restricting access to, 311
- removing requirements, F-38
- renumbering requirements, F-38
- reorganizing organizations, 329
- repetition, avoiding, 7, F-48
- repetitive strain injury, accessibility concerns
  - and, 171, 174
- replay attack on an inter-system interface, 305
- replication of hardware, 231
- reports
  - access control, 190
  - availability, 233
  - content requirements, 164–165, 190
  - defined, 189
  - delivery mechanisms for, 190
  - designing, 162, 190
  - dynamic use, 214
  - format considerations, 167, 190
  - multiness requirements and, 263
  - purging, 190
  - recipients of, 163, 166, 190
  - scalability requirements and, 245
  - scheduling of, 190
  - size limits on, 244
  - static capacity, 217
- report design changes, recording, 149
- report design tool, 190
- report fee requirement, 335
- report instance, 189
- report requirement pattern, 161–168
- report run request, recording, 149
- reporting infrastructure, 31, 155, 189–190
- representative configuration values, 141
- Request For Comment (RFC) standards, 74
- requirements
  - arguments against specifying, F-4
  - audiences for, F-3
  - case for specifying, F-4
  - compared to requirement patterns, 21
  - defined, 4, F-2, F-8
  - grouping by functional area, 41
  - identifying, 9, F-24
  - impact of an agile outlook on, F-23
  - indications of pattern use, 40
  - introducing a variety into, 41
  - items given for each, F-36–F-43
  - lack of confidence in, F-4
  - levels of detail for, 4, F-3
  - nonfunctional, F-40
  - not specifying for an infrastructure, F-66
  - overall approaches to, 3, F-1
  - presenting in a specification, F-34
  - prioritizing, F-39
  - proportion covered by patterns, 41
  - putting customer in control, F-5
  - referencing an infrastructure, F-65
  - referencing applicable, 80
  - refinements of the main, 35–36
  - reflecting business rules, 38
  - relationships, F-43
  - removing, F-38
  - requirement patterns when defining, 39
  - responsibility for overall organization of, F-27
  - specifying, 4–5, F-3, F-28, F-64
  - treating constraints as, F-54
  - types of, 19
  - writing down during extreme programming, 10, F-26
- requirement definitions, 13, F-33, F-35–F-36, F-42–F-43
- Requirement format section, 13–14, F-33–F-43
- requirement IDs, 13, F-33
  - example of, F-35
  - grouping, F-37
  - qualities of, F-36–F-43
- requirement pattern classifications, 34
- requirement pattern domains. *See* domains
- requirement pattern groups, 31–32, 33
- requirement pattern use cases, 37–38
- requirement priority. *See* priority of a requirement
- requirement summary. *See* summary description
- requirement template, 24, 46–47
- requirement patterns. *See also specific names of requirement patterns*
  - access control, 281
  - applicability of, 24
  - in association with extreme programming, 10, F-26
  - benefits of using, 19, 40–41
  - classifications of, 33–35
  - commercial, 325
  - contents of, 21–29
  - data entity, 119–154
  - defined, 19
  - diversity of approaches to, 36–37
  - drawbacks of, 41
  - and extreme programming, F-25–F-26
  - finding candidate, 43–44
  - flexibility, 239–279
  - fundamental, 51–84
  - information, 85–117
  - misapplying, 41
  - naming, 21
  - performance, 191–238
  - refining, 42
  - relationships between, 32–38
  - relationships with domains and infrastructures, 30, 31
  - during requirement definition, 39
  - reviewing, 48
  - sections of, 21
  - sources of, 20
  - specifying better in the future, 44
  - tailoring, 41–42

requirement patterns, *continued*

use cases for, 37–38

user function, 155–190

using after the fact, 40

using and producing, 39–48

using to consider completeness, 40

writing, 21, 42–48

requirements approaches, diversity of, 36

Requirements Management Tools Survey, F-2

requirements process, 7–8, F-12–F-14, F-15

requirements specifications

contents of, 11–18, F-29–F-71

defined, 4, F-2

essential items of information in, F-33

example for infrastructures, F-66

example text for, F-35–F-36

formal and informal parts of, F-11–F-12

for an infrastructure, F-65, F-67

introduction of, F-33

language, 22

organizing, 11, F-29–F-30

reviewing, 8, 40, F-19

as a source of candidate infrastructures, F-64–F-65

suggested structure for, 11, F-29–F-30

requirements-take-too-long argument, F-4

resilience, 57, 58

resource checker utility, 269

resources, managing multiple sets of, 269

response time

evaluating, 203–204

scalability and, 242

response time requirement pattern, 195–204

responsiveness over plan, agile exhortation, F-23

restatement of a formal requirement, F-42

restoring data, 123

restructuring, corporate, 245

revenue model, 331–332

review cycles, F-20–F-21

reviewers, F-19, F-20

reviews, feedback from, 48

Revise after review step, 8, F-20–F-21

RFC (Request For Comment) standards, 74

roles, assigning to users, 285

rolling window, 188

rounding, 103

rule IDs, 100

run time, infrastructures used during, F-63

## S

sales tax, 335

sales tax rate, 129

Sarbanes-Oxley Act (SOX), 75

scalability

dynamic capacity and, 212

of an interface, 57

specifying, 205

scalability requirement pattern, 241–246

scaling out, 242

scaling up, 242

scheduling of reports, 190

scope

defined, F-7

of the glossary, F-44

of a glossary definition, F-46

of a glossary term, F-49

of requirement ID uniqueness, F-36–F-37

writing, F-8

scope boundary, 15, F-55, F-56

scope document, F-31–F-32

Scope section of a requirements specification, 15–16,

F-53, F-54–F-58

screen

refreshing, 177, 188

returning to previous, 125

screen focus, accessibility requirements, 179

screen size, accessibility requirements, 179

secondary reviewers, F-19

secret information, identifying users, 298

Section 508 of Rehabilitation Act, 170, 175, 186

sections in a requirement pattern, 21

security

against deliberate shutdowns, 231

archiving for, 110

extendability requirements and, 249, 252

of an interface, 57

outlawing bad practices, 293

requirements for good practices, 287, 292–293

requirements for interfaces, 60–61

requirements for reports, 166

as a source of nonfunctional system aspects, F-70

security breaches, helping users to spot, 298, 301–302

security procedures manual, 83

security risks during installation, 276

security violations, recording, 149

segregated data, 261–262, 269–270, 271

selection criteria

for inquiries, 157, 161

for recorded events, 152

semantic requirements, 178, 186

sender, verifying the identity of, 58, 59

sensitive data, recording access to, 148

sensitive information, 292

sensors, data recorded by, 123

separate requirements specification for a standard,

77–78

separator characters, 88

sequence numbers in templates, 25

sequential numbers, allocating, 99, 101

sequential requirement IDs, 13, F-37

sets of requirements, 20

severity level, 145, 150, 152–153

shutdowns, 225, 232. *See also* failures

“sign off” of requirements, F-21

- signed numbers, data type, 87
- simple data type, IDs as, 97
- simplicity of identifiers, 98
- simplification of transactions, 135
- simultaneous customer capacity, 213
- single requirement for standards compliance, 77
- single site, installing system in, 255
- sites, scaling number of, 245
- sizing model, 195, 211, 216
- skeleton pattern, 45
- software
  - for archiving, 116
  - availability issues related to, 226
  - changing at both ends of an interface, 59, 60
  - data types represented in, 94
  - for drivers, 251–252
  - extending using, 246
  - over documentation, agile exhortation, 8, F-23
  - performance targets for, 192–193
  - recording changes, 149
  - scalability requirements and, 245
  - for system integrity monitoring, 236
  - writing to support documentation, 83
- software download
  - accessibility concerns, 177
  - need for, 188
- solutions
  - distinguishing problems from, 9
  - formulating based on requirements, F-8
  - not specifying in requirements, 6, F-9
- sort order for IDs, 99
- sort sequence for inquiries, 157
- sound, accessibility concerns and, 171, 174, 181–182
- sound alerts, 182
- sounds, 123
- source code
  - commenting, 84
  - documentation, 83, 84
  - for driver software, 252
- SOX (Sarbanes-Oxley Act), 75
- specific authorization, 281, 282
- specific authorization requirement pattern, 47, 308–313
- specific needs for accessibility, 168, 169, 172, 174–175, 178–182
- specific privileges, 305
- specifications, purpose of, F-2
- spelling notes in a glossary, F-47
- splitting a requirement pattern in two, 46–47
- spreadsheets of mind-boggling complexity, 332
- stale user sessions, 213, 215
- standards
  - categories of, 71–72
  - complying with multiple versions, 72–73
  - complying with parts of, 73
  - contents of requirements mandating, 73–74
  - defined, 71
  - defining an interface, 55
  - development considerations, 78–79
  - for documentation, 82
  - examples of, 74–77
  - location of, 74
  - making accessible, 78
  - multiple different for the same thing, 72–73
  - names of, 73
  - purposes of, 73
  - as a source of nonfunctional system aspects, F-70
  - specifying that a system comply with, 71
  - technology and, 66
  - testing considerations, 79
  - versions of, 73
- standard patterns, agreement on, 36
- standard-related requirements, prioritizing, 78
- state transition diagram, 17, F-62–F-63
- static capacity requirement pattern, 47, 215–217
- statistical functions for viewing chronicles, 152
- statistics on requirements for a system, 33–34
- stopwatch, recording response times, 203
- storage medium for archiving, 112
- strategic stratosphere, report design and, 163
- “strikethrough” text, F-38
- style sheet, 177
- subcalculations, in a calculation formula, 103, 106
- subtotals on reports, 166
- subtransactions, 134
- suffix, adding to a previous requirement ID, F-38
- suggestions, resolving conflicting, F-21
- suitability conditions, 256
- summary description, 14, F-34, F-36, F-41
- support, accessible, 184
- surreptitious unavailability, 221, 224–225, 238
- switching time transaction, 200
- systems
  - accommodating old and new interface versions, 59, 60
  - activities in building new, 5, F-6
  - availability window of, 208, 219
  - defined, 4, F-1
  - describing purpose of, F-31
  - difficulty of migrating from old, F-58
  - driver type requirements for, 250–251
  - at each end of an interface, 55
  - human being as part of, 171
  - load on, reducing, 214, 215, 245
  - monitoring response times within, 202–203
  - popular times for using, 208
  - reliability of, 231
  - replacing existing, F-58
  - specifying, 6, F-10
- system activity inquiry, 143
- system building, specificity of, 254
- system capacity. *See* dynamic capacity; static capacity
- system clock, data longevity and, 109
- system configuration. *See* configuration

system context diagram. *See* context diagrams  
 system defects, testing for, 238  
 system design, documentation requirement, 83  
 system designer, F-15  
 system events, recording significant, 148  
 system expansion, allowing for. *See* scalability  
 system failures, minimizing, 225  
 system flexibility. *See* flexibility  
 system glossary. *See* glossary  
 system gone live state, 143  
 system information, viewing of, 160  
 system monitor, requirements for, 234  
 system operation efforts, non-growth in, 244  
 system performance, 193  
 system processing state, 143  
 System Purpose section in requirements specification, 12, F-31–F-32  
 system resources, freeing up, 214  
 system threshold, passing of defined, 148  
 system unavailable page, 224  
 systematic approach to capturing patterns, 43  
 system-generated passwords, no access via, 293  
 systemwide configuration values, 138, 139

## T

table, distinguishing a list of requirements, F-34, F-35  
 taxes, 330–331  
 “TBD” paragraphs, removing, F-54  
 teams, splitting development between multiple, 54  
 technical data types, 86  
 technical standards, 71, 72, 76–77  
 technology  
   constraints on the user interface, 188  
   defined, 65  
   description of, 66  
   development considerations, 70  
   examples of, 68–69  
   for an interface, 55  
   specifying to build or run a system, 65  
   usage of, 67  
   used in development, 66  
   for user interface infrastructure, 187  
   versions of, 67  
   working with a range of, 67  
   working with multiple, 70  
 technology requirement pattern, 65–70  
 telephone numbers, data type 92, 259  
 templates section of a requirement pattern, 21, 24–25  
 temporal parochialness, 256, 257  
 temporary dispensation from compliance to  
   a standard, 73  
 terminology, variation in, 260  
 terms  
   choosing new, F-45–F-46  
   establishing the meaning of, F-44  
   in a glossary, 14, F-45  
   listing alphabetically in a glossary, F-48  
   of local scope, F-46  
 test system for external developers interface software, 61  
 testers, 47, 48, F-15  
 testing  
   acceptance, 9, F-25  
   accessibility, 186  
   availability, 238  
   calculation formula, 106–107  
   chronicle, 153  
   configuration, 144  
   considerations for, 29  
   considerations for interfaces, 65  
   data archiving, 117  
   data longevity, 109–110  
   data structure, 96  
   data types, 94  
   of documentation, 84  
   dynamic capacity, 215  
   extendability, 253  
   flexibility, 240  
   ID, 101  
   inquiries, 161  
   of interfaces, 62  
   living entities, 133, 144  
   report, 167–168  
   response time, 204  
   scalability, 246  
   static capacity, 217  
   technology requirements demands on, 70  
   throughput, 211  
   transactions, 138  
   unparochialness, 260–261  
   using patterns during, 40  
 testing regime, F-27  
 tests  
   exclusions and, F-60  
   using requirements, F-3  
 text  
   display, accessibility requirements, 179, 184  
   equivalent of cosmetic script, 181  
   not burying in code, 269  
 theme  
   introduction explaining, 30  
   for a new domain, 44  
   of patterns in a domain, 29, 32  
 third-party interface development, 57, 61  
 third-party system, interfacing to, 54  
 threshold rates for fees, 333  
 throughput of an interface, 57  
 throughput requirement pattern, 204–211  
 throughput targets  
   CPU cycle rate and, 207  
   for inter-system interface, 206  
   justification for, 209  
   purpose of, 205  
   selecting, 205, 206

- timeframe for, 208, 209
- transactions as, 206
- tiered rates for fees, 333
- time, 89–90
  - of system use, 208
  - systemwide aspects of, 93
- time limits for data longevity, 108
- time periods, measuring, 337
- time, response. *See* response time
- time to detect (availability), 232–234
- time to fix (availability), 232, 235–237
- time to react (availability), 232, 234–235
- time zones, 90, 93, 133, 259
- timed changes, 127–128, 129
- timed responses, control of, 177
- timeless requirements specification, 6, F-11, F-56
- timestamps, 88, 90, 92, 153
- title for each reference, F-50
- “To Be Done” (TBD) paragraph format, F-18
- top-level sections of requirements specification, F-29–F-30
- totaling levels for reports, 165, 166
- traceability, ID facilitating, F-35
- traditional approach, 3, F-6
  - to specifying requirements, 7–8, F-1–F-2, F-12–F-22
- traditional documentation, 84
- traffic verification and recording, 57, 58–59, 60–61
- training for installation, 277
- transactions
  - determining origin of, 206
  - integrity of, 126
  - multiness of, 264
  - requirement pattern for, 133–138
  - restoring offline, 117
  - steps in, 125
  - switching time, 200
    - as throughput target, 206
- transaction data scalability, 244
- transaction fee requirement, 334
- transaction monitor, 154
- transaction number, 125
- transaction requirement pattern, 133–138
- transference of restrictions, 308, 312
- transitions, state, 17, F-62
- troubleshooting installation, 276

## U

- U.K. Data Protection Act (1998), 75, 292
- U.K. Disability Discrimination Act, 170
- UML (Unified Modeling Language) standard, 74
- unaddable amounts, not sorting on, 270
- unaddable data, 262, 270, 271
- unapproved actions, storing, 323
- unavailability, surreptitious, 221, 224–225, 238
- unavailability window, 219
- undecipherable form of passwords, 292
- Unified Modeling Language (UML) standard, 74
- uninstalling, 277, 279
- unique identifiers, 98–99
  - for requirements, 13, F-33, F-36–F-37
  - scheme for assigning, 97
- unique interface IDs, 52
- unit IDs for organizations, 329
- units, associated with values, 87
- Universal Time Co-ordinated (UTC), 90, 92
- unknown information, 160
- unpaid fees, tracking, 336
- unparochialness
  - content requirements, 256
  - examples of, 257
  - extra requirements for, 257–260
  - specifying, 254–255
  - templates for, 256
  - testing, 260–261
- unparochialness requirement pattern, 254–261
- unsigned numbers, data type, 87
- upgrade from any previous version requirement, 277
- upgrade instructions, documentation requirement, 83
- upgrade requirements for an interface, 59–61
- upgrades
  - as downtime, 225
  - duration of, 230
  - frequency of, 229–230
  - preparation for, 230
  - scalability requirements and, 245
  - testing, 278
  - uninstalling, 277
- upgrading, 276
  - of an interface, 57
  - specifying requirements for, 277
  - by versions, 278
- urgency of a requirement, F-39
- U.S. Rehabilitation Act, Section 508, 170, 175, 186
- usability, 168, 169, 175, 185, F-70. *See also* accessibility
- use cases
  - compared to user stories, F-24
  - defined, F-13
  - for requirement patterns, 37–38
  - writing, F-13–F-14
- use case diagram, F-13
- users
  - allocating IDs to, 99
  - availability of system to. *See* availability
  - behavior of, 212–213
  - counting dynamic, 213
  - forcibly ejecting, 214
  - geographic distribution of, 209
  - limiting number of, 214
  - number of active, 149
  - with specific needs, 168, 169, 172, 174–175, 183–184.
    - See also* accessibility
  - as system, 171
  - wait time warnings for, 203

- user access state, specifying, 143
- user access via Web browser, 68
- user accountability, 282
- user actions, 147, 148
- user authentication, described, 281, 282
- user authentication requirement pattern, 295–305
- user authorization
  - described, 281, 282
  - inquiry requirement, 317
- user authorization requirement pattern, 305–307
- user classes, 285
- user colors, accessibility requirements, 180
- user de-registration, 287, 290
- user experiences, distinct, 261, 267–269
- user function domain, 29
- user function requirement pattern, 155–190
- user IDs, requirement for no special, 293
- user interfaces
  - allowing user to adjust to, 175
  - capabilities, 187, 188
  - inter-system interface requirement pattern not used for, 52
  - multiness in, 267
  - semantic requirements, 178, 186
  - tailoring for specific needs, 183–184
- user interface designer, F-15
- user interface infrastructure, 155, 187–189
- user preferences, 132–133
- user registration, 281, 282
- user registration requirement pattern, 284–295
- user response time, 196, 201
- user roles, 15, 316, F-55
- user sessions
  - allocating, 214
  - created by authentication, 296
  - ending, 298, 300–301
  - viewing, 298, 303–304
- user stories, 9, 10, F-24, F-26
- user times per time zone, 93
- user-accessible information, viewing of, 160
- users
  - blocking, 298, 302–303
  - details about, 285
  - performing their own registration, 284
  - protection of, 298, 301
  - registering, 284
  - special processes for new, 287, 293–294
- UTC (Universal Time Co-ordinated), 90, 92

## V

- valid interactions, testing for an interface, 65
- values
  - allowed, list of, 88
  - of requirement patterns, 20, 45
- variables, 103, 195

- variants of a requirement pattern, 22
- versions
  - required for an upgrade, 277
  - of a requirement pattern, 22
  - of technology, 67
- version history, F-51, F-53
- version numbers
  - in the document history, F-51
  - of patterns, 22
  - of references, F-50
  - strategies for, F-52–F-53
- viewing
  - archived data, 114, 115–116
  - of systemwide information, 160
- vision, accessibility concerns and, 171, 173–174, 178–181
- visual cue for audio alert, 182
- visually distinct requirements, F-34
- voice use, accessibility concerns, 171, 174
- Voluntary Product Accessibility Template (VPAT), 184

## W

- waiving a fee, 335
- warnings about wait times, 203
- waterfall approach to software development, F-6
- Web browsers, 68, 188
- Web content accessibility guidelines, 170
- Web page display time, 200
- Web pages, 123
- Web site availability requirements, 223
- “what changed” section of a document history, F-52
- whole numbers, data type, 87
- whole system, requirements for, 20
- window, rolling, 188
- window size, accessibility requirements, 179
- Windows operating system, specifying, 69
- Windows Vista password complexity requirements, 288
- Word, specifying documentation in, 69
- word processor table, presenting requirements with, F-34
- World Wide Web Consortium, 74
- Write draft requirements specification step, 8, F-18–F-19

## X

- X.509 standard, 76
- XP. *See* extreme programming

## Y

- Y2K problem, 89, 256
- year
  - date storage by, 90
  - financial, 259
  - interest days in, 105
- yes or no data type (Boolean), 88

## About the Author

Stephen Withall has been working as a software professional since 1979, in a range of roles from programmer to chief technical officer. Along the way, he has accumulated many years experiencing both the satisfactions and the frustrations of business analysis, systems analysis, and specifying requirements. He has worked in diverse environments in companies big and small, in 17 countries across four continents. The first half of his career found him roaming the world of banking and finance, and the second half (so far) in diverse kinds of e-commerce. He actively maintains his hands-on software development skills.

Stephen holds a BSc in Mathematical Sciences from Bristol University, U.K. He lives in Sydney, Australia, and he enjoys going to places where few others are. You can reach him at <http://www.withallyourequire.com>.

