This page intentionally left blank
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreword</td>
<td>xxiii</td>
</tr>
<tr>
<td>Preface</td>
<td>xxvii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>xxxiii</td>
</tr>
<tr>
<td>About the Author</td>
<td>xxxv</td>
</tr>
</tbody>
</table>

### Part I  Overview: The Big Picture  1

**Chapter 1**  A Brief History of Software Requirements Methods  3

- Software Requirements in Context: Decades of Advancing Software Process Models  3
- Predictive, Waterfall-Like Processes  5
- Problems with the Model  6
- Requirements in the Waterfall Model: The Iron Triangle  6
- And Yet, the Waterfall Model Is Still Amongst Us  8
- Iterative and Incremental Processes  9
  - Spiral Model  10
  - Rapid Application Development  10
  - Rational Unified Process  11
  - Requirements in Iterative Processes  11
- Adaptive (Agile) Processes  12
  - The Agile Manifesto  12
  - Extreme Programming (XP)  14
  - Scrum  15
Contents

Interlude  Case Study: Tendril Platform  93
  Background for the Case Study  93
  System Context Diagram  95

Part II  Agile Requirements for the Team  97

Chapter 6  User Stories  99
  Introduction  99
    User Story Overview  100
    User Stories Help Bridge the Developer–Customer Communication Gap  101
    User Stories Are Not Requirements  101
  User Story Form  102
    Card, Conversation, and Confirmation  102
    User Story Voice  103
    User Story Detail  104
    User Story Acceptance Criteria  104
  INVEST in Good User Stories  105
    Independent  106
    Negotiable…and Negotiated  107
    Valuable  107
    Estimable  108
    Small  109
    Testable  111
  Splitting User Stories  111
  Spikes  114
    Technical Spikes and Functional Spikes  114
    Guidelines for Spikes  115
  Story Modeling with Index Cards  116
  Summary  117

Chapter 7  Stakeholders, User Personas, and User Experiences  119
  Stakeholders  119
    System Stakeholders  120
    Project Stakeholders  120
    Voice of the Stakeholder: Product Owner  120
    Levels of Stakeholder Involvement  121
    Building Stakeholder Trust  122
    Stakeholder Interactions  122
### Chapter 8 - Agile Estimating and Velocity

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>135</td>
</tr>
<tr>
<td>There’s a Method to This Madness</td>
<td>135</td>
</tr>
<tr>
<td>The Goal Is the Same: More Reliable Estimates</td>
<td>136</td>
</tr>
<tr>
<td>Estimating Scope with Story Points</td>
<td>138</td>
</tr>
<tr>
<td>Understanding Story Points: An Exercise</td>
<td>138</td>
</tr>
<tr>
<td>Exercise Part 1: Relative Estimating</td>
<td>138</td>
</tr>
<tr>
<td>Exercise Part 2: Estimating Real Work with Planning Poker</td>
<td>139</td>
</tr>
<tr>
<td>How Much Time Should We Spend Estimating?</td>
<td>142</td>
</tr>
<tr>
<td>A Parable of Estimating Caution: A Story within a Story</td>
<td>144</td>
</tr>
<tr>
<td>Distributed Estimating with Online Planning Poker</td>
<td>144</td>
</tr>
<tr>
<td>An Alternate Technique: Tabletop Relative Estimation</td>
<td>145</td>
</tr>
<tr>
<td>From Scope Estimates to Team Velocity</td>
<td>146</td>
</tr>
<tr>
<td>Exercise Part 3: Establishing Velocity</td>
<td>146</td>
</tr>
<tr>
<td>Caveats on the Relative Estimating Model</td>
<td>147</td>
</tr>
<tr>
<td>Another Parable: Increasing Velocity, Be Careful</td>
<td>148</td>
</tr>
<tr>
<td>What You Ask For</td>
<td>148</td>
</tr>
<tr>
<td>From Velocity to Schedule and Cost</td>
<td>148</td>
</tr>
<tr>
<td>Estimating Schedule</td>
<td>149</td>
</tr>
<tr>
<td>Estimating Cost</td>
<td>149</td>
</tr>
<tr>
<td>Estimating with Ideal Developer Days</td>
<td>149</td>
</tr>
<tr>
<td>A Hybrid Model</td>
<td>151</td>
</tr>
<tr>
<td>Normalizing Velocity</td>
<td>152</td>
</tr>
<tr>
<td>Summary</td>
<td>152</td>
</tr>
</tbody>
</table>
### Chapter 9  Iterating, Backlog, Throughput, and Kanban

Iterating: The Heartbeat of Agility  
- Iteration Length  
- Iteration Pattern: Plan, Execute, Review, and Retrospective  
- Team Backlog  
- Planning the Iteration  
- Iteration Commitment  
- Executing the Iteration  
- Tracking and Adjustment  
- Review and Retrospective  
- Feature Preview

Backlog, Lean, and Throughput  
- Backlog Maturity, Lean, and Little's Law  
- A Blog Story: Is That Well–Formed Product Backlog Decreasing Your Team's Agility?  
- Little's Law and an Agile Team's Backlog  
- Applying Little's Law to Increase Agility and Decrease Time to Market

Readers React  
- Managing Throughput by Controlling Backlog Queue Length

Software Kanban Systems  
- Kanban System Properties  
- Classes of Service in Kanban

Summary

### Chapter 10  Acceptance Testing

Why Write About Testing in an Agile Requirements Book?  
- Agile Testing Overview  
- What Is Acceptance Testing?  
- Story Acceptance Tests  
- Characteristics of Good Story Acceptance Tests  
- They Test Good User Stories  
- They Are Relatively Unambiguous and Test All the Scenarios  
- They Persist

Acceptance Test-Driven Development  
- Acceptance Test Template  
- Automated Acceptance Testing  
- Automated Acceptance Testing Example: The FIT Approach

Unit and Component Testing  
- Unit Testing  
- Component Testing

Summary
Chapter 11  Role of the Product Owner 201
  Is This a New Role? 201
  Perspectives on Dual Roles of Product Owner and Product Manager 202
    The Name Game: Experimenting with the Product Owner Role/Title 206
    Our Conclusion: Apply the Dual Roles 207
  Responsibilities of the Product Owner in the Enterprise 207
    Managing the Backlog 208
    Just-in-Time Story Elaboration 211
    Driving the Iteration 212
    The Problem of Technical Debt and the Value Stream 216
    Co-planning the Release 217
  Five Essential Attributes of a Good Product Owner 218
  Collaboration with Product Managers 220
  Product Owner Bottlenecks: Part-Time Product Owners, Product Owner Proxies, Product Owner Teams 221
    Product Owner Proxies 221
    Product Owner Teams 221
  Seeding the Product Owner Role in the Enterprise 222
    TradeStation Technologies 222
    CSG Systems 223
    Symbian Software Limited 223
    Discount Tire 224
  Summary 224

Chapter 12  Requirements Discovery Toolkit 228
  The Requirements Workshop 228
    Preparing for the Workshop 229
    Setting the Agenda 231
    Running the Workshop 232
  Brainstorming 232
    Idea Generation 233
    Idea Reduction 235
    Idea Prioritization 236
  Interviews and Questionnaires 237
    Context-Free Questions 238
    Solutions-Context Questions 238
    The Moment of Truth: The Interview 239
    Compiling the Needs Data 239
    A Note on Questionnaires 240
  User Experience Mock-Ups 241
Chapter 14  Role of the Product Manager  
Product Manager, Business Analyst?  276  
Responsibilities of the Product Manager in a Product Company  276  
Business Responsibilities of the Role in the IT/IS Shop  278  
Responsibility Summary  279  
Phases of Product Management Disillusionment in the Pre-Agile Enterprise  280  
Phase 1: Unbridled Enthusiasm  281  
Phase 2: False Sense of Security  281  
Phase 3: Rude Awakening  281  
Phase 4: Resetting Expectations  282  
Phase 5: The Season of Perpetual Mistrust  282  
Exiting the Season of Perpetual Mistrust  282  
Evolving Product Management in the Agile Enterprise  283  
Understanding Customer Need  284  
Documenting Requirements  284  
Scheduling  285  
Prioritizing Requirements  285  
Validating Requirements  286  
Managing Change  286  
Assessing Status  287  
Responsibilities of the Agile Product Manager  287  
Own the Vision and Release Backlog  288  
Managing Release Content  290  
Maintaining the Roadmap  295  
Building an Effective Product Manager/Product Owner Team  295  
Summary  297  

Chapter 15  The Agile Release Train  
Introduction to the Agile Release Train  300  
Rationale for the Agile Release Train  301  
Principles of the Agile Release Train  303  
Driving Strategic Alignment  304  
Institutionalizing Product Development Flow  305  
Designing the Agile Release Train  308  
Planning the Release  308  
Release Objectives  308  
Tracking and Managing the Release  309  
Release Retrospective  310
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 16  Release Planning 319</td>
</tr>
<tr>
<td>Preparing for Release Planning 319</td>
</tr>
<tr>
<td>Release Planning Domain 320</td>
</tr>
<tr>
<td>Planning Attendance 320</td>
</tr>
<tr>
<td>Release Planning Facilitator 320</td>
</tr>
<tr>
<td>Release Planning Checklist 321</td>
</tr>
<tr>
<td>Release Planning Narrative, Day 1 322</td>
</tr>
<tr>
<td>Opening 323</td>
</tr>
<tr>
<td>Business Context 323</td>
</tr>
<tr>
<td>Solution Vision 324</td>
</tr>
<tr>
<td>Architecture Vision 324</td>
</tr>
<tr>
<td>Team Planning Breakouts 325</td>
</tr>
<tr>
<td>Draft Plan Review 327</td>
</tr>
<tr>
<td>Managers’ Review and Problem Solving Meeting 328</td>
</tr>
<tr>
<td>Release Planning Narrative, Day 2 328</td>
</tr>
<tr>
<td>Opening 330</td>
</tr>
<tr>
<td>Planning Adjustments: A United Front 330</td>
</tr>
<tr>
<td>Planning Continues: Team Planning Breakouts Session II 330</td>
</tr>
<tr>
<td>Establishing Release Objectives 330</td>
</tr>
<tr>
<td>Final Release Plans Review 332</td>
</tr>
<tr>
<td>Addressing Risks and Impediments 333</td>
</tr>
<tr>
<td>The Commitment 334</td>
</tr>
<tr>
<td>Planning Retrospective 335</td>
</tr>
<tr>
<td>Final Instructions to Teams 336</td>
</tr>
<tr>
<td>Stretch Goals 336</td>
</tr>
<tr>
<td>Summary 338</td>
</tr>
</tbody>
</table>

| Chapter 17  Nonfunctional Requirements 339 |
| Modeling Nonfunctional Requirements 340 |
| Expressing Nonfunctional Requirements as User Stories 342 |
| Exploring Nonfunctional Requirements 342 |
| Usability 343 |
| Reliability 344 |
## Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Backlog</td>
<td>Activities: Cadence-Based Review, Discussion, and Peer Rating</td>
<td>415</td>
</tr>
<tr>
<td></td>
<td>Prioritization and Rating System</td>
<td>417</td>
</tr>
<tr>
<td></td>
<td>Weighted Rating and Decision Criteria</td>
<td>417</td>
</tr>
<tr>
<td></td>
<td>Pull from Transition to Analysis</td>
<td>418</td>
</tr>
<tr>
<td></td>
<td>Work-in-Process Limits</td>
<td>418</td>
</tr>
<tr>
<td>3. Analysis</td>
<td>Activities</td>
<td>418</td>
</tr>
<tr>
<td></td>
<td>Collaboration with Development</td>
<td>419</td>
</tr>
<tr>
<td></td>
<td>Collaboration with the Business: Solution Management, Product Management, Business Analysts</td>
<td>420</td>
</tr>
<tr>
<td></td>
<td>Work-in-Process Limits</td>
<td>420</td>
</tr>
<tr>
<td></td>
<td>Architectural Epic Business Case Template</td>
<td>420</td>
</tr>
<tr>
<td></td>
<td>Decision Authority</td>
<td>422</td>
</tr>
<tr>
<td>4. Implementation</td>
<td>Implementation Path A: Transition to Development</td>
<td>423</td>
</tr>
<tr>
<td></td>
<td>Implementation Path B: Create a New Team</td>
<td>424</td>
</tr>
<tr>
<td></td>
<td>Implementation Path C: Outsourced Development</td>
<td>425</td>
</tr>
<tr>
<td></td>
<td>Implementation Path D: Purchase a Solution</td>
<td>425</td>
</tr>
<tr>
<td></td>
<td>Work in Process Limits</td>
<td>426</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>427</td>
</tr>
</tbody>
</table>

**Chapter 22** Moving to Agile Portfolio Management

- Portfolio Management                                                                 | 429  |
- When Agile Teams Meet the PMO: Two Ships Pass in the Night                           | 431  |
- Legacy Mind-Sets Inhibit Enterprise Agility                                           | 432  |
- The Problem Is Not “Theirs”, It Is “Ours”                                            | 432  |
- Legacy Mind-Sets in Portfolio Management                                             | 433  |
- Eight Recommendations for Moving to Agile Portfolio Management                       | 436  |
- Rethinking Investment Funding                                                        | 436  |
- Rethinking Change Management                                                         | 440  |
- Rethinking Governance and Oversight                                                  | 442  |
- Summary: On to Agile Portfolio Planning                                              | 447  |

**Chapter 23** Investment Themes, Epics, and Portfolio Planning

- Investment Themes                                                                   | 449  |
- Communicating Investment Themes                                                      | 450  |
- Why Investment Mix Rather Than Backlog Priority?                                      | 451  |
- Epics                                                                                | 452  |
- Subepics                                                                             | 453  |
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxii</td>
<td>Contents</td>
<td>453</td>
</tr>
<tr>
<td></td>
<td>Expressing Epics</td>
<td>453</td>
</tr>
<tr>
<td></td>
<td>Discriminating Epics, Features, and Stories</td>
<td>454</td>
</tr>
<tr>
<td></td>
<td>Types of Epics</td>
<td>456</td>
</tr>
<tr>
<td></td>
<td>Identifying and Prioritizing Business Epics: A Kanban System for Portfolio Planning</td>
<td>456</td>
</tr>
<tr>
<td></td>
<td>Overview</td>
<td>457</td>
</tr>
<tr>
<td></td>
<td>State Diagram View</td>
<td>458</td>
</tr>
<tr>
<td></td>
<td>The Funnel: Problem/Solution Needs Identification</td>
<td>459</td>
</tr>
<tr>
<td></td>
<td>Backlog</td>
<td>461</td>
</tr>
<tr>
<td></td>
<td>Analysis</td>
<td>463</td>
</tr>
<tr>
<td></td>
<td>Implementation</td>
<td>466</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>467</td>
</tr>
<tr>
<td>Chapter 24</td>
<td>Conclusion</td>
<td>469</td>
</tr>
<tr>
<td></td>
<td>Further Information</td>
<td>470</td>
</tr>
<tr>
<td>Appendix A</td>
<td>Context-Free Interview</td>
<td>471</td>
</tr>
<tr>
<td>Appendix B</td>
<td>Vision Document Template</td>
<td>475</td>
</tr>
<tr>
<td>Appendix C</td>
<td>Release Planning Readiness Checklist</td>
<td>485</td>
</tr>
<tr>
<td>Appendix D</td>
<td>Agile Requirements Enterprise Backlog Meta-model</td>
<td>489</td>
</tr>
<tr>
<td>Bibliography</td>
<td></td>
<td>491</td>
</tr>
<tr>
<td>Index</td>
<td></td>
<td>495</td>
</tr>
</tbody>
</table>
Why do product development projects miss their economic objectives? Studies show that 80 to 85 percent of project failures are due to incorrect requirements. Experienced developers know that managing requirements is a greater challenge than technical execution. And, although we have known this for decades, we really haven’t gotten much better at it. Why? At first, we were functionally organized, so we simply displaced the problem outside the boundary of engineering—we blamed marketing and product management. Later, as we adopted cross-functional teams, we told these teams to listen to the voice of the customer and assumed that this would solve the problem.

It didn’t. We never challenged the idea that it was feasible to develop valid requirements up front—we just told people to try harder. We just told them to pay more attention to what the customer was asking for. We ignored the fact that many customers don’t know what they want. We ignored that fact that even when they know what they want, they can’t describe it. We ignored the fact that even when they can describe it, they often describe a proposed solution rather than the real need. For example, customers told us that they wanted suitcases that were easy to carry, and asked us to make them lightweight. We did this, but they rejected our elegant designs and bought the heavier designs of our competitors—the ones with wheels on them!

The sad truth is that there is no one “voice of the customer.” It is a cacophony of voices asking for different things. Even at a single customer, we need to balance the needs of technical decision makers, end users, system operators, and financial decision makers. All of these actors weigh different attributes differently, and they change their weighting as they acquire more experience using the product. We also need to understand the needs of distributors, regulators, manufacturing, and field service. If we focus only on the user, we could miss what Dean calls the “nonfunctional requirements.”
And this problem is dynamic, not static. In the course of our development effort, the context constantly changes—competitors introduce new products and customer needs evolve. If it is not feasible to develop valid requirements before we begin design, what is our alternative? In my opinion, we should start with the belief that even the best requirements will contain major errors, and that these errors grow exponentially with time. This shifts our focus. Instead of believing that we are hearing a high-fidelity signal coming from the customer, we need to recognize that it is a noisy, low-fidelity signal—a signal that must be continually checked for errors. Rather than using heavy front-end investment to create perfect requirements, we invest in creating processes and infrastructure that can rapidly detect and correct poor fits between our solution and the customer’s evolving needs.

What better test for this alternative approach than the development of large systems? Many of the methods that work superbly on small projects break down on large ones. For example, in small systems, costs and benefits are typically local. System performance does not suffer when a team makes locally optimal decisions. This is not true for large systems where we must deal with economic effects that are dispersed physically, temporally, and organizationally.

We need better approaches to understanding and managing software requirements, and Dean provides them in this book. He draws ideas from three very useful intellectual pools: classical management practices, agile methods, and lean product development. By combining the strengths of these three approaches, he has produced something that works better than any one approach in isolation.

First, although it might be unfashionable to say this, classic management practices still offer us some very useful methods. Not all of our predecessors were stupid dolts, incapable of recognizing a working solution. For decades I have seen relatively simple concepts like technology and product roadmaps producing great results. They ensure that work on technology begins early enough to keep it off the critical path. They create strong logical links between technology efforts and the programs that they serve. We don’t need to blindly accept all traditional practices, but we’d be foolish to discard everything our predecessors already learned. Dean shows you how to apply some of these great ideas at the program and portfolio level.

Second, the agile community has developed a very powerful set of ideas that has already produced impressive results. These methods have grown rapidly for a very good reason—they work. Agile decomposes the large batches of the waterfall model into a series of time-boxed iterations. These smaller batches dramatically accelerate feedback, producing enormous benefits.
Since much of agile’s success has occurred in smaller projects, it is natural to ask whether it is equally useful in large systems. While I deeply respect the value of agile methods, I think Dean is correct in recognizing that these methods must be extended to meet the needs of large system development. It is quite risky to assume that large system architectures will naturally emerge and that any shortcomings can be refactored away. For example, a naval warship is designed for a 30-year operating life. Good naval architects anticipate evolving threats, emerging technologies, and changing missions. We do not create such systems by letting architecture “emerge.” Once we recognize the unique challenge of managing at the system level, we can start investing in the organizational infrastructure needed to meet this challenge. Dean shows you how to do this with agile method extensions such as architectural runways.

Dean also draws upon the ideas of what I call “second-generation lean product development.” Many of the initial attempts to use lean in product development focused on ideas such as standardization of work and variability reduction. They lacked agile’s intrinsic appreciation that developing great new solutions requires learning to thrive in the presence of uncertainty. These lean product development methods have now evolved, and the results are impressive. For example, today’s “kanban” approaches are limiting WIP, accelerating feedback, and making flow visible to all participants. You can see the influence of these ideas on Dean’s approaches at the program and portfolio levels. Dean has also recognized the importance of the new emphasis on economics. This emphasis helps us make better decisions and it enables us to explain our choices to management in terms they readily understand.

As you read this book, I suggest paying attention to several things. First, try to understand the reasons why certain of these approaches work, not just what they are. If you understand why things work, then you can more easily adapt them to your own unique context. Second, treat these ideas as a portfolio of useful patterns rather than a rigid set of practices that must be adopted as a group. This will reduce the batch size of your adoption process, produce less resistance, and provide faster results. Finally, as you use these ideas, strive for balance. You will have a natural tendency to prefer certain ideas—they address issues you feel are important, and they feel comfortable. You may have given other areas little attention for a long time. Often the areas that have received little attention hold great untapped opportunity.

—Don Reinertsen

Author of The Principles of Product Development Flow:
Second Generation Lean Product Development
Introduction to the Book

In the past decade, the movement to lighter-weight and increasingly agile methods has been the most significant change to affect the software enterprise since the advent of the waterfall model in the 1970s. Originated by a variety of thought and practice leaders and proven in real-world, successful experiments, the methods have proven themselves to deliver outstanding benefits on the “big four” measures: productivity, quality, morale, and time to market.

In the past five years, the methods spread virally. Within the larger enterprise, the initiatives usually started out with individual teams adopting some or all of the practices espoused by the various methods, primarily XP, Scrum, Lean, Kanban (later), and various combinations and variants.

However, as the methods spread to the enterprise level, a number of extensions to the basic agile methods were necessary to address the larger process, organizational, application scope, and governance challenges of the larger enterprise.

Not the least of these is the challenge of agile requirements, which is the necessity to scale the basic, lightweight practices of team agile—product backlogs, user stories, and the like—to the needs of the enterprise’s Program and Portfolio levels. For example, agile development practices introduced, adopted, and extended the XP-originated “user story” as the primary currency for expressing application requirements. The just-in-time application of the user story provided a much leaner approach and helped eliminate many waterfall-like practices, such as imposing overly detailed and constraining requirements specifications on development teams.

However, as powerful as this innovative concept is, the user story by itself does not provide an adequate, nor sufficiently lean, construct for reasoning about investment,
system-level requirements, and acceptance testing across the larger software enterprise’s project Team, Program, and Portfolio organizational levels. That is the purpose of this book.

This book describes an agile requirements artifact model, corresponding practices, suggested roles, and an organizational model that provides a quintessentially lean and agile requirements subset for the agile project teams that write and test the code. Yet this model also scales to the full needs of the largest software enterprise.

**Why Write This Book?**

In 2000, after about 25 years of managing software development as an entrepreneur and executive, along with my coauthor Don Widrig, I published my first book: *Managing Software Requirements: A Unified Approach*. In 2003, we updated the book with a second edition: *Managing Software Requirements: A Use Case Approach*. These are considered to be definitive texts on managing application requirements—a lot of copies were sold, and the books have been translated into five languages. More importantly, many individuals, teams, and companies told me that these works helped them achieve better software outcomes. That was always the goal.

In the following years, I turned my attention to agile development methods. I continue to be more and more impressed with the power of these innovative methods, the quality and productivity results they delivered, and the way in which they reenergized and empowered software teams. Though the methods were developed and proven in small team environments, the challenges of building software at scale is a more fascinating puzzle—part science, part art, part engineering, part organizational psychology. As a result, I became engaged in helping a number of larger enterprises in adopting and adapting these methods in projects affecting hundreds—and then thousands—of software practitioners. Fortunately, with some extensions, the methods did scale to the challenge. Based on these experiences, in 2007 I published *Scaling Software Agility: Best Practices for Large Enterprises*, a book designed to help larger enterprises achieve the benefits of agile development.

*Scaling Software Agility* took a broad view of software methods and didn’t focus much on software requirements. Even though the management of requirements continued to be a struggle for many agile teams, there were bigger organizational and cultural challenges, as well as a number of emerging agile technical practices, that needed to be addressed.

In the past couple of years, the movement to lean thinking in software development captured my interest, in part because I have some background in lean manufacturing
from earlier days. Generally, lean provides a comprehensive, deeply principled, rigorous, and mathematical framework for reasoning about product development economics and the increasingly important subset, software development.

So, my thinking, along with that of many others, evolved further. Many of us started to see agile development, especially agile at scale, as a “software instance of lean.” In addition, lean scales beyond the software development labs and provides tools to address changes in other departments such as deployment, IT, distribution, and program and portfolio management. Simply put, lean provides a broader framework for organizational change, and it helps us address these larger challenges. I’m a big fan of lean thinking.

At its core, lean focuses on the value stream and provides philosophies, principles, and tools to continually decrease time to market, enhance value delivery, and eliminate waste and delays. As enterprises head down the lean path, it is again beneficial to focus on optimizing the understanding and implementation of software requirements, because they are the unique carriers—or at least the best proxy—for that value stream.

Lean thinking brings us full circle. Once again, it is useful to focus on requirements management practices in our agile—and increasingly lean—software development paradigm. That’s why I wrote this book.

My hope is that the book will help the individual software practitioner, project team, program, and enterprise adopt and adapt agile and lean practices, deliver better solutions to their users and stakeholders, and thereby achieve the personal and business benefits that success engenders. After all, you can never be too rich or too lean.

**HOW TO READ THIS BOOK**

With this book, I’m hoping to tell a somewhat complex story—how to address the challenge of managing software requirements in an agile enterprise that may employ just a few developers building a single product to those employing hundreds or even thousands of software practitioners building systems of previously unseen complexity—in a practical, straightforward, and understandable manner.

To do so, the book is written in four parts, the last three of which are dedicated to describing specific agile requirement practices at increasing levels of sophistication and scale.
Part I, Overview: The Big Picture of Agile Requirements in the Enterprise

In Part I, we describe an overall process model intended to communicate the “Big Picture” of how to apply agile requirements practices at the project Team, Program, and Portfolio levels.

We provide a brief history of software methods, describing the evolution from waterfall through iterative and incremental development, to agile and lean. We describe the big picture of agile requirements—an organization, requirements, and process model that works for the team and yet scales to the full needs of the enterprise.

We then provide an overview of the model and illustrate how it can be applied in agile requirements for the team, agile requirements for the program, and agile requirements for the portfolio.

*If you need an introduction and orientation to the concepts, terms, and general practices of managing agile requirements, this part is intended to stand alone.*

Part II, Agile Requirements for the Team

In Part II, we describe a simple yet comprehensive model for managing requirements for agile project teams. This portion of the model is designed to be as lightweight as possible, quintessentially agile, and to not encumber the agile teams with any unnecessary complexity and overhead. We introduce the agile team, user stories, stakeholders, users and user personas, iterating, agile estimating and velocity, acceptance testing, the role of the product owner, and, finally, methods for discovering requirements.

*If your teams are using agile, this comprehensive, explanatory guide to applying agile requirements is intended for you.*

Part III, Agile Requirements for the Program

Part III is intended for those involved in building more complex systems that often require the cooperation of a number of agile teams. We expand the picture and introduce additional requirements artifacts, roles, organizational constructs, and effective practices designed for this purpose. We describe Vision, product and system features, the product Roadmap, the role of the product manager, the Agile Release Train, release planning, nonfunctional requirements, techniques for requirements analysis, and use cases.
If you are a developer, tester, manager, team lead, QA, architect, project or program manager, or development director/executive involved in building systems of this scope, this part is intended for you.

**Part IV, Agile Requirements for the Portfolio**

In Part IV, we describe the final, *Portfolio* level, of requirements practices. This level is intended to guide enterprises building ever-larger systems of systems, application suites, and product portfolios. These often require the coordination and cooperation of large numbers (20 or 50 or 100 or more) of agile project teams. We introduce additional requirements artifacts, roles, organizational constructs, and practices designed for this purpose. We describe the role that larger-scale, intentional, system-level architectures play in agile development. We introduce a kanban system for reasoning about how to evolve and, when necessary, rearchitect, such systems in an agile manner. We also describe some of the legacy thinking in portfolio and project management and give some suggestions as to what to do about it. We conclude with a chapter describing investment themes, epics, and, finally, one of the ultimate objectives—agile portfolio planning.

*If you are a program manager, development director, system architect, executive, or portfolio manager or planner who is involved in managing investments for a portfolio of products, systems, software services, or IT applications, this part is intended for you.*
Chapter 2

The Big Picture of Agile Requirements

This would all be a lot easier to understand if you could just draw me a picture.
—Anonymous senior executive

Effectively implementing a new set of lean and agile requirements principles and practices in a project team, program, or enterprise is no small feat. Even the language is different and seemingly odd (user stories, sprints, velocity, story points, epics, backlog?). In addition, further “leaning” the organization often requires eliminating or reducing requirements specifications, design specifications, stage-gated governance models (with incumbent requirements reviews), sign-offs (with incumbent delays...), implementing work-in-process limits (which may seem counterproductive to those who measure “utilization”), and so on. So, there will likely be many challenges.

Even for the fully committed, it can take six months to a year to introduce and implement the basic practices and even more time to achieve the multiples of productivity and quality results that pay the ultimate dividends in customer satisfaction, revenue, or market share. To achieve these benefits, we must change many things, including virtually all of our former requirements management practices. However, many of the existing required artifacts, milestones, and so on, serve as safeguards to “help” avoid the types of project problems that software has often experienced. So, we have a dilemma—how do we practice this new high-wire act without a safety net, when the safety net itself is a big part of the problem?

Fortunately, we are now at the point in time where a number of organizations have made the transition before us and some common patterns for lean and agile software process success have started to emerge. In our discussions with teams, managers, and executives during this transition, we often struggled to find a language for discussion, a set of abstractions, and an appropriate graphic that we could use to quickly describe “what your enterprise would look like and how it would work after such an agile transformation.”

To do so, we need to be able to describe the new software development and delivery process mechanisms, the new teams and organizational units, and some of the roles key individuals play in the new agile paradigm. In addition, any such Big Picture
should highlight the requirements practices of the model, because those artifacts are the proxy for the value stream.

Eventually, and with help from others, we arrived at something that worked reasonably well for its purpose.¹ We call it the Agile Enterprise Big Picture, and it appears in Figure 2–1.

**THE BIG PICTURE EXPLAINED**

In this chapter, we’ll explain the Big Picture in a summary format intended to provide the reader with a quick gestalt of this new, agile, leaner, and yet fully scalable software requirements model.

---

¹ Special thanks to Matthew Balchin and others at Symbian Software, Ltd., and Juha-Markus Aalto of Nokia Corporation.
In the remaining chapters of Part I of this book, we'll describe the basic big-picture requirements management practices for the individual Team, Program, and Portfolio levels. In Parts II, III, and IV, we'll further elaborate on the requirements management artifacts, roles, and activities at a level of detail suitable for implementation and action.

**Big-Picture Highlights**

Because this picture serves as both the organizational and process model for our agile requirements practices, we'll have time throughout this book to explore its many nuances. However, from an overview perspective, the following highlights emerge.

**The Team Level**

At the *Team level*, agile teams of 7±2 team members define, build, and test *user stories* in a series of *iterations* and *releases*. In the smallest enterprise, there may be only a few such teams. In larger enterprises, groups, or *pods*, of agile teams work together to support building up larger functionality into complete products, features, architectural components, subsystems, and so on. The responsibility for managing the *backlog* of user stories and other things the team needs to do belongs to the team's *product owner*.

**The Program Level**

At the *Program level*, the development of larger-scale systems functionality is accomplished via multiple teams in a synchronized *Agile Release Train* (ART). The ART is a standard cadence of timeboxed iterations and milestones that are date- and quality-fixed, but scope is variable (no iron triangle). The ART produces *releases* or *potentially shippable increments* (PSIs) at frequent, typically fixed, 60- to 120-day time boundaries. These evaluable increments can be released to the customer, or not, depending on the customer’s capacity to absorb new product as well as external events that can drive timing.

We'll use the generic *product manager* label as the title for those who are responsible for defining the features of the system at this level, though we'll also see that many other titles can be applied to this role.

**The Portfolio Level**

At the *Portfolio level*, we’ll talk about a mix of *investment themes* that are used to drive the investment priorities for the enterprise. We’ll use that construct to assure that the work being performed is the work necessary for the enterprise to deliver on its chosen business strategy. Investment themes drive the portfolio vision, which will be expressed in as a series of larger, *epic*-scale initiatives, which will be allocated to various release trains over time.
In the rest of this chapter, we’ll walk through the various elements of the Big Picture to describe how it works. While we’ll highlight the requirements value delivery stream, we’ll also expose the rest of the picture including the roles, teams, and processes that are necessary to deliver value. In this way, we’ll provide a systemic view of our lean and agile requirements process that works for teams and yet scales to the full needs of the enterprise.

**Big Picture: Team Level**

Figure 2–2 summarizes the Team level of the Big Picture.

The Agile Team

The “front line” of software development consists of some number of agile teams that implement and test code and collaborate on building the larger system. It’s appropriate to start with the team, because in agile, the team is the thing, because they write and test all the code that delivers value to the end user. Since it’s an agile team, each has a maximum of seven to nine members and includes all the roles necessary to define/build/test the software for their feature or component. The roles include a Scrum/Agile Master, product owner, and a small team of dedicated developers, testers and (ideally) test automation experts, and maybe a tech lead.

In its daily work, the team is supported by architects, external QA resources, documentation specialists, database specialists, source code management (SCM)/build/infrastructure support personnel, internal IT, and whoever else it takes such that the core team is fully capable of defining, developing, testing, and delivering working and tested software into the system baseline.

---

Since testing software is integral to value delivery (teams get no credit for untested code), testers are integral to the team. Often the testers are logically part of the QA organization but are physically assigned and dedicated to an agile team. In this matrix fashion, their primary allegiance is to the team, but as members of the QA organization, they can leverage other QA teammates and managers for skills development, automation expertise, and any specialty testing capabilities that may be necessary at the system level. In any case, it must be clear that the agile team itself is responsible for the quality of their work product and that responsibility cannot be delegated (or abrogated!) to any other organization, in or out of house.

Teams are typically organized to deliver software features or components. Most enterprises will have a mix of both types—some component teams focused on shared infrastructure, subsystems, and persistent, service-oriented architectural components and some feature teams focused on vertical, user-facing, value-delivery initiatives. Agile teams are self-organizing and reorganize when necessary based on the work in the program backlog. Over time, the makeup of the teams themselves is more dynamic than static—static enough to “norm, storm, and perform”3 for reasonable periods of time and dynamic enough to flex to the organization’s changing priorities.

**Pods of Agile Teams**

In addition, within the larger enterprise, there are typically some number (three to ten) or so of such teams that cooperate to build a larger feature, system, or subsystem (the program domain in the Big Picture). Although this isn’t a hard or fast rule, experience has shown that even for very large systems, the logical partitions defined by system or product family architecture tend to cause “pods” of developers to be organized around the various implementation domains. This implies that perhaps 50 to 100 people must intensely collaborate on building their “next bigger thing” in the hierarchy, which we’ll call a program. As we’ll discover later, this is also about the maximum size for face-to-face, collaborative release planning.

Of course, even that’s an oversimplification for a really large system, because there are likely to be a number of such programs, each contributing to the portfolio (product portfolio, application suite, systems of system).

---

Roles in the Agile Team

Product Owner
As we have described, Scrum is the dominant agile method in use, and the product owner role is uniquely, if arbitrarily, defined therein. In Scrum, the product owner is responsible for determining and prioritizing user requirements and maintaining the product backlog. Moreover, even if a team is not using Scrum, it has been our experience that implementing the product owner role—as largely defined by Scrum—can deliver a real breakthrough in simplifying the team’s work and organizing the entire team around a single, prioritized backlog.

But the product owner’s responsibilities don’t end there. In support of Agile Manifesto principle #4—Business people and developers must work together daily throughout the project—the product owner is ideally co-located with the team and participates daily with the team and its activities.

Scrum/Agile Master
For teams implementing Scrum, the Scrum Master is an important (though sometimes transitional) role. The Scrum Master is the team-based management/leadership proxy whose role is to assist the team in its transition to the new method and continuously facilitate a team dynamic intended to maximize performance of the team.

In teams that do not adopt Scrum, a comparable leadership role typically falls to a team lead, an internal or external coach, or the team’s line manager. As their skills develop, many of these Agile Masters become future leaders by illustrating their ability to deliver user value and by driving continuously improving agile practices.

Developers and Testers
The rest of the core team includes the developers and testers who write and test the code. Since this is an agile team, the team size is typically limited to about three to four developers plus one to two testers, who are (ideally) co-located and work together to define, build, test, and deliver stories into the code baseline.

Iterations
In agile development, new functionality is built in short timeboxed events called iterations (sprints in Scrum). In larger enterprises, agile teams typically adopt a

---

4. As the teams master the agile process, the role becomes less critical. Some very agile teams, even those who have adopted Scrum, no longer have a Scrum Master per se. Everybody knows the rules, and they are self-enforced.
standard iteration length and share start and stop boundaries so that code maturity is comparable at each iteration-boundary system integration point.

Each iteration represents a valuable increment of new functionality, accomplished via a constantly repeating standard pattern: plan the iteration, build and test stories, demonstrate the new functionality to stakeholders, inspect and adapt, repeat.

The iteration is the “heartbeat of agility” for the team, and teams are almost entirely focused on developing new functionality in these short timeboxes. In the Big Picture, the iteration lengths for all teams are the same since that is the simplest organizational and management model. Although there is no mandated length, most have converged on a recommended length of two weeks.

**Number of Iterations per “Release”**

A series of iterations is used to aggregate larger, system-wide, functionality for release (or potential release) to the external users. In the Big Picture, we’ve illustrated four development iterations (indicated by a full iteration backlog) followed by one hardening (or stabilization) iteration (indicated by an empty backlog) prior to each release increment.

This pattern is arbitrary, and there is no fixed rule for how many times a team iterates prior to a potentially shippable increment (PSI). Many teams apply this model with four to five development iterations and one hardening iteration per release, creating a cadence of a potentially shippable increment about every 90 days. This is a fairly natural production rhythm that corresponds to a reasonable external release frequency for customers, and it also provides a nice quarterly planning cadence for the enterprise itself.

In any case, the length and number of iterations per release increment, and the decision as to when to actually release an increment, are left to the judgment of the enterprise.

**User Stories and the Team Backlog**

User stories (stories for short) are the general-purpose agile substitute for what traditionally has been referred to as software requirements (the stuff in the middle of the iron triangle of Chapter 1).

Originally developed within the constructs of XP, user stories are now endemic to agile development in general and are typically taught in Scrum, XP, and most other agile implementations. In agile, user stories are the primary objects that carry the customer’s requirements through the value stream—from needs analysis through code and implementation.
As opposed to requirements (which by common definition represent something the system must do to fulfill a business need or contractual obligation), user stories are brief statements of intent that describe something the system needs to do for some user. As commonly taught, the user story often takes a standard user-voice form of the following:

\[
\text{As a } \text{<user role>}, \text{ I can } \text{<activity> so that } \text{<business value>}.\]

With this form, the team learns to focus on both the user’s role and the business benefit that the new functionality provides. This construct is integral to agile’s intense focus on value delivery.

**Team Backlog**

The team’s backlog (typically called a project or product backlog) consists of all the user stories the team has identified for implementation. Each team has its own backlog, which is maintained and prioritized by the team’s product owner. Although there may be other things in the team’s backlog as well—defects, refactors, infrastructure work, and so on—the yet-to-be-implemented user stories are the primary focus of the team.

Identifying, maintaining, prioritizing, scheduling, elaborating, implementing, testing, and accepting user stories is the primary requirements management process at work in the agile enterprise.

Therefore, we will spend much of the rest of this book further describing processes and practices around user stories.

**Tasks**

For more detailed tracking of the activities involved in delivering stories, teams typically decompose stories into tasks that must be accomplished by individual team members in order to complete the story. Indeed, some agile training uses the task object as the basic estimating and tracking metaphor.

However, the iteration tracking focus should be at the story level, because this keeps the team focused on business value, rather than individual tasks. Tasks provide a micro–work breakdown structure that teams can use (or not) to facilitate coordinating, estimating, tracking status, and assigning individual responsibilities to help assure completion of the stories—and thereby—the iteration.

**Big Picture: Program Level**

Figure 2–3 summarizes the Program level of the Big Picture.
Here, we find additional organizational constructs, roles, processes, and requirements artifacts suited for building larger-scale systems, applications, products, and suites of products.

**Releases and Potentially Shippable Increments**

Although the goal of every iteration is to produce a shippable increment of software, teams (especially larger-scale enterprise teams) find that it may simply not be practical or appropriate to ship an increment at each iteration boundary. For example, during the course of a series of iterations, the team may accumulate some technical debt that needs to be addressed before shipment. Technical debt may include things such as defects to be resolved, minor code refactoring, deferred system-wide testing for performance, reliability, or standards compliance, or finalization of user documentation. **Hardening iterations** (indicated by an iteration with an empty backlog) are included in the Big Picture to provide the time necessary for these additional activities.

Moreover, there are legitimate business reasons why not every increment should be shipped to the customer. These include the following:

- Potential interference with a customer’s licensing and service agreements
- Potential for customer overhead and business disruption for installation, user training, and so on
- Potential for disrupting customer’s existing operations with minor regressions or defects
For these and other reasons, most programs aggregate a series of iterations into a potentially shippable increment, which can be released, or not, based on the then-current business context.

**Vision, Features, and the Program Backlog**

Within the enterprise, the product management (or possibly program management or business analyst) function is primarily responsible for maintaining the Vision of the products, systems, or application in their domain of influence.

The Vision answers the big questions for the system, application, or product, including the following.

- What problem does this particular solution solve?
- What features and benefits does it provide?
- For whom does it provide it?
- What performance, reliability, and so on, does it deliver?
- What platforms, standards, applications, and so on, will it support?

**The Primary Content of the Vision Is a Set of Features**

A Vision may be maintained in a document, in a backlog repository, or even in a simple briefing or presentation form. But no matter the form, the prime content of the Vision document is a prioritized set of features intended to deliver benefits to the users.

**Nonfunctional Requirements**

In addition, the Vision must also contain the various nonfunctional requirements, such as reliability, accuracy, performance, quality, compatibility standards, and so on, that are necessary for the system to meet its objectives.

**Undelivered Features Fill the Program Backlog**

In a manner similar to the team’s backlog, which contains primarily stories, the program (or release) backlog contains the set of desired and prioritized features that have not yet been implemented. The program backlog may or may not also contain...
estimates for the features. However, any estimates at this scale are coarse-grained and imprecise, which prevents any temptation to over-invest in inventory of too-early feature elaboration and estimation.

**Release Planning**

In accordance with emerging agile enterprise practices, each release increment time-box has a kickoff release planning session that the enterprise uses to set the company context and to align the teams to common business objectives for the release. The input to the release planning session is the current Vision, along with a set of objectives and a desired, prioritized feature set for the upcoming release.

By breaking the features into stories and applying the agreed-to iteration cadence and knowledge of their velocity, the teams plan the release, typically in a group setting. During this process, the teams work out their interdependencies and design the release by laying stories into the iterations available within the PSI timebox. They also negotiate scope trade-offs with product management, using the physics of their known velocity and estimates for the new stories to determine what can and can’t be done. In addition to the plan itself, another primary result of this process is a commitment to a set of release objectives, along with a prioritized feature set.

Thereafter, the teams endeavor to meet their commitment by satisfying the primary objectives of the release, even if it turns out that not every feature makes the deadline.

**The Roadmap**

The results of release planning are used to update the (product or solution) *Roadmap*, which provides a sense of how the enterprise hopes to deliver increasing value over time.

The Roadmap consists of a series of planned release dates, each of which has a theme, a set of objectives, and a prioritized feature set. The “next” release on the Roadmap is *committed to the enterprise*, based on the work done in the most recent release planning session. Releases beyond the next one are not committed, and their scope is fuzzy at best.
The Roadmap, then, represents the enterprise’s current “plan of intent” for the next and future releases. However, it is subject to change—as development facts, business priorities, and customers need change—and therefore release plans beyond the next release should not generally be used to create any external commitments.

**Product Management**

In agile, there can be a challenge with the apparently overlapping responsibilities of the *product manager* and the product owner. For example, in Scrum, the product owner is responsible for the following:

- representing the interests of everyone with a stake in the resulting project . . . achieves initial and ongoing funding by creating the initial requirements, return on investment objectives, and release plans.\(^5\)

In some smaller organizational contexts, that definition works adequately, and one or two product owners are all that are needed to define and prioritize software requirements. However, in the larger software enterprise, the set of responsibilities imbued in the Scrum product owner is more typically a much broader set of responsibilities shared between team and technology-based product owners and market or program-based *product managers*, who carry out their traditional responsibilities of both defining the product and presenting the solution to the marketplace.

However, we also note that the title of the person who plays this role may vary by industry segment, as shown in Table 2–1.

**Responsibilities of the Agile Product Manager in the Enterprise**

No matter the title (we’ll continue to use *product manager* generically), when an agile transition is afoot, the person playing that role must fulfill the following primary responsibilities:

- Own the Vision and program (release) backlog
- Manage release content
- Maintain the product Roadmap
- Build an effective product manager/product owner team

---

\(^5\) [Schwaber 2007]
Table 2–1  Product Manager Role May Have Different Titles

<table>
<thead>
<tr>
<th>Industry Segment</th>
<th>Common Title for the Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information systems/information technology (IS/IT)</td>
<td>Business owner, business analyst, project or program manager</td>
</tr>
<tr>
<td>Embedded systems</td>
<td>Product, project, or program manager</td>
</tr>
<tr>
<td>Independent software vendor</td>
<td>Product manager</td>
</tr>
</tbody>
</table>

**Big-Picture Elements: Portfolio Level**

Figure 2–4 summarizes the Portfolio level of the Big Picture.

At the top of the Big Picture, we find the portfolio management function, which includes those individuals, teams, and organizations dedicated to managing the investments of the enterprise in accordance with the enterprise business strategy. We also find two new artifact types, *investment themes* and *epics*, which together create the *portfolio vision*.

**Investment Themes**

A set of *investment themes* establishes the relative investment objectives for the enterprise or business unit. These themes drive the vision for all programs, and new epics are derived from these themes. The derivation of these decisions is the responsibility of the portfolio managers, either line-of-business owners, product councils, or others who have fiduciary responsibilities to their stakeholders.

The result of the decision process is a set of themes—*key product value propositions that provide marketplace differentiation and competitive advantage*. Themes have a much longer life span than epics, and a set of themes may be largely unchanged for up to a year or more.

**Epics and the Portfolio Backlog**

Epics represent the highest-level expression of a customer need. Epics are development initiatives that are intended to deliver the value of an investment theme and are identified, prioritized, estimated, and maintained in the *portfolio backlog*. Prior to release planning, epics are decomposed into specific features, which in turn are converted into more detailed stories for implementation.
Epics may be expressed in bullet form, in user-voice story form, as a sentence or two, in video, in a prototype, or indeed in any form of expression suitable to express the intent of the product initiative. With epics, clearly, the objective is strategic intent, not specificity. In other words, the epic need only be described in detail sufficient to initiate a further discussion about what types of features an epic implies.

**Architectural Runway**

In Chapter 1, we described how design (architecture) and requirements are simply two sides of the same coin—the “what” and the “how.” In this book, we’ll have time to explore this topic in more detail, and we’ll provide some discriminators that help us think about the differences in architecture and requirements, as well as the commonalities. However, even though this book focuses on requirements, we can’t ignore architecture, because experience tells us that teams that build some amount of architectural runway, which is the ability to implement new features without excessive refactoring, will eventually emerge as the winners in the marketplace. So, any effective treatment of agile requirements must address the topic of architecture as well.

Therefore, system architecture is a first-class citizen of the Big Picture and is a routine portfolio investment consideration for the agile enterprise.
SUMMARY

In this chapter, we introduced the Big Picture as the basic requirements artifact, process, and organizational model for managing software requirements in a lean and agile manner. For agile teams, the model uses the minimum number of artifacts, roles, and practices that are necessary for a team to be effective. However, the model expands as needed to the Program and Portfolio levels, in each case providing the leanest possible approach to managing software requirements, even as teams of teams build larger and larger systems of systems. In the next few chapters, we’ll elaborate on each of these levels.
## Index

<table>
<thead>
<tr>
<th>A</th>
<th>Acceptable spikes, 115</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceptance criteria for user stories, 104–105</td>
<td></td>
</tr>
<tr>
<td>Acceptance Test-Driven Development (ATDD), 173, 190–191</td>
<td></td>
</tr>
<tr>
<td>Acceptance tests, 183</td>
<td></td>
</tr>
<tr>
<td>automated, 193–195</td>
<td></td>
</tr>
<tr>
<td>characteristics, 188–190</td>
<td></td>
</tr>
<tr>
<td>checklist, 192–193</td>
<td></td>
</tr>
<tr>
<td>component testing, 198</td>
<td></td>
</tr>
<tr>
<td>defining, 187</td>
<td></td>
</tr>
<tr>
<td>features, 77, 187</td>
<td></td>
</tr>
<tr>
<td>importance, 183–184</td>
<td></td>
</tr>
<tr>
<td>overview, 184–187</td>
<td></td>
</tr>
<tr>
<td>story, 187–190</td>
<td></td>
</tr>
<tr>
<td>teams for, 58–59</td>
<td></td>
</tr>
<tr>
<td>test-driven development, 190–191</td>
<td></td>
</tr>
<tr>
<td>tester responsibilities, 53</td>
<td></td>
</tr>
<tr>
<td>unit testing, 196–198</td>
<td></td>
</tr>
<tr>
<td>and user stories, 103</td>
<td></td>
</tr>
<tr>
<td>Accepted risk category in release planning, 334</td>
<td></td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
</tr>
<tr>
<td>estimate predictions, 136</td>
<td></td>
</tr>
<tr>
<td>in reliability, 344</td>
<td></td>
</tr>
<tr>
<td>in usability, 350</td>
<td></td>
</tr>
<tr>
<td>Achievability in SMART acronym, 163</td>
<td></td>
</tr>
<tr>
<td>Activities</td>
<td></td>
</tr>
<tr>
<td>in kanban system analysis, 463</td>
<td></td>
</tr>
<tr>
<td>RUP, 11</td>
<td></td>
</tr>
<tr>
<td>system stakeholder, 125</td>
<td></td>
</tr>
<tr>
<td>in user stories, 104</td>
<td></td>
</tr>
<tr>
<td>Activity diagrams, 357</td>
<td></td>
</tr>
<tr>
<td>Actors in use cases, 370–371, 373–374</td>
<td></td>
</tr>
<tr>
<td>Adaptive processes, 12</td>
<td></td>
</tr>
<tr>
<td>Agile Manifesto, 12–14</td>
<td></td>
</tr>
<tr>
<td>enterprise-scale, 19</td>
<td></td>
</tr>
<tr>
<td>Extreme Programming, 14–15</td>
<td></td>
</tr>
<tr>
<td>requirements management, 16–19</td>
<td></td>
</tr>
<tr>
<td>Scrum, 15</td>
<td></td>
</tr>
<tr>
<td>Agenda for requirements workshops, 231</td>
<td></td>
</tr>
<tr>
<td>Aggregate behavior in use case modeling, 247</td>
<td></td>
</tr>
<tr>
<td>Agile Estimating and Planning (Cohn), 138, 262</td>
<td></td>
</tr>
<tr>
<td>Agile Manifesto, 12–14, 23–24, 80–81</td>
<td></td>
</tr>
<tr>
<td>Agile Masters, 36</td>
<td></td>
</tr>
<tr>
<td>Agile Product Management with Scrum (Pichler), 204</td>
<td></td>
</tr>
<tr>
<td>Agile Release Train (ART), 33, 80–82, 299</td>
<td></td>
</tr>
<tr>
<td>decentralized rolling-wave planning, 445–446</td>
<td></td>
</tr>
<tr>
<td>designing, 308</td>
<td></td>
</tr>
<tr>
<td>introduction, 300–301</td>
<td></td>
</tr>
<tr>
<td>principles, 303–304</td>
<td></td>
</tr>
<tr>
<td>product development flow, 305–307</td>
<td></td>
</tr>
<tr>
<td>rationale, 301–303</td>
<td></td>
</tr>
<tr>
<td>releases, 290–291</td>
<td></td>
</tr>
<tr>
<td>frequency, 313–317</td>
<td></td>
</tr>
<tr>
<td>planning, 308–309</td>
<td></td>
</tr>
<tr>
<td>predictability, 310–313</td>
<td></td>
</tr>
<tr>
<td>retrospective, 310</td>
<td></td>
</tr>
<tr>
<td>tracking and managing, 309–310</td>
<td></td>
</tr>
<tr>
<td>strategic alignment, 304–305</td>
<td></td>
</tr>
<tr>
<td>Agile Testing (Crissan and Gregory), 185–186</td>
<td></td>
</tr>
<tr>
<td>Agility</td>
<td></td>
</tr>
<tr>
<td>and legacy mind-sets, 432</td>
<td></td>
</tr>
<tr>
<td>Little’s law for, 172–175</td>
<td></td>
</tr>
<tr>
<td>Alignment in Agile Release Train, 304–305</td>
<td></td>
</tr>
<tr>
<td>Alternate flows in use cases, 375</td>
<td></td>
</tr>
</tbody>
</table>
Alternate scenarios in use cases, 368, 371
Alternatives and Competition section in vision documents, 479
Amazon Architecture, 392–393
Analysis
  kanban system, 411, 463–466
  rearchitecting with flow, 418–422
  requirements. See Requirements analysis
Analyst’s Summary for interviews, 240
Anderson, David, 179–180
Andres, Cynthia, 14
Annual funding, 440–441
Annual planning, 445–447
Applicable Standards section in vision documents, 482
Application standards, 346
Architects
  kanban system analysis, 464
  responsibilities, 54
  system, 388–390
Architectural epics, 88–89, 384
  analysis, 418–422
  backlog, 415–418
  business case templates, 420–421
  business drivers for, 387–388
  implementing, 89–90, 423–426
  Kanban system objectives, 408–409
  overview, 409–410
  queue descriptions, 410–411
  ranking, 414–415
  sources, 413–414
  splitting, 403–405
  state descriptions, 411–412
  systems architecture, 399–403
Architectural flow, 384, 399
Architectural governance, 386, 388
Architectural runways, 44, 88–91, 383, 395–396
Architectural work in process (AWIP), 408–409
Architecture
  architectural flow principle, 399
  architectural runway principle, 395–396
  coding and modeling principle, 392–394
  emergence of, 385–386
  innovation principle, 397–399
  portfolio level, 383–384
  principles overview, 390
  role collaboration principle, 396–397
  simplicity principle, 391–392
  splitting architecture epics, 403–405
  systems, 384–390
  team coding principle, 390–391
  testing principle, 395
  vision for, 324
ART. See Agile Release Train (ART)
Assessing release status, 287, 289
Assumptions and Dependencies section in vision documents, 481
ATDD (Acceptance Test-Driven Development), 173, 190–191
Attendance for release planning, 320
Attitudes of product managers, 280–283
Attributes of product owners, 218–220
Automated tests, 186, 193–195
Availability issues in reliability, 344
AWIP (architectural work in process), 408–409
B
Backlogs, 33, 43, 383
  benefits, 169–170
  blog story, 170–171
  as business drivers, 387
  changing and eliminating, 178
  epics, 85–87, 453, 461–463
  features, 76, 255
  investment themes in, 451–452
  in iteration, 157–158, 169
  kanban system, 411, 415–418, 461–463
  Little’s law in, 171–175
  meta-model, 489
  nonfunctional requirements, 78–79, 341, 348
  prioritizing, 209–211
  problems with, 368
  product development flow, 306
product managers for, 289–290
product owners for, 208–211
program level, 40–41
reader reactions to, 176–177
rearchitecting with flow, 415–418
and throughput, 177
user stories in, 38, 55–56
Baker, Steven, 432
Balchin, Mathew, 224
Bartleman, John, 222
Basic features in Kano model, 270
Basic flows in use cases, 371, 375
Batch sizes
  lean software, 26
  Little’s law, 109
  product development flow, 306
  user stories, 110–111
Beck, Kent
  simplicity, 392
  TDD, 190
  XP, 14, 100
Beedle, Mike, 221
Berra, Yogi, 47
Best practices and standards, 346
Beyond Budgeting: How Managers Can Break Free from the Annual Performance Trap, 450
Bias in interviews, 238
Bibliography, 491–494
Big, up-front design (BUFD) requirements, 11
Big Picture, 31–32
  highlights, 33–34
  overview, 32–33
  portfolio level, 43–44
  program level, 38–43
  team level, 34–38
Big visible information radiator (BVIR), 164–165
Black, Keith, 69, 222
Black-box testing, 351–352
Boehm, Barry, 10
Bottom-up, team-based estimating, 259

Brainstorming
  idea generation, 233–234
  idea prioritization, 236–237
  idea reduction, 235–236
  online, 237
  requirements discovery, 232–237
Breakouts in team planning, 325–327, 330
Breaks in system building, 401–402
Broad brushstroke iteration approach, 160–161
Budgeting in investment themes, 450–451
BUFD (big, up-front design) requirements, 11
Build step in define/build/test sequence, 49
Burndown charts
  in iteration, 164
  release, 309
  task hours, 162
Business Analysis Planning and Monitoring knowledge area, product manager responsibilities in, 278
Business analysts. See also Product managers rearchitecting with flow analysis, 420 responsibilities, 201, 203, 276
Business cases
  epics, 464–466
  investment funding, 438
  kanban system rearchitecting with flow analysis, 420–421
  in vision, 253
Business context in release planning, 323–324
Business drivers for architectural epics, 387–388
Business epics, 89, 449–450, 452–453
  backlog, 461–463
  business case templates, 464–466
  expressing, 453–454
  vs. features and stories, 454–456
  identifying and prioritizing, 456–467
  kanban system, 456
  ranking, 460
  sources, 459–460
  subepics, 453
  types, 453
Business-facing tests, 185
Business plans in investment funding, 438
Business rules in user story splitting pattern, 112
Business sense of product owners, 219
Business systems analysts, 206
Business team collaboration for rearchitecting with flow analysis, 420
Business value
  in backlog ratings, 417, 462
  of estimates, 137
  for release objectives, 331–332
BVIR (big visible information radiator), 164–165

C
Cadence
  description, 26
  product development flow, 307
Cadence-based reviews, 415–416, 461–463
Cadence calendars in iteration, 168
Calendars in iteration, 168, 215–216
Capacity in performance, 345
Cards
  acceptance tests, 58–59
  user stories, 102–103, 116–117, 211
Cargill, Tom, 339
Case studies. See Tendril case study
Centralized annual planning, 445–447
Centralized development in user experience, 131–132
Change management
  portfolio management, 430
  product managers for, 286
  in requirements discovery, 245–246
  rethinking, 440–442
Changing requirements, 16
Chaos report survey, 6
Chapman, Chris, 224
Chief engineer role in iteration, 161
Cho, Fuijo, 155
Classes of service
  investment themes, 452
  kanban system, 180
Classifying system stakeholders, 125
Clean Code: A Handbook of Agile Software Craftsmanship (Martin), 346
Cleland-Huang, Jane, 262
Co-development and Alliances knowledge area, product manager responsibilities in, 277
Co-location of teams, 71
Co-planning releases, 217–218
Cockburn, Alistair, 367–369
CoD. See Cost of Delay (CoD)
Code
  in architecture principles, 392–394
  developer responsibilities, 52
  team, 390–391
Cohn, Mike
  Agile Estimating and Planning, 138, 262
  product owners, 219, 221
  Scrum guidance, 205
  Succeeding with Agile: Software Development Using Scrum, 219
  User Stories Applied, 100, 103
Collaboration
  kanban system analysis, 464
  product owners, 220, 296–297
  rearchitecting with flow analysis, 419–420
  role collaboration principle, 396–397
Collective team judgment, 141
Collins, James, 449
Commercialization phase, product manager responsibilities in, 277
Commitment
  estimates, 137
  iteration, 159–163, 214
  in release planning, 273, 292–293, 334–335
Common infrastructure as business driver, 388
Common usage model, 413
Communication
  investment themes, 451
  product owner skills for, 219
  with product owners, 294
  for requirements analysis, 355–356
  user stories for, 101
  vision, 289
Competitive analysis
  in requirements discovery, 244–245
  in vision documents, 479
Complexity
  story points, 138
  in user stories, 110, 112
Component teams, 35, 65–66, 68–71
Component tests, 186, 198
Compound stories, 111–114
Conditions of satisfaction in user stories, 103
Confidence for releases, 273
Confirmation
  acceptance tests, 59
  user stories, 103, 211
Constraints
  backlog. See Backlogs
  nonfunctional requirements, 340–341, 345–347
  WIP, 26
Construction phase in RUP, 11
Context
  in release planning, 323–324
  use cases for, 367
Context diagrams, 95
Context-free interviews, 238, 471–474
Continuous content delivery, 441–442
Continuous improvement, 24
Continuous integration, 72
Conversation
  acceptance tests, 58
  user stories, 103, 211
Cooper, Alan, 127
Cooper, James, 409
Corporate best practices and standards, 346
Cost and Pricing section in vision
documents, 481
Cost of Delay (CoD)
  backlog ratings, 417
  estimating, 266–267
  introduction, 263–266
  kanban system, 411
  lean software, 25
Costs
  as business driver, 388
  business epic savings, 459
  estimates, 137, 149, 259
  predicting, 311
  and queue size, 175
  and velocity, 148
Costs per story point, 259
Cottmeyer, Mike, 69
Cranky Product Manager, 205
Crispin, Lisa, 183, 185–186, 192
Criteria for user stories, 104–105
Critical category in idea prioritization, 236–237
Critiquing products, acceptance tests for, 185
CSG Systems, 223
Cumulative voting in idea prioritization, 236
Cunningham, Ward, 119, 194, 391
Customer & Market Research knowledge area,
  product manager responsibilities in, 277
Customer change request systems, 245–246
Customer needs, product managers for, 284
Customer satisfaction
  Kano model of, 269–271
  in requirements management, 16

D
Daily stand-ups, 165
Data entry in user stories, 113
Data sheet approach for vision, 253–254
Data variations in user stories, 112
Data view in UML, 394
Dates, release, 271, 311
Davis, Alan M., 361–362
*Dealing with Darwin* (Moore), 3
Debt in iteration, 216–217
Decentralized control, 27, 307
Decentralized rolling-wave planning, 445–447
Decision authority
  - business epics, 461
  - funnel queues, 415
  - kanban system analysis, 466
  - rearchitecting with flow analysis, 422
Decision criteria in backlog ratings, 417, 462
Decision-making workshops, 232
Decision tables and decision trees, 359–361
Decisiveness of product owners, 219
Decreased complexity in user stories, 110
Defect logs in requirements discovery, 246
Defects
  - backlogs for, 208
  - in reliability, 344
Define/build/test sequence, 49
Defining features, 235–236
Degradation modes in performance, 345
Delay cost
  - backlog ratings, 417
  - estimating, 266–267
  - introduction, 263–266
  - kanban system, 411
  - lean software, 25
Delighters in Kano model, 270
Delivering releases, 293
Deming, W. Edwards
  - process descriptions, 1
  - quotas, 97
  - system aim, 381
  - system contributions, 249
  - systems, 27
Demonstrable spikes, 115
Demonstration in iteration, 167
Denne, Mark, 262
Dependencies in user stories, 106, 110–111
Deployment, program level, 64
Deployment view in UML, 394
Descriptions in use cases, 371
Design constraints
  - nonfunctional requirements, 340, 345–347
  - testing, 352
Detail in user stories, 104
Developer-customer communication gap, user stories for, 101
Developer role and responsibilities, 36, 52
Development
  - constraints on, 346
  - infrastructure building, 72–73
  - iterations, 37, 300
  - kanban system analysis, 464
  - product manager responsibilities, 276–277
  - releases, 314–316
Development management needs, 124
Development team
  - iteration planning meeting preparation responsibilities, 159
  - rearchitecting with flow analysis, 419
Development time estimates, 260
Development WIP, 409
Differential value in optimization, 269–271
Discount Tire, 224
Discovery-based models, 9, 11
Discovery phase, product manager responsibilities in, 276
Disillusionment phases of product managers, 280–283
Disruptive technology for architectural epics, 413
Distributed but governed user experience, 131–133
Distributed teams for iteration plans, 163
Do no harm refactoring approach, 89
Documentation Requirements section in vision documents, 483
Documenting requirements, product managers for, 284
Domains in release planning, 320
Draft plan reviews for release planning, 327–328
Drivers for architectural epics, 387–388
Duplicate investment and architectural epics, 413
Duplication of effort as business driver, 388

E
Eclipse foundation, 11
Economic view in product development flow, 25, 305
Effort estimates, 258–259
Elaboration phase in RUP, 11
Elicitation knowledge area, product manager responsibilities in, 278
Emotional response in usability, 350
Energy independence case study. See Tendril case study
Enterprise Analysis knowledge area, product manager responsibilities in, 279
Enterprise-class systems, systems architecture in, 384–390
Enterprise-scale adaptive processes, 19
Entity-relationship diagrams (ERDs), 365
Epics
architectural. See Architectural epics
business. See Business epics
and portfolio backlogs, 43, 85–87
ERDs (entity-relationship diagrams), 365
“Establishing an Agile Portfolio to Align IT Investments with Business Needs” (Baker), 432
Estimable spikes, 115
Estimating
business value of, 137
cautions, 144, 147
cost, 137, 149, 259
Cost of Delay, 266–267
development time, 260
effort, 258–259
features, 257–260
goals of, 136–137
hybrid models, 151–152
with ideal developer days, 149–151
introduction, 135–136
in INVEST model, 108–109
investment funding, 439–440
with planning poker, 139–142, 144–145
relative, 138–139, 145–147, 258
schedules, 149
story points for, 138, 259
tabletop relative, 145–146
time devoted to, 142–144
velocity, 146–149
Exception conditions in use cases, 375
Exciters in Kano model, 270
Execute phase in iteration, 157
Executing iteration, 164, 214–215
Executive stakeholders in kanban system analysis, 464
Exemplary Use Cases section in vision documents, 482
Existing product offerings, investment in, 85
Exit conditions in use cases, 372, 375
Expectations of system stakeholders, 125
 Expedite service class in kanban system, 180
Exploring Requirements: Quality Before Design (Gause and Weinberg), 238
Exponential effect on Lambda, 174
External releases, 300
Extreme Programming (XP), 14–15, 100

F
Facilitation by product owners, 121
Facilitators
release planning, 320–321
requirements workshops, 230–231
Fact-based governance, 444–445
False sense of security phase, product manager attitude in, 281
Fawcett, Jennifer, 220, 296
Feature prioritization evaluation matrix, 267–268
Feature sets in planned release dates, 271
Features
- acceptance tests, 77, 187
- vs. epics and stories, 454–456
- estimating, 257–260
- in iteration, 169
- overview, 255–257
- portfolio level, 87
- prioritizing, 261–271
- program level, 40–41
- teams, 34–35, 66–71, 75–77
- testing, 77, 260–261
- voice, 257

Feedback in product development flow, 26–27, 307
Fibonacci estimating sequence, 110
Finding user personas, 127–129
Finite state machines (FSMs), 361–363
First degree system stakeholders, 125
FIT (Framework for Integrated Tests) method, 194–195
FitNesse component, 194
Fixed delivery dates in kanban system, 180
Fixed features in Roadmap, 272
Fixed requirements scope assumption, 7

Flow
- architecture principle, 399
- events in use cases, 371, 374–375
- in kanban system, 179–180
- product development, 25–27, 305–307
- rearchitecting with flow. See Rearchitecting with flow

Follow-up in requirements workshops, 232
Forming-Storming-Norming-Performing model, 35
Fowler, Martin, 100
Framework for Integrated Tests (FIT) method, 194–195
FSMs (finite state machines), 361–363
Functional silos, 50
Functional spikes, 114–115
Functional tests, 186

Functionality
- in FURPS acronym, 339
- predicting, 311
- Funnel queues, 410, 412–415, 459–461
- FURPS acronym, 339–340
- Futures, investment in, 85

G
Gat, Israel, 136, 429
Gause, Donald, 238
General availability firewalls, 316
Get it done belief in legacy mind-sets, 434–436
Global optimization, 304
Global WIP, 409
Glossary section in vision documents, 483

Goals
- of estimating, 136–137
- lean software, 22–23
- Scrum Masters/Agile Masters responsibilities, 52
- stretch, 336–337

Good to Great (Collins), 449
Gottesdiener, Ellen, 176
Governance
- architectural, 386, 388
- portfolio management, 430
- rethinking, 442–447
Gregory, Janet, 183, 185–186, 192
Gross estimates, 258
Gross profit, 18
Grouping ideas, 235

Guide to the Business Analysis Body of Knowledge, 278
Gustafsson, Bjorn, 176

H
Hackathons, 398–399
Hardening iterations, 37, 39, 300, 398
Help section in vision documents, 483
Hendrickson, Elizabeth, 194
High, Timothy, 388
High Delay Cost First approach in Cost of Delay, 264–265
Highsmith, Jim, 67
History of software requirements methods, 3–5
  adaptive processes, 12–19
  iterative methods, 9–12
  lean software, 20–28
  waterfall software process model, 5–9
Http Unit testing, 198
Hybrid estimating model, 151–152

I
Ideal developer days (IDDs), 149–151
Ideas in brainstorming
  generation, 233–234
  prioritization, 236–237
  reduction, 235–236
Identifying
  epics, 456–467
  stakeholders, 122–125
IIBA (International Institute of Business Analysts), 278
Impediments
  program level, 64
  in release planning, 333–334
  Scrum Masters/Agile Masters responsibilities, 52
Implementation
  architectural epics, 89–90
  kanban system, 411, 467
  rearchitecting with flow, 423–426
Implementation view in UML, 394
Implementing Lean Software (Poppendieck and Poppendieck), 20
Important category in idea prioritization, 236–237
Imposed standards as constraints, 346–347
Improvement
  kanban system opportunities, 180
  Scrum Masters/Agile Masters responsibilities, 52
Inception phase in RUP, 11
Increased throughput in user stories, 109–110
Incremental delivery, 17, 444–445
Incremental funding, 440–441
Incremental system building, 400
Independence in INVEST model, 106, 110–111
Index cards for user stories, 116–117
Infrastructure enablers, 384
Infrastructure work, backlogs for, 208
Initiative and queue size, 175
Inmates are Running the Asylum, The (Cooper), 127
Innovation as business driver, 388
Innovation principle, 397–399
Installation section in vision documents, 482
Installation Guides section in vision documents, 483
Intangible service class in kanban system, 180
Integration in product development flow, 307
Intentional architecture, 386–387
Interactions, stakeholder, 122
Interface development in user experience, 130
Internal releases, 300
International Institute of Business Analysts (IIBA), 278
Interviews in requirements discovery, 237–240
Introduction section in vision documents, 477
INVEST model attributes for user stories, 105
  estimable, 108–109
  independent, 106
  negotiable, 107
  small, 109–111
  testable, 111
  valuable, 107–108
Investment funding
  portfolio management, 430
  rethinking, 436–440
Investment themes, 33, 43, 383, 450–451
  vs. backlogs, 451–452
  communicating, 451
  overview, 84
Involvement levels of stakeholders, 121
Iron triangle
  eliminating in agile, 16
  in iteration, 160
  waterfall model, 6–8
IS/IT shops, product manager responsibilities in, 278–279
Iteration, 9, 33, 155–156
  backlogs for, 210
  calendars in, 168, 215–216
  commitment, 159–163, 214
  demonstration, 167
demos, 294
executing, 164, 214–215
feature preview, 169
just-in-time story elaboration, 211–212
length, 156
pattern, 157
planning phase, 157–159, 163, 213–214
product owners driving of, 212–216
Rapid Application Development, 10
Rational Unified Process, 11
requirements, 11–12
retrospective, 157, 167, 215
reviews, 167, 215
spiral method, 10
team backlog in, 157–158
team level, 36–37
technical debt and value stream, 216–217
tracking and adjustment in, 164–167
unit testing in, 197–198

J
Jeffries, Ron, 58, 102
Just get it done belief, 434–436
Just-in-time story elaboration, 211–212

K
Kaizen, 24
Kanban system, 28, 179, 408
  analysis, 418–422, 463–466
  architectural epics, 413–415
  backlogs, 411, 415–418, 461–463
  classes of service, 180
  funnel queues, 412–415, 459–461
  implementation, 411, 423–426, 467
  objectives, 408–409
  overview, 409–410, 457–458
  portfolio planning, 456–467
  properties, 179–180
  queue descriptions, 410–411
  state descriptions, 411–413
  state diagram views, 458–459
Kano, Noriaki, 269
Kano model of customer satisfaction, 269–271
Kay, Alan, 251
Key product value propositions, 43
Key User Needs section in vision documents, 479
Knowledge areas, product manager responsibilities in, 277–279
Knowledge in story points, 138
Kolsky, Amir, 191
Kouzina, Olga, 176
Kroll, Per, 377
Kruchten, Philippe, 393

L
Labeling and Packaging section in vision documents, 483
Lambda in Little’s law, 172–173
Larman, Craig, 19, 21–22, 67–68
Last responsible moment (LRM), 253
Lawrence, Richard, 111
Leadership
  in lean software, 24
  product owners, 121
Lean-Agile Software Development: Achieving Enterprise Agility (Shalloway), 20, 256
Lean Software and Systems Consortium, 20
Lean software overview, 20
  continuous improvement, 24
  house of, 20–22
  kanban, 28
management support, 24–25
product development flow, 25–27
respect for people, 23
sustainably delivering value fast goal, 22–23
systems view, 27
*Lean Software Strategies* (Middleton and Sutton), 20
Leffingwell, Dean
acceptance tests, 183
features, 75
finite state machines, 362
*Managing Software Requirements: A Use Case Approach*, 255, 340, 369
requirements vs. design, 385
Legacy mind-sets in portfolio management, 432–436
Length of queue in backlog, 174–175
Length of time in queue in backlog ratings, 417
Licensing, Security, and Installation section in vision documents, 482
Lifecycle phases
product manager responsibilities in, 276–277
RUP, 11
Lightweight business cases
epics, 464–466
investment funding, 438
rearchitecting with flow analysis, 420–421
Limited WIP Society, 179
Linear effect on Lambda, 174
Linear features in Kano model, 270
Little’s law
for agility and time to market, 172–175
in backlog, 171–175
batch size in, 109
in blog story, 170–171
queues, 26
reliability of, 177
Local optimization, 304
Local prioritizations of features, 268–269
Logical view in UML, 394
Logistics in requirements workshops, 229
Low-fidelity interface development, 130
LRM (last responsible moment), 253
Lulls in idea generation, 234

**M**
Maintainability in nonfunctional requirements, 345
Management support, 24–25
Managers’ review meeting in release planning, 328
*Managing Software Requirements: A Use Case Approach* (Leffingwell and Widrig), 255, 340, 369
*Managing the Design Factory* (Reinertsen), 20
Marick, Brian, 185
Market/customer-facing product managers, 206
Market Demographics section in vision documents, 478
Marketing and ART cadence, 315–316
Marketing requirements documents (MRDs), 251–252, 281
Marketing stakeholder guidelines, 123
Martens, Ryan, 70, 390
Martin, James, 10
Martin, Robert, 110, 346
Mean time between failures (MTBF) issues, 344
Mean time to repair (MTTR) issues, 344
Measurability in SMART acronym, 163
Measuring workflow in kanban system, 179–180
Meet-after boards in iteration, 166
Message sequence diagrams (MSDs), 364–365
Metrics review in iteration, 168
Middleton, Peter, 20
Milestones
legacy mind-sets, 434–436
rethinking, 444–445
Minimum guarantees in use cases, 372
Minimum marketable feature (MMF) in Kano model, 270–271
Mitigated risk category in release planning, 334
Modeling principle in architecture, 392–394
Models in kanban system, 180
Monson-Haefel, Richard, 407
Moore, Geoffrey, 3
Motivation and queue size, 175
MRDs (marketing requirements documents), 251–252, 281
MSDs (message sequence diagrams), 364–365
MTBF (mean time between failures) issues, 344
MTTR (mean time to repair) issues, 344
Mugridge, Rick, 194
Must-have features in Kano model, 270

Names in use cases, 370–371, 374
Needs
  compiling, 239–240
  product managers for, 284
  stakeholders, 125–126
Negotiability of user stories, 107
Net ratings for backlogs, 462
New opportunities in business epics, 459
New products
  as business drivers, 387
  investment in, 85
Next + 1 releases, 273
97 Things Every Software Architect Should Know (Monson-Haefel), 407
Nonfunctional requirements (NFRs), 77–78, 339–340
  and backlogs, 78–79, 341, 348
  constraints, 340–341, 345–347
  examples, 342–343
  modeling, 340–342
  performance, 345
  persistent, 347–348
  program level, 40, 77–80
  reliability, 344
  supportability, 345
  templates, 352–354
  testing, 79–80, 348–352
  usability, 343–344
  as user stories, 342
  in vision, 255, 290
Nonfunctional Requirements section in vision documents, 482
Normalizing velocity, 152
Number of iterations per release, 37

Object-based commitment in iteration, 161
Ohno, Taiichi, 22, 319
On-site customers, 201
$100 test, 236
Online brainstorming, 237
Online Help section in vision documents, 483
Opaque transparent firewalls, 316
Operations
  guidelines, 123
  user story splitting pattern, 113
Optimization, 269–271, 304
Order-taker mentality, 433–435
Out-of-the-box thinking in requirements workshops, 230
Outlines in use case flows, 374–375
Outsourced development in rearchitecting with flow implementation, 425
Oversight
  portfolio management, 430
  rethinking, 442–447
Owned risk category in release planning, 334

Partnerships in product manager/product owner team, 296–297
Partner stakeholders, 123
Parkolla, Mikko, 429–430
Parkolla, Mikko, 429–430
Paying down technical debt, 217
Peer ratings for backlogs, 415–416, 461–463
People, respect for, 23
People, Teams, & Culture knowledge area, product manager responsibilities in, 277
Performance
   in FURPS acronym, 340
   nonfunctional requirements, 345
testing, 352
Perpetual Mistrust phase, product manager attitude in, 282–283
Persistent acceptance tests, 190
Persistent nonfunctional requirements, 347–348
Personas, 126
   finding, 127–129
   primary and secondary, 127
Phases of product manager disillusionment, 280–283
Pichler, Roman, 204
Planned release dates in Roadmap, 271
Planning
   centralized annual vs. decentralized rolling-wave, 445–447
   investment funding, 439–440
   in legacy mind-sets, 434–435
   product development flow, 307
   releases. See Release planning
Planning phase in iteration, 157–159, 163, 213–214
Planning poker, estimating with, 139–142, 144–145
Plans of intent, 42, 272, 295
PMBOK (Project Management Body of Knowledge), 442–443
PMO (project management office), 85, 430–431
Pods of agile teams, 33, 35
Policies in kanban system, 180
Poppendieck, Mary, 20, 22–23, 164
Poppendieck, Tom, 20, 22–23, 164
Portfolio backlogs, 383
   as business drivers, 387
   epics in, 43, 86–87, 453
   product development flow, 306
Portfolio level, 33–34
   architectural runways, 88–91
   backlogs, 86–87
   epics, 85–91
   features and stories, 87
   introduction, 83–84, 383–384
   investment themes, 84
   portfolio management team, 85
   summary, 43–44
Portfolio management
   change management, 440–442
   epics identification and prioritization for, 456–467
   governance and oversight, 442–447
   investment funding, 436–440
   legacy mind-sets, 432–436
   overview, 429–431
   project management office, 430–431
   team decisions, 85
Portfolio vision in business epics, 459
Portfolio WIP, 409
Potentially shippable increments (PSIs)
   in ART, 80, 300–301, 314–317
   program level, 33
   in releases, 37, 39–40
   in system building, 401–402
PRDs (product requirements documents), 251–252, 281
Preconditions in use cases, 372, 375
Predictability of releases, 310–313
Predictions in estimates, 136
Predisposition in interviews, 238
Preliminary press release approach in vision, 254
Primary user personas, 127–128
Principles of Agile Architecture, 390–399
   Principles of Product Development Flow
   (Reinertsen), 20, 25, 262
Prioritization
backlog ratings, 417, 462
backlogs, 209–211
epics, 456–467
estimates, 137
features, 261–271
ideas, 236–237
in iteration, 214
product managers for, 285
in Roadmap, 272
Problem solving meeting in release planning, 328
Process control band, 312
Process, Execution, Metrics knowledge area, product manager responsibilities in, 277–278
Process review in iteration, 168
Process view in UML, 394
Product backlogs, 38
Product champions, 201
Product companies, product manager responsibilities in, 276–278
Product councils in requirements discovery, 243–244
Product Development and Management Association, 276
Product development flow, 25–27, 305–307
Product Features section in vision documents, 481
Product Management and Product Owner Role in Large Scale Agile Software Development (Parkolla), 429
Product management in kanban system analysis, 464
Product managers, 33, 74, 275
change management, 286
customer needs, 284
documenting requirements, 284
evolution of, 283
phases of disillusionment, 280–283
prioritizing requirements, 285
product owner collaboration, 220, 295–297
program level, 42–43
rearchitecting with flow analysis, 420
release management, 290–294
responsibilities, 276–280, 286–287
Roadmap, 295
scheduling, 285
status assessment, 287
validating requirements, 286
vision, 288–290
Product Overview section in vision documents, 480
Product owners, 33
attributes, 218–220
backlog management, 208–211
bottlenecks, 221
communication with, 294
CSG Systems, 223
description, 120
Discount Tire, 224
dual roles, 207
emergence of, 201–202
iteration driving, 212–216
iteration planning meeting preparation responsibilities, 159
just-in-time story elaboration, 211–212
product manager collaboration, 220, 295–297
proxies, 221
release planning, 217–218
responsibilities, 51, 202–208
role/title, 206–207
roles, 36
Scrum, 15
Symbian Software, 223–224
teams, 221
technical debt and value stream, 216–217
TradeStation Technologies, 222–223
Product Position Statement section in vision documents, 480–481
Product requirements documents (PRDs), 251–252, 281
Product/technology councils in rearchitecting with flow analysis, 422
Production requirements workshops, 232
Productivity in usability, 350
Program backlogs
features for, 76
product development flow, 306
Program level, 33
architectural epics, 90
ART, 80–82
introduction, 63–64
nonfunctional requirements, 77–80
product managers, 42–43
program managers, 275
release planning, 41
releases and PSIs, 39–40
Roadmap, 41–42, 64
summary, 38–39
teams. See Teams and team level vision, features, and program backlog, 40–41
Programming support, acceptance tests for, 185
Progress tracking in iteration, 164–167
Project backlogs, 38
Project-free, continuous content delivery, 441–442
Project Management Body of Knowledge (PMBOK), 442–443
Project management office (PMO), 85, 430–431
Project Management tool, iteration tracking with, 166–167
Project plans in investment funding, 438
Project-specific information for requirements workshops, 230
Project stakeholders, 120
Proxies, product owner, 221
Pruning ideas, 235
Pseudocode, 358–359
PSIs (potentially shippable increments) in ART, 80, 300–301, 314–317
program level, 33
in releases, 37, 39–40
in system building, 401–402
Purchased solutions for rearchitecting with flow implementation, 425–426

Q
Qualities, testing, 79–80, 186–187
Quality
agile, 17
predicting, 310–311
and queue size, 175
unit tests, 60
waterfall model, 8
Quality assurance
QA and deployment teams, 71
responsibilities, 54
system, 72
Quality Function Deployment technique, 245
Quality management at program level, 64
Quantitative review in iteration, 168
Questionnaires in requirements discovery, 240–241
Questions in interviews, 238–239
Queues. See also Backlogs
analysis, 418–422
backlog blog story, 170–171
funnel, 410, 412–415, 459–461
in kanban system, 410–411
lean software, 25–26
length, 174–175, 177
in product development flow, 306

R
Rally Software Development, 90, 245–246
Ranking
architectural epics, 414–415
business epics, 460
Rapid Application Development (RAD), 10
Ratings for backlogs, 415–417, 461–463
Rational Unified Process (RUP), 11, 252–253, 367, 393
Realized features, 76
Rearchitecting system portions, 387
Rearchitecting with flow, 407–408
analysis, 418–422
architectural epics, 413–415
backlogs, 415–418
funnel queues, 412–415
implementation, 423–426
kanban system, 408–412
Recall in usability, 350
Refactors, backlogs for, 208
References section in vision documents, 478
Refining use cases, 375
Regulations as constraints, 346–347
Reinertsen, Donald G.
on AWIP, 408
batch size, 299
Managing the Design Factory, 20
Principles of Product Development Flow, 20, 25, 262
prioritizing work, 263
on queues, 155, 175
Relationships in use cases, 374
Relative estimating, 138–139, 145–147, 258
Release backlogs in product development flow, 306
Release levels, program managers at, 275
Release management
product managers for, 290–294
program level, 64
Release management teams (RMTs), 73–74, 294
Release planning, 299
ART, 80–81, 308–309
attendance, 320
checklist, 321–322, 485–488
Day 1, 322–323
architecture vision, 324
business context, 323–324
draft plan reviews, 327–328
managers’ review and problem solving meeting, 328
opening meeting, 323
team planning breakouts, 325–327
Day 2, 328–329
adjustments, 330
commitment, 334–335
final instructions to teams, 336
final review, 332–333
opening meeting, 330
release objectives, 330–332
retrospective, 335–336
risks and impediments, 333–334
solution vision, 324
team planning breakouts, 330
domains, 320
facilitators, 320–321
preparing for, 319
product manager role, 290–293
product owner role, 217–218
program level, 41
stretch goals, 336–337
Release teams, 71
Release train domains, 308
Releases, 33, 39–40
ART. See Agile Release Train (ART)
commitment, 292–293
frequency, 313–317
managing, 309–310
planning. See Release planning
predictability, 310–313
PSIs, 80
retrospective, 310
Roadmap for, 273
tracking, 293–294, 309–310
Relevancy in SMART acronym, 163
Reliability
in FURPS acronym, 340
nonfunctional requirements, 344
testing, 350–351
Reliable estimates, 136
Reports, requirements analysis, 358
Requirements
backlog meta-model, 489
nonfunctional. See Nonfunctional requirements (NFRs)
user stories as, 101–102
Requirements analysis, 355–356
  activity diagrams, 357
  decision tables and decision trees, 359–361
  entity-relationship diagrams, 365
  finite state machines, 361–363
  message sequence diagrams, 364–365
  pseudocode, 358–359
  sample reports, 358
  use case modeling, 366
Requirements Analysis knowledge area,
  product manager responsibilities in, 279
Requirements analysts, 206
Requirements architects, 206
Requirements discovery, 227–228
  brainstorming, 232–237
  competitive analysis, 244–245
  customer change request systems, 245–246
  defect logs, 246
  interviews, 237–240
  product councils, 243–244
  questionnaires, 240–241
  use case modeling, 247
  user experience mock-ups, 241–243
  workshops, 228–232
Requirements Management and
  Communication knowledge area,
  product manager responsibilities in, 279
Requirements view in UML, 393
Research and development, 267
Resetting expectations phase, product
  manager attitude in, 282
Resolved risk category in release planning, 334
Resource management at program level, 64
Resource utilization in performance, 345
Respect for people, 23
Response time in performance, 345
Responsibilities
  product managers, 42–43, 276–280
  product owners, 51, 202–208
  teams, 50–54
Resynchronization, 26
Retrospective
  iteration, 157, 167, 215
  release planning, 335–336
  releases, 310
Return on investment (ROI)
  incremental value delivery for, 17
  for prioritizing features, 261–263
Review in iteration, 157, 167, 215
Risks and risk reduction
  in backlog ratings, 417
  backlogs for, 210
  in Cost of Delay, 267
  and queue size, 175
  in release planning, 333–334
RMT (release management team), 73–74, 294
Roadmaps, 271–272
  for architectural epics, 413
  description, 81–82
  product managers for, 295
  program level, 41–42, 64
  for releases, 273
ROAM categories in release planning, 334
ROI (return on investment)
  incremental value delivery for, 17
  for prioritizing features, 261–263
Role collaboration principle
  rearchitecting with flow analysis, 419–420
  system architecture, 396–397
Roles
  product owners, 36, 207
  teams, 36, 50–54
  in user stories, 104
Rolling-wave planning, 445–447
Roof in lean software, 22–23
Royce, Winston, 5–6
Rude Awakening phase, product manager
  attitude in, 281
Rules, responsibility for, 52
Runways, architectural, 44, 88–91, 383,
  395–396
RUP (Rational Unified Process), 11, 252–253,
  367, 393
S
SaaS (software as a service) application, 90, 314
Sales stakeholder guidelines, 123
Sample reports in requirements analysis, 358
SATs (story acceptance tests), 187–190
Scalability in performance, 345
Scale, team organization at, 64
Scaling Lean and Agile Development (Larman and Vodde), 19
Scaling Software Agility (Leffingwell), 16, 19, 65, 88, 387
Scheduled features, 290
Schedules
estimating, 137, 149
myths, 135
product managers for, 285
and velocity, 148
Schwaber, Ken, 202, 221
Scope
in architectural epics, 384
in business epics, 453
of releases, 290
story points for, 138
in waterfall model, 7
Scotland, Karl, 176
Scrum/Agile Masters, 15, 34
responsibilities, 51–52
roles, 36
Scrum Alliance, 15
Scrum Certified Product Owner courses, 204
Scrum of Scrums planning checkpoints, 326–327
Scrum project management method, 15
product owners, 202–204
story of, 24
SDS (system design specification), 281
Second degree system stakeholders, 125
Secondary user personas, 127–128
Security
in reliability, 344
testing, 351–352
Security section in vision documents, 482
Security stakeholder guidelines, 124
Sense of balance in product manager/product owner teams, 296
Separation of development from releases and marketing, 315–316
Sequencing in prioritizing features, 262–263
Shakespeare, William, 63, 99, 135, 183, 201
Shalloway, Alan, 20, 256
Shortest Job First in Cost of Delay, 263–264
Silos, functional, 50
Simplicity in architecture, 391–392
Size
backlog ratings, 417
queue, 174–175
in user stories, 110–111
Sliger, Michele, 443
Small attribute in INVEST model, 109–111
SMART acronym, 162–163
Smart energy grids. See Tendril case study
Software as a service (SaaS) application, 90, 314
Software by Numbers (Denne and Cleland-Huang), 262
Software kanban. See Kanban system
Software project management's Bridge to Agility (Sliger), 443
Software Requirements (Wiegers), 262
Software requirements history, 3–5
adaptable processes, 12–19
iterative methods, 9–12
lean software, 20–28
waterfall software process model, 5–9
Software requirements specifications (SRSs), 251–252
Solely emergent architectures, 386
Solution Assessment and Validation
knowledge area, product manager responsibilities in, 279
Solution management, 276
kanban system analysis, 464
rearchitecting with flow analysis, 420
Solution/product/technology-facing product owners, 206
Solution vision in release planning, 324
Solutions-context questions in interviews, 238–239
Sources
architectural epics, 413–414
business epics, 459–460
Special requirements in use cases, 372
Specialist responsibilities, 54
Specificity in SMART acronym, 163
Spikes, 114
guidelines, 115–116
technical and functional, 114–115
user experience, 130
Spiral model of development, 10
Splitting
architecture epics, 403–405
user stories, 111–114
Sponsor guidelines, 124
Sprint status boards, 165
Sprints, Scrum, 15, 36
SRSs (software requirements specifications), 251–252
Stakeholder/product owner teams, 126
Stakeholders, 119
identifying, 122–125
interactions, 122
interviews, 237–238
involvement levels, 121
kanban system analysis, 464
product owners, 120
project, 120
requirements workshops, 229
system, 120
trust, 122
in use cases, 372
Stakeholders section in vision documents, 480–481
Standard service class in kanban system, 180
Standards as constraints, 346–347
Standish Group, 6
Starbucks backlog blog story, 170–171
State descriptions in kanban system, 411–413
State diagram views, 458–459
State transition diagrams, 362–363
Statements of intent, 38
Steering committees, 73
Sterling, Chris, 177
Stories
vs. features and epics, 454–456
just-in-time story elaboration, 211–212
portfolio level, 87
user. See User stories
Story acceptance tests (SATs), 59, 187–190
Story points, 138, 257, 259
Story spikes in user experience, 130
Story time in iteration, 169
Strategic alignment in ART, 304–305
Strategic intent in epics, 44
Strategic investment themes, 450
Strategy, Planning, and Decision Making
knowledge area, product manager
responsibilities in, 277
Strategic product themes, 450
Stretch goals in release planning, 336–337
Stretch objectives in iteration, 160
Subepics, 453
Subjective review in iteration, 168
Subsystems in use cases, 368, 372
Succeeding with Agile: Software Development Using Scrum (Cohn), 219
Success guarantees in use cases, 372
Summary of Capabilities section in vision documents, 481
Support stakeholder guidelines, 124
Supportability
in FURPS acronym, 340
nonfunctional requirements, 345
testing, 352
Supporting roles, 54
Sustainably delivering value fast goal, 22–23
Sutton, James, 20
Symbian Software, 223–224
Synchronization

- lean software, 26
- product development flow, 307

System acceptance tests, 186
System architecture, 383
System context diagrams, 95
System design specification (SDS), 281
System qualities
  - tests, 80, 186–187
  - user story splitting pattern, 113
  - in vision, 255
System Requirements section in vision
documents, 482
System stakeholders, 120
  - classifying, 125
  - identifying, 124
  - needs of, 125–126
System teams, 71–73
Systems architecture, 384–385
  - architectural epics, 387–388, 399–403
  - business drivers, 387–388
  - emergence of, 385–386
  - intentional, 386–387
  - system architect role, 388–390
Systems in use cases, 368, 370, 372
Systems view in lean software, 27

T
Tabletop relative estimation, 145–147
Task-based commitment in iteration, 162–163
Tasks, team level, 38, 57–58
TDD (test-driven development), 60, 190–191
Team backlogs
  - in iteration, 157–158
  - product development flow, 306
  - user stories in, 38, 55–56
Team-based estimating, 259
Teams and team level
  - acceptance tests, 58–59
  - architectural epics, 90–91
  - coding by, 390–391
  - features, 34–35
  - and functional silos, 50
  - importance, 47–49
  - iterations, 36–37
  - overview, 34–35
  - planning breakouts, 325–327, 330
  - pods, 35
  - product owners, 221
  - program level
    - co-location, 71
    - component, 65–66, 68–71
    - features, 66–71, 75–77
    - organization at scale, 64
    - product management, 74
    - release management, 73–74
    - system, 71–73
    - vision, 74–75
  - rearchitecting with flow implementation, 424, 426
  - roles and responsibilities, 33, 36, 50–54
  - tasks, 38, 57–58
  - user stories, 37–38, 56–57
Technical debt, 39
  - backlogs for, 208
  - iteration, 216–217
Technical foundation of product owners, 219
Technical spikes, 114–115
Technological changes as business driver, 387
Technology and Intellectual Property
  - knowledge area, product manager
    - responsibilities in, 277
Technology-facing tests, 185
Technology Roadmaps, 413
Templates
  - business cases, 420–421, 464–466
  - nonfunctional requirements, 352–354
  - vision documents, 475–477
Temporal prioritizations of features, 268–269
Tendril case study, 93
  - activity diagrams, 357
  - background, 93–94
  - sample reports, 358
  - system context diagrams, 95
system stakeholders, 125–126
use case example, 375–377
user personas, 128–129
Tendril end-to-end solution, 94
Test-driven development (TDD), 60, 190–191
Testability of user stories, 111
Tester roles and responsibilities, 36, 53
Testing principle in architecture, 395
Tests
acceptance. See Acceptance tests
in define/build/test sequence, 49
features, 77, 260–261
nonfunctional requirements, 79–80, 348–352
performance, 352
reliability, 350–351
security, 351–352
supportability and design constraints, 352
system-level, 72
system qualities, 80
teams, 58–60
usability, 350
user stories, 188–189
Themes
investment, 33, 43, 84, 383, 450–452
planned release dates, 271
Third degree system stakeholders, 125
Thomas, M., 7
Throughput
and backlog queue length, 177
in performance, 345
in user stories, 109–110
Time
in architectural epics, 384
in business epics, 453
Time to market, Little’s law for, 172–175
Time value
in backlog ratings, 417, 462
backlogs for, 210
in Cost of Delay, 267
Timeboxed tasks in SMART acronym, 163
Toyota Production System (TPS), 20
Toyota Way, The (Liker), 20
Tracking
in iteration, 164–167
releases, 293–294, 309–310
TradeStation Securities trading system, 68–69
TradeStation Technologies product owners, 222–223
Training time, 343
Transitions phase in RUP, 11
Transparent firewalls, 316
Trust
product manager/product owner team, 296–297
in software development, 386
stakeholders, 122
Trustworthiness of product owners, 219
Tuckman, Bruce, 35
Tyranny of the urgent iteration, 169
U
UML (Unified Modeling Language), 393–394
Unambiguous acceptance tests, 189
Unbridled enthusiasm phase, product manager attitude in, 281
Uncertainty
in product development flow, 307
story points, 138
Undelivered features, 40–41
Unified Modeling Language (UML), 393–394
Unit tests
in agile testing matrix, 186
in iteration, 197–198
overview, 196–197
purpose, 60
United management fronts for releases, 330
Unscheduled features, 290
Usability
in FURPS acronym, 339
nonfunctional requirements, 343–344
testing, 350
Use-case view in UML, 393
Use cases
  applying, 377–378
  basics, 369–370
  benefits, 368–369
  example, 375–377
  model building for, 372–375
  overview, 367–368
  requirements analysis, 366
  requirements discovery, 247
  requirements model, 378–379
  structure, 370–371
  user story splitting pattern, 113
  vision documents, 482
Useful category in idea prioritization, 236–237
User Description section in vision documents, 478–479
User Environment section in vision documents, 479
User experience (UX) design, 129–130
  centralized development, 131–132
  distributed but governed, 131–133
  interface development, 130
  story spikes, 130
User experience mock-ups in requirements discovery, 241–243
User interaction, use cases for, 368
User interviews in requirements discovery, 237–238
User Manual section in vision documents, 483
User/Market Demographics section in vision documents, 478
User personas, 126
  finding, 127–129
  primary and secondary, 127
User Personas section in vision documents, 478
User stories, 33
  acceptance criteria, 104–105
  backlogs for, 208
  basics, 57
  cards, conversation, and confirmation, 102–103, 116–117, 121
detail, 104
  for developer-customer communication gap, 101
  introduction, 99–100
  INVEST model, 105–111
  nonfunctional requirements as, 342
  overview, 100–101
  problems with, 368
  as requirements, 101–102
  size and Lambda, 174
  spikes, 114–116
  splitting, 111–114
  teams, 37–38, 56–57
  testing, 188–189
  for user personas, 127–129
  voice, 103–104, 257
User Stories Applied (Cohn), 100, 103
User value
  in architectural epics, 384
  in backlog ratings, 417
  backlogs for, 210
  in Cost of Delay, 266–267
User voice
  features, 257
  user stories, 103–104
Utilization
  in legacy mind-sets, 434–435
  in performance, 345
V
Validating requirements, product managers for, 286
Value
  in architectural epics, 384
  in backlog ratings, 417, 462
  in Cost of Delay, 266–267
  of estimates, 137
  in INVEST model, 107–108
  in prioritizing features, 269–271
in release objectives, 331–332
in user stories, 104
Value chain, 23
Value/effort ROI proxy, 262
Value stream in iteration, 216–217
Variability
  lean software, 26
  product development flow, 306
  and queue size, 175
Variable features in Roadmap, 272
Velocity, 136
  establishing, 146–149
  feature estimates, 257
  increasing, 148
  normalizing, 152
  schedule and cost connections, 148
Velocity-based commitment in iteration, 160–162
Views in UML, 393–394
Vision
  communicating, 289
  data sheet approach, 253–254
  feature backlog with briefing approach, 255
  nonfunctional requirements, 255
  overview, 251–252
  portfolio, 459
  preliminary press release approach, 254
  product managers for, 288–290
  program level, 40, 64
  release planning, 324
  teams, 74–75
  vision documents, 252–253
Vision document template, 475–477
  Assumptions and Dependencies section, 481
  Documentation Requirements section, 483
  Exemplary Use Cases section, 482
  Glossary section, 483
  Introduction section, 477
  Nonfunctional Requirements section, 482
  Product Features section, 481
  Stakeholders section, 480–481
  User Description section, 478–479
Vodde, Bas, 19, 21–22, 67–68
Vogels, Werner, 392
Voice
  features, 257
  user stories, 103–104
Volume of story points, 138
Von Neumann, John, 355
W
Wake, Bill
  INVEST acronym, 105
  SMART acronym, 162–163
  on user stories, 101
Warm-up materials in requirements workshops, 230
Waterfall model, 5–6
  current use, 8–9
  problems, 6
  requirements, 6–8
Weighted Shortest Job First (WSJF) prioritization
  backlog ratings, 417
  in Cost of Delay, 265–267
  epic ratings, 462
  for features, 290
  for requirements, 285
Weinberg, Gerald, 238
White-box testing, 351
Widget engineering mind-set, 433–435
Widrig, Don, 227
  acceptance tests, 183
  Managing Software Requirements: A Use Case Approach, 255, 340, 369
Wiegner, Karl E., 262
Wikis, 348
Wireframe models, 242–243
Work breakdown structure in investment funding, 439–440
Work-in-process (WIP)
  backlog ratings, 418, 463
  business epics, 461
  funnel queues, 415
  in kanban system, 179, 464
  lean software, 26
  product development flow, 306–307
  rearchitecting with flow, 420, 426
Work items, 56
Workflow
  kanban system, 179–180
  user story splitting pattern, 112
Working software in systems architecture, 386
Workshops, requirements, 228–232
Write-the-test-first approach, 111
Writing Effective Use Cases (Cockburn), 369
WSJF. See Weighted Shortest Job First (WSJF) prioritization

X
XML Unit testing, 198
XP (Extreme Programming), 14–15, 100

Z
Zamora, Mauricio, 223, 390