SOFTWARE
ENDGAMES
Also Available from Dorset House Publishing

**Agile Software Development in the Large:**
*Diving Into the Deep*
by Jutta Eckstein

**Best Practices for the Formal Software Testing Process:**
*A Menu of Testing Tasks*
by Rodger D. Drabick foreword by William E. Perry

**The Deadline: A Novel About Project Management**
by Tom DeMarco

**Five Core Metrics: The Intelligence Behind Successful Software Management**
by Lawrence H. Putnam and Ware Myers

**Hiring the Best Knowledge Workers, Techies & Nerds:**
*The Secrets & Science of Hiring Technical People*
by Johanna Rothman foreword by Gerald M. Weinberg

**Peopleware: Productive Projects and Teams, 2nd ed.**
by Tom DeMarco and Timothy Lister

**Project Retrospectives: A Handbook for Team Reviews**
by Norman L. Kerth foreword by Gerald M. Weinberg
ISBN: 0-932633-44-7 Copyright ©2001 288 pages, softcover

**Waltzing with Bears: Managing Risk on Software Projects**
by Tom DeMarco and Timothy Lister
ISBN: 0-932633-60-9 Copyright ©2003 208 pages, softcover

---

For More Information

✔ Contact us for prices, shipping options, availability, and more.
✔ Sign up for DHQ: The Dorset House Quarterly in print or PDF.
✔ Send e-mail to subscribe to e-DHQ, our e-mail newsletter.
✔ Visit Dorsethouse.com for excerpts, reviews, downloads, and more.
SOFTWARE ENDGAMES

Eliminating Defects, Controlling Change, and the Countdown to On-Time Delivery

Robert Galen

DORSET HOUSE PUBLISHING
353 West 12th Street
New York, New York 10014
To all of my “Bugs”...
This page intentionally left blank
Many of the lessons I bring forward in the text were hard-won, through the many endgames I’ve been a part of, at numerous companies and with many teams. I want to thank everyone who has been a part of my endgames. The lessons were sometimes difficult, not always resulting in success. However, for me, one constant throughout has been the privilege of working with so many dedicated, talented, and insightful individuals. In particular, lessons I’ve derived from teams at Bell & Howell MMT, EMC, Micrognosis, and WWG/Acterna have been central to many of this book’s themes. Thank you all for your effort, patience, trust, and above all, perseverance through some very challenging software project endgames.

I’ve come to realize that a book is a difficult project in itself. It’s not solely the author’s effort. It requires support and help from a wide variety of individuals. It’s also an incredible amount of work. I want to warmly thank those individuals who reviewed chapters in the book and provided feedback and council. They include Terry Bradley, Erik Hemdal, and Carla Merrill. Of particular note are Trish Gertner and Robert Sabourin. They provided incredibly valuable insight and feedback on the entire book. I thank you both for your time, thoroughness, and effort.
ACKNOWLEDGMENTS

Another realization was just how important the publisher and editorial staff are in the creative process. I’ve been truly blessed with a talented and dedicated group of editors at Dorset House. My appreciation to David McClintock, Vincent Au, and Nuno Andrade for their tireless effort and collaboration in helping to refine raw thoughts into a book and taking the time to “get it right.” Thank you.

Finally, I must thank my family for their enthusiastic support. Kids, thanks for your kind words, interest, and faith. Diane, the book started out as quite a rocky path, simply a gleam in my eye and a dream. Your staunch belief and continuous support made all the difference in getting me through the hard times and to get it on paper. Sometimes, it’s not the little or the big things that matter, but the constant things that are most important. I couldn’t have had a better partner throughout the effort—thank you for your faith and continuous support.
Contents

**Preface**  
My Motivation for the Book  
Intended Audience  
How to Approach This Book  
What We’re Not Trying to Explore

<table>
<thead>
<tr>
<th>Chapter One: Introduction</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>You Know You’re in the Endgame When . . .</td>
<td>4</td>
</tr>
<tr>
<td>You Know You’ve Exited the Endgame When . . .</td>
<td>5</td>
</tr>
<tr>
<td>Focus of the Book: It’s Mostly About Defects</td>
<td>5</td>
</tr>
<tr>
<td>Triage Workflow</td>
<td>6</td>
</tr>
<tr>
<td>Dynamics of Software Development</td>
<td>7</td>
</tr>
<tr>
<td>Endgame Workflow</td>
<td>8</td>
</tr>
<tr>
<td>Repair Triage and Change Control Workflows</td>
<td>8</td>
</tr>
<tr>
<td>Phase 1: Initial Defect Analysis (Low-Level Triage)</td>
<td>8</td>
</tr>
<tr>
<td>Phase 2: High-Level Triage and Change Control Decision-Making</td>
<td>9</td>
</tr>
<tr>
<td>Phase 3: Communication</td>
<td>10</td>
</tr>
<tr>
<td>Phase 4: Package Analysis</td>
<td>11</td>
</tr>
<tr>
<td>Phase 5: Repair Scheduling</td>
<td>11</td>
</tr>
<tr>
<td>Phase 6: Construction (Repairs)</td>
<td>12</td>
</tr>
<tr>
<td>Phase 7: Release and Verification</td>
<td>12</td>
</tr>
<tr>
<td>Flow of the Book</td>
<td>12</td>
</tr>
</tbody>
</table>
CONTENTS

Part One: Endgame Basics 17

Chapter Two: Triage and Change Control Process and Meeting Framework 19

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Different Names and Formats</td>
<td>20</td>
</tr>
<tr>
<td>Formal CCB Format</td>
<td>21</td>
</tr>
<tr>
<td>What's Bad About a Formalized CCB?</td>
<td>21</td>
</tr>
<tr>
<td>What's Useful About a CCB?</td>
<td>22</td>
</tr>
<tr>
<td>A Lightweight CCB (LW-CCB) Explored</td>
<td>22</td>
</tr>
<tr>
<td>Guidelines for Defect Triage Outside of the CCB Meeting</td>
<td>23</td>
</tr>
<tr>
<td>Determining Meeting Frequency</td>
<td>24</td>
</tr>
<tr>
<td>A Quick Look at the Meeting Flow</td>
<td>25</td>
</tr>
<tr>
<td>Meeting Participant Roles</td>
<td>26</td>
</tr>
<tr>
<td>Setting Expectations</td>
<td>27</td>
</tr>
<tr>
<td>Don't Get Stuck</td>
<td>27</td>
</tr>
<tr>
<td>Setting Up Meeting Dynamics</td>
<td>28</td>
</tr>
<tr>
<td>Meeting Assistance</td>
<td>28</td>
</tr>
<tr>
<td>Establishing Norms</td>
<td>28</td>
</tr>
<tr>
<td>Feelings: Good and Bad CCB Meetings</td>
<td>30</td>
</tr>
<tr>
<td>A Bad Example</td>
<td>31</td>
</tr>
<tr>
<td>Triage Meeting—Agenda Format</td>
<td>32</td>
</tr>
<tr>
<td>Step 1: Functional Roundtable</td>
<td>33</td>
</tr>
<tr>
<td>Step 2: Old Work Review</td>
<td>33</td>
</tr>
<tr>
<td>Step 3: New Work Planning</td>
<td>33</td>
</tr>
<tr>
<td>Step 4: Examine Trending</td>
<td>34</td>
</tr>
<tr>
<td>Step 5: Adjustments to Release Criteria</td>
<td>34</td>
</tr>
<tr>
<td>Step 6: Meeting Follow-Up</td>
<td>35</td>
</tr>
<tr>
<td>Step 7: Meeting Metrics</td>
<td>35</td>
</tr>
<tr>
<td>Decision-Making</td>
<td>36</td>
</tr>
<tr>
<td>Convergent Versus Divergent Thinking</td>
<td>36</td>
</tr>
<tr>
<td>Participatory Decision-Making: Core Values</td>
<td>37</td>
</tr>
<tr>
<td>Decision-Making Models</td>
<td>38</td>
</tr>
<tr>
<td>Guiding Team Decisions</td>
<td>40</td>
</tr>
<tr>
<td>Wrap-Up: Additions to Your Endgame Toolbox</td>
<td>40</td>
</tr>
</tbody>
</table>

Chapter Three: Developing Release Criteria and Working Views 42

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release Criteria or Defining Success</td>
<td>43</td>
</tr>
<tr>
<td>Step 1: Define Success</td>
<td>43</td>
</tr>
<tr>
<td>Step 2: Learn What's Important for This Project</td>
<td>43</td>
</tr>
<tr>
<td>Step 3: Draft Release Criteria</td>
<td>44</td>
</tr>
<tr>
<td>Step 4: Make the Release Criteria SMART</td>
<td>44</td>
</tr>
<tr>
<td>Step 5: Achieve Consensus on the Release Criteria</td>
<td>45</td>
</tr>
<tr>
<td>How to Use Release Criteria</td>
<td>45</td>
</tr>
</tbody>
</table>
CONTENTS

Better Decisions: Aspects of a Project Working View 46
The Problem 46
Definitions 46
Dimension Expansion 48
Fixed Versus Variable Dimensions 49
Steps to Establishing a Working View 49
Step 1: Identify Your Project Stakeholders 49
Step 2: Set the Stage 49
Step 3: Project Vision, Essence, and Release Criteria 50
Step 4: Explore Product and Project Dimensions 51
Examples of Working View Application Scenarios 52
Project Application: Example #1 54
The Problem 54
The Solution 54
Participants 54
Dimensions of the Problem 54
Ranking 55
Worded Working View 55
Agreement 55
Project Application: Example #2 56
The Problem 56
The Solution 56
Participants 56
Dimensions of the Problem 56
Ranking 57
Worded Working View 57
Agreement 58
Example Results 58
Project Application: Example #3, Another Approach 59
Resetting Your Working View 59
Wrap-Up: Additions to Your Endgame Toolbox 60

Chapter Four: Endgame Release Framework 62
Overview of the Framework 64
Gathering Input from the Team 65
Input from Software Development 65
Input from Testing 66
Input from Other Team Members 68
Mapping the Data Into the Framework 69
A Word on Strategy 70
Robert Sabourin’s Rule of Thumb for Test Estimation 71
Default Strategy 71
Release Framework Example 72
CONTENTS

Release Framework: Historical Notes 72
Release Framework: Plan Details 73
Release Framework: Endgame Flow 73
Measuring Progress for the Release Framework 75
Ownership of the Release Framework 75
Updating Your Release Framework 76
The Impact of Granularity on Methodologies 76
Wrap-Up: Additions to Your Endgame Toolbox 77

Chapter Five: Reducing the Rate of Change 78
Change Reduction Milestones for the Endgame Framework 79
The Notion of a Code Freeze 80
Microsoft—Code Complete 81
Beyond Code Freeze and Code Complete—Defect Repair Change Cycles 83
Change Reduction Anti-Patterns 85
  Anti-Pattern: Delaying Code Freeze 85
  Anti-Pattern: Ad Hoc Testing Delays 87
  Anti-Pattern: Inherent Instability (Architecture, Infrastructure, or Performance) 88
  Anti-Pattern: Never-Ending Rework 89
  Anti-Pattern: Feature Creep or Requirements Changes 91
  Anti-Pattern: Fluctuations in Release Criteria 92
  Anti-Pattern: Inability to Complete Repairs and/or Run Tests 93
  Anti-Pattern: Customer Introduction Too Early or Too Late (Alpha, Beta, or Demo) 95
Anti-Pattern Wrap-Up 96
How the CCB Assists in Change Reduction 96
Wrap-Up: Additions to Your Endgame Toolbox 97

Chapter Six: Configuration Management in the Endgame 98
Configuration Management Readiness Checklist 99
Level of CM Control Granularity 100
Early On—Run a Practice Build Test 101
Beware of Changing Tools in the Endgame 101
The Role of Team Leaders in Configuration Management 102
  Development Lead 102
  Test Lead 103
  Periodic (Nightly) Builds and Smoke Testing 104
    Build Frequency 104
    Smoke Testing 105
  Release Turnover Meetings 107
Wrap-Up: Additions to Your Endgame Toolbox 108

Part Two: Endgame Defects 109

Chapter Seven:
Defect Basics: Terms, Tools, Methods, and Management 111
CONTENTS

Fundamental Defect Types 112
  Duct Tape: Customer Perceptions 113
  Internal Versus External Enhancements 114
Basic Defect Data Fields 114
  Quick Field Annotations: External Data 115
  Quick Field Annotations: Internal Data 117
  Resolution Summary—Expanded 118
  Where Found 119
  Functional Area 121
  Severity and Priority 122
Defect Evolution 129
  E-Mail Notification 131
Introduction to Work Queues 131
  Phase 1 132
  Phase 2 132
  Phase 3 132
Information Weight 133
Using Defects to Track Development (and Other) Work 134
Frequent Defect Monitoring—What Needs Attention 135
  Defect Arrival and Closure Rates 135
  Defect Assignment Times 135
  Overall Defect Repair Times 136
  Alternate Defect Transitions 136
The Idea of Bug Advocacy 136
Wrap-Up: Additions to Your Endgame Toolbox 137

Chapter Eight: Useful and Relevant Metrics 138
Find (New) Versus Fixed (Closed, Deferred) 139
  Points to Observe 139
  Trends to Watch Out For 140
Factoring in Priority 141
  Points to Observe 141
  Trends to Watch Out For 142
Factoring In Keywords 144
  Points to Observe 144
  Trends to Watch Out For 146
Defect Transition Progress 146
  Points to Observe 147
  Trends to Watch Out For 148
Functional Areas and Defect Distribution 149
  Points to Observe 149
  Trends to Watch Out For 150
  Pareto Analysis 151
Chapter Nine: The Many Ways to Fix Defects

Just Repair It

Make a Partial Repair to Reduce Impact and Severity

Log It As a Known Defect and Move On

Simply Change the Severity or Priority

Ignore It and Hope It Goes Away . . .

Consider It Anomalous Behavior and Wait for the Next Occurrence

Defer the Repair to a Later Release

Negotiate with Marketing or the Customer to Change the Requirement

Add More System Resources to Reduce the Impact

Find a Workaround (Procedural, Operational, Documentation, or Automated)
CONTENTS

Caution—Use Workarounds Sparingly 173
Remove the Functionality/Code 173
  Decision Factors 173
  Example Problem 174
Change or Remove Interacting Software Products 174
  Decision Factors 175
Wrap-Up: Additions to Your Endgame Toolbox 175

Part Three: Endgame Workflow 177

Chapter Ten: Work Queues and Packaging 179
  Work Queues 180
  Deriving Work Queues from Your Defect-Tracking System 182
  Queue Loading Rules 184
    Changing Queue Loading 185
  How Should Testing Interact with the Work Queues? 185
  Deal with Defect Repairs in Packages 186
    Package Themes 187
    Package Strategy 190
  Thinking About Package Costs 190
  Think About Your Regression Trends 191
  Package Plan Status and General Replanning 191
  Wrap-Up: Additions to Your Endgame Toolbox 193

Chapter Eleven: Defect Repair Selection: Other Considerations 194
  Reproducing the Defect 195
    Bug Isolation 195
  Overall Level of Difficulty 196
  Locality and Relationship to Other Defects 198
  How Will the Repair Impact the Test Team? 199
    Effects on Test Automation 200
  Exploring Possible Workarounds 201
  Handling Gold Features 201
  Considering Your Available Resources 203
  Likelihood That the Defect Will Surface in Normal Operation of the Product 206
  Wrap-Up: Additions to Your Endgame Toolbox 207

Chapter Twelve: Endgame Estimation: A Few Useful Collaborative Techniques 208
  Defect Estimation Life Cycle 209
  PSP PROBE Method–Proxy-Based Estimation 211
    Endgame Applications 213
  Wideband Delphi Method 214
    Endgame-Modified Wideband Delphi 215
    Endgame Applications 216

xv
CONTENTS

Other Estimation Techniques 217
  *Endgame Applications* 218
  *A Quick Example* 218
Collaborative Estimation—What to Collect 220
  *Estimate Data* 220
  *Peripheral Estimate Data* 221
  *Related Data* 221
Wrap-Up: Additions to Your Endgame Toolbox 222

Part Four: Endgame Management 223

Chapter Thirteen: Management Dynamics 225
  *The Importance of Team Leads: Clear Roles and Responsibilities* 226
    *Mining for Team Leaders* 227
    *Team Sizes* 228
    *The Sign-Up* 228
  *The Tone of the Endgame* 229
    *Energy Gaps or a Lack of Focus* 231
    *Communication Gaps* 232
    *The Team Has Lost Its Sense of Feasibility* 232
    *Contention and Conflict* 232
  *Be Aware of the Natural Tension Points Across Functional Groups* 233
  *Identify Your Best Debuggers* 234
  *Reserve Resources or Develop Generalists for Later Repairs* 235
  *Team Types: Strengths for the Endgame* 237
    *Myers-Briggs Type Indicator* 238
    *How Do You Determine and Use Types?* 238
Wrap-Up: Additions to Your Endgame Toolbox 239

Chapter Fourteen: Leadership Practices 241
  *The Burden of Leadership* 242
    *The Power of Leadership* 243
  *Use of Overtime* 244
  *Establish a War Room* 245
  *Find Room for Play* 247
  *Daily Status Meetings* 248
    *Daily Meeting—A Quick Example* 249
    *A Similar Meeting Framework Resides in Scrum* 250
    *Other Benefits of Daily Meetings* 250
  *Gathering General Status* 251
  *Handling Distributed Endgame Teams* 252
    *Colocate Resources Whenever and Wherever Possible* 254
  *Knowing When to Say When—Release* 255
    *Failing to Fill Your Top 10* 256
    *QA and Testing Say It's Ready* 256
CONTENTS

When You’ve Met Your Release Criteria 257
Knowing When to Say When—You’re in Trouble 257
Recognition 257
Actions—What to Do and What Not to Do 258
If You Are Going to Reset, Do It Only Once 258
Collaborative Planning 259
Stick with Your Project Dynamics 259
Notify the Business—and Execute! 259
The Testing Team—Your Secret Weapon 259
Wrap-Up: Additions to Your Endgame Toolbox 262

Chapter Fifteen: Endgame Retrospectives and Conclusions 263
Keeping an Endgame Log or Diary 264
Should You Conduct a Retrospective? 265
Key Points of a Retrospective 265
Time Investment, Timing, and Preparation 266
The Notion of Safety 266
A Sample Meeting Flow 267
Guidelines for an Endgame Retrospective 268
Using Your Endgame Data in the Retrospective 268
Release Criteria and Working Views 269
Endgame Release Framework 269
Defect Analysis and Trending 270
Driving the Retrospective from the Data 271
Endgames Provide Wonderful Insights into the Character of Your Team 272
Celebrating Success 273
Planning 273
Making It Personal 274
Generate Stories 274
Make It a Big Deal 275
Agile Endgames 276
Daily Meetings 277
Heavyweight Defect Entry 277
Release Criteria and Endgame Release Framework 278
Work Planning 278
Concluding Thoughts 279

Afterword: An Invitation to Endgame Collaboration 281
Appendix A: Pre-Endgame Preparation Checklist 283
Appendix B: Collaborative Estimation, Data Focus Checklist 286
Appendix C: Sticky Note Guidelines 288
Note-Generator Guidelines 288
CONTENTS

Note-Facilitator Guidelines 289

*Initial Ordering Guidelines* 289

*Sequence Ordering Guidelines* 290

Note Formats 290

Appendix D:

**Guidelines for Constructing Endgame Release Frameworks** 291

  - Historical Notes 291
  - Plan Details 292
  - Endgame Flow 293

References 295

Index 299
I’ve earned most of my management scars during project endgames. Early in my career, the endgame appeared to be simply a chaotic, ad hoc, reactive period during the final phases of project delivery. It was a time to test your courage, mettle, and resolve. It was a gut check. Do you have what it takes? Can you do whatever is necessary to release a product?

The endgame, it seemed, was a time when defects ran rampant and were unpredictable, amorphous things. You didn’t plan to fix them—you simply reacted to them. Depending on your functional point of view, the endgame had different meanings. If you were in testing, then it was the culmination of all your plans. You were energized, at least at first, and ready to find as many defects as possible. Of course, you had less time than was originally planned, and everyone was pushing to reduce the testing effort. Still, it could be a very exciting time, and it was certainly your time.

If you were in software development, it was a frightening time. Woe to every developer whose cube entrance was darkened by a tester. That usually meant only one thing—the testers had found yet another defect and you were about to get more work than you had time for or had planned for. Moreover, if it was a high priority defect, you could expect every leader on the team to stop by to
check if he or she could “help” you with the resolution. And feature creep didn’t happen just at Halloween—it occurred steadily and consistently throughout the endgame.

If you were in marketing, you quite frankly had no time for the endgame. You had customer and sales commitments hanging out there, so the product needed to ship—now! And it needed to work . . . and it needed to meet all requirements . . . and . . . Actually, check that—you didn’t really care about the endgame. Your thoughts were already focused on the next project.

Problems got fixed due to clear, and sometimes not so clear, criteria. Oh yes, the fatal crash led to an easy repair decision. As did the database performance issue or the GUI screen errors. So, some decisions seemed to make perfect sense . . . but others did not!

- Sometimes, the loudest argument resulted in a repair, other times not.
- Sometimes, we seemed to be able to figure out when we were done, other times not.
- Sometimes, we could fix all priority or severity one defects, other times not.
- Sometimes, we repaired or corrected the right levels of functionality, other times not.

There were two constants within the endgame. First was the inconsistency. Second was the incessant pressure to be done—finished, released, and on to the next thing—and the tremendous effort the team would need to expend to get there.

The project manager always seemed to be gazing at the project plan and defect trends like a fortune-teller gazing into a crystal ball, wishing for the project’s end. It was as if the project endgame was simply happening in an enclosed room and the crystal ball of trends was the only hope for predicting what might happen. Everyone dutifully kept their fingers crossed, looking for a positive downturn in defect trending (which could imply success, but only some of the time).

Project goals were never really clear. For example, in one project, my team aspired to deliver a defect-free product to our customer. Or, at least, that was the role our test team envisioned for itself. We found, after many testing iterations, that we could not
get the product into a state that we could accept. So we kept iterating and iterating.

One day, we were in a release content meeting, and the point came up that our customers were experiencing many of the problems we’d already fixed in our current version. You see, our customers had not received an update in a year and a half. This point of clarity, which shifted our view of the release drivers from perfection to providing value to our existing customers, was a critical step in this particular endgame. After this epiphany, we shipped a new version of the product within six weeks, and our customers were delighted with the increased value and stability.

As I gained in my understanding of the endgame, my skill in negotiating it also increased. I began to react less, plan better, and succeed more often. I also began to think about the core lessons I was learning, which naturally led to the genesis of this book. My overriding goal is to share tools and techniques with you that should improve your endgame engagements.

**MY MOTIVATION FOR THE BOOK**

Simply put, the reactive nature of the endgame is due to a lack of attention on the part of everyone involved. While conducting research for this book, I was surprised to find very little work on triage and endgame management practices. Typically, in a text on the software life cycle, methodologies, or project management, only a few pages would address the subject.

Endgame processes, methodologies, and project management techniques are typically left for the reader to extrapolate as an exercise. The problem is—how do you do that? It’s not very clear what’s different about the dynamics of the endgame versus other aspects of the software development life cycle, nor what works and what doesn’t. There is simply not enough practical guidance available that is focused on the dynamics of the endgame. That’s what’s hard and unique about it—almost everything!

The idea behind this book is to give you some practical advice, templates, checklists, tools, and examples to help you improve your abilities. Not everything will work in every situation. However, it is my experience that there are common practices that will have a tremendous impact on your project endgames. I also want
to get you *thinking* about endgame activity as early as possible in your project planning. That's another key to success.

Finally, I want you to have fun—yes, *fun*—in the endgame. One of my biggest frustrations regarding endgame activity is that we typically lose the thrill and sense of accomplishment associated with completing a project. Normally, we end up so physically and mentally exhausted that we can’t enjoy the success. Or, worse than that, our project fails completely.

**INTENDED AUDIENCE**

My formal training is as a software engineer, so I initially come at things from that perspective. Over time, I began to lead software development teams as a group leader and manager. Within the past ten years, I’ve started to lead endgame efforts as a senior manager, test manager, and project manager. Each of these additional roles allowed me to look at things from a different perspective. It also meant that I led quite a few endgames.

The primary audience for this book is composed of technical managers within a software project endgame. Whether you are a software development manager, test manager, or project manager, I believe you’ll benefit greatly from the techniques and approaches I present within the book.

However, I think the audience is much broader than that, including virtually anyone who is involved as part of a software project endgame:

- product managers, marketing, sales reps, and customers
- project managers
- test managers, testers, and QA resources
- software development managers and software developers
- individual engineers engaged in hardware and/or software development and/or testing
- technical writing teams
- manufacturing, customer support, and other team members

Regardless of your life cycle or methodology, sooner or later, *everyone* arrives in the endgame—that is, if the project survives that far.
Always keep in mind that endgames are won and lost as a team, so whatever your function, role, or responsibility, you can and should support the achievement of the best possible results. Bring some of these ideas to your endgames and try them out. Everyone plays a part in endgame success!

**HOW TO APPROACH THIS BOOK**

Naturally enough, I think there is a sensible and important flow to the text, and I recommend that you read it in the order presented, for maximum benefit. My intention has been to make the text light enough to be read quickly, so that a sequential reading wouldn’t be too onerous a task.

However, if you prefer to focus on subtopics, then I recommend you scan the individual parts for points of interest in each of the focused areas:

- **Part 1: Endgame Basics**
- **Part 2: Endgame Defects**
- **Part 3: Endgame Workflow**
- **Part 4: Endgame Management**

Each chapter relates topic areas to one of the four primary focus points of the text. If you take a piecemeal approach to reading the text, I would recommend you first read the Introduction and Chapter 15, the retrospective chapter, to set some global context before moving to individual sections and chapters.

If you take away just a few key concepts or themes from the text, they should include the following:

- Many project-level activities apply equally well in the endgame and *should* be applied there as well. A good example of this is using collaborative estimation techniques on defect repair work.
- A project-level compass, the Working View, will help your team focus on a clearly articulated vision and will improve your decision-making. It is equally important to update this view as aspects of the project change in the endgame.
- A data-heavy view of defect entry and management will not only help you on your current project but will provide
valuable historical data for future projects. You will meet resistance here, but work through it and insist on solid data entry and maintenance.

- It's all about your team! Set it up with a framework for the endgame and then "Get out of the way!"

WHAT WE’RE NOT TRYING TO EXPLORE

I think it's just as important to discuss what’s not covered in the text.

First of all, I'm not exhaustive in discussing my scheduling or estimating techniques. Frankly, I'm not sure they can be applied to larger scale project planning. The estimating techniques certainly have the potential to be applied broadly, while the scheduling techniques probably have lesser applicability.

Second, I don’t discuss testing or the test process. These areas are much too broad, and many authors have explored test process dynamics. Our key interfaces to testing are at the following points:

- prerelease strategy and planning
- release framework planning
- defect data entry and status
- build and release hand-off
- triage and team meeting

Next, I offer only a very lightweight view of personality and team types in software management, to give you a feel for the topic and its implications. There is much more on this to pursue elsewhere.

Finally, my sole view of the endgame is from within it, from the initial release of a software project for testing, through to its release to the customer. I intentionally stay away from endgame process analysis, root causal analysis, retrospectives facilitation, and other detailed means of improving the processes within the endgame. That is, until the final chapter. It's there that I try to tease out some correlations between the endgame and broader software project management and methodology lessons I’ve learned.

November 2004
Cary, North Carolina

R.G.
SOFTWARE
ENDGAMES
All too often, product development teams struggle with effective priority decision-making. Usually, things get more difficult as the project progresses and business pressures to release the product build. Typically, projects define release criteria to guide the decision-making process. They define success for the project in the form of key objectives and requirements that the product must meet prior to being released. In many cases, the criteria focus on functionality, performance, and quality targets for the product.

I’ve developed a decision method or tool to help you visualize key project drivers or priorities and balance them alongside one another—I call it a “Working View,” as I mentioned earlier in the book. In doing so, teams gain insight into the product’s scope, development time, cost, and quality. How successfully you balance these dimensions determines how successfully you complete and deploy a product.

In this chapter, I examine the more traditional release criteria, exploring what they are and how to define them properly within the context of your endgame projects. Then we contrast release criteria against my notion of a Working View. Whereas release criteria are usually objective and requirement-focused, the Working View expands on them in several important ways:
it acknowledges that project release criteria changes are inevitable, particularly in the endgame, and it effectively handles this dynamic
• it considers every project dimension (scope, cost, time/schedule, quality, and team) and contrasts decision impacts across them
• it provides more in the way of a succinct vision for the project: where it is going and what it’s trying to accomplish
• it engages the team in the inevitable trade-off decisions and initiates changes across the team

While I’m a strong proponent of the Working View, at the end of the day, I’m not sure that I feel strongly about which approach or technique to use to manage release criteria. What is important is that you have release criteria of some sort, and that you define and agree on them as a team and adjust them as the project dynamics change.

RELEASE CRITERIA OR DEFINING SUCCESS

In [Rothman, 2002], Johanna Rothman writes about defining project release criteria. She discusses a five-step process for their definition. I will walk you through the steps and provide some brief examples. My preferred method for defining and managing release criteria is the Working View, which I will discuss in detail in later sections. However, the Rothman approach serves as a nice contrast and emphasizes quite similar activities.

Step 1: Define Success

What problem is the project trying to solve? What are the project’s goals? What is the business case? What are the key customer requirements? Craft a clear picture of what success looks like for the project effort. It should be tangible; you should almost be able to reach out and taste it.

Step 2: Learn What’s Important for This Project

Find out what the critical drivers are for this project. What is truly important? I worked designing and building medical systems for a
number of years, and it was always very clear that quality and safety were my highest priorities. This level of importance pervaded everything that we did as a team and how we approached our products.

Step 3: Draft Release Criteria

Take the time to draft a set of release criteria for review and discussion, and drive the team to clarity and agreement. Here is a sample:

- The code must support both Windows 2000 and Windows XP.
- All defects of priority P0, P1, and P6 will be repaired or addressed.
- For all documented bugs, the online help, release notes, and formal documentation will contain relevant updates.
- All QA tests will be run at 100 percent of expected coverage.
- No new defects P0 to P3 will be found within the last three weeks of testing.
- Release target: General Availability release on April 1, 2003.

Step 4: Make the Release Criteria SMART

Within HR circles, there is a notion of SMART objectives for defining personnel objectives. The acronym represents five key attributes for crafting good objectives and is just as applicable when defining release criteria. When you are validating your release criteria, it’s a good idea to make them SMART, too:

- Specific
- Measurable
- Attainable
- Relevant (or Realistic)
- Trackable

Here is a quick example:

The performance shall meet customer expectations
Having such a performance-related release criterion as your release criterion is far too ambiguous and not very useful. It leaves too many questions unanswered: Which customer? What exactly are its expectations? What specific areas of performance? A much better, or SMARTer, release criteria would be

**GUI screen update responses will never exceed 5 seconds.**

**Step 5: Achieve Consensus on the Release Criteria**

Once you've defined your release criteria, you need to gain stakeholder and team agreement that the criteria indeed capture the focus for the release. It's also a good idea to propose some project scenarios, check if the release criteria still hold, and provide guidance for them. For example, what will you do if you do not meet the above performance criteria? Will you stop the release and repair it? What if only one screen out of a hundred is affected—will you take the same action? What if the repair requires a total rewrite of the system architecture and approximately six months of effort—will you take the same action?

As you can see, there are quite a few "it depends" conditions in handling release criteria. Any preparation work you can do to establish what I call "decision boundaries" will help you later—when you're making these decisions on-the-fly, in the endgame.

**How to Use Release Criteria**

Again in [Rothman, 2002], Rothman talks about release criteria being binary in nature—it is either met or not met. Release criteria changes are limited to learning more about what it means to be "done" and realizing that you can't meet all the release criteria.

While release criteria are binary in nature, it is my experience that they may be quite volatile, as well. Perhaps I've worked in more dynamic or change-friendly projects, but I believe that release criteria have to be very dynamic in most projects. Expect to change them often in the endgame—almost every day, you'll gather change information that can potentially impact your release criteria and decision-making.

Finally, Rothman makes a wonderful point regarding the ultimate use of release criteria, describing them as a continuous metric
for determining whether the project is on track or not. It’s better not to wait until you reach a release decision point to determine whether you have met your release criteria. You should constantly monitor your progress against the release criteria you’ve set. The minute you think there is a problem, raise the flag and evaluate where the project stands.

**Better Decisions: Aspects of a Project Working View**

**The Problem**

The problem we’re trying to solve is that of effective decision-making. This becomes particularly crucial in the triage or endgame processes. Too often, the following problems arise:

- general team decision-making is difficult and usually ad hoc in nature
- it can take too long
- it may not include the right group or team in the decision
- decisions can be influenced by the wrong factors—for example, the strongest, loudest, or most extreme personalities and voices
- political factors
- often, decisions go undocumented and don’t stick for very long
- usually, decisions are biased toward one functional group, with very little balance or compromise at the individual decision level

All of this is exacerbated in the endgame because of the number of decisions that need to be made and the intense pressure on the project.

**Definitions**

The Working View is intended to capture the priority essence of your project. It is part release criteria, part project vision, and part key requirements. For general purposes, consider it a virtual replacement for release criteria. By defining it clearly and deeply, you provide definitive direction to team members on what’s truly important within their functional and individual efforts. Even
more importantly, it is dynamic—as your project dynamics change and discoveries are made, you continually adjust the view to capture new information.

At the highest level, it is composed of the following dimensions:

- scope
- cost
- time schedule
- quality
- team

The first three make up the standard project management triangle, with quality being related to all of the primary three dimensions. I added the team dimension because I believe it's equally important.

You evaluate a particular project decision trade-off based on each of these dimensions at the highest level. This forces you to consider the interactions between project drivers and to balance the decisions more effectively.

It is quite useful to capture your Working View graphically, as you drill down and define dimensional attributes. Kiviat, spider, or radar charts are useful for this purpose. Use them with 10 degrees of ranking per axis, with 10 referring to the most important dimensional factors and 0 to the least important. Use 4 to 6 axes per chart, preferably 6.

Example Project Working View

![Example Project Working View](image)

*Figure 3.1: Example Project Working View.*
In Figure 3.1, you can clearly determine that Time to Market and maintaining Cost are the driving forces for the effort, and that Scope is compromised in order to achieve that goal.

**Dimension Expansion**

First, let’s expand each of the dimensions to explore some primary attributes.

**Scope**

- overall release contents
- features, requirements, and key constraints
- performance characteristics
- key team members or teams (Team)
- required technologies, third-party integration (Cost)

**Cost**

- human resources (Team)
- tool resources
- third-party resources (Team, Cost, and Quality)
- recruiting/attrition costs (Team)
- TTM acceleration (Time)

**Time/Schedule**

- potential lost opportunities
- slippage: What do we do if we're slipping? Drop which core functions?
- early: What do we do if we’re ahead? Add which core functions?
- testing: Effects on testing? Increasing or reducing time? Changing release criteria?
- resources: Does adding resources help? Where is the best place to add them?

**Quality**

- impact on customers (Cost, Team)
• impact on company or product image (Cost)
• cost of rework: technical support, repair, and distribution (Cost)
• trade-offs associated with functionality, requirements traceability, performance, and ad hoc testing
• impact on team morale (Team)

**Team**

• effects on overtime, vacations
• awards, compensation, and recognition (Cost)
• handling possible attrition (Cost)
• consulting and contracting assistance (Cost)

**Fixed Versus Variable Dimensions**

In every project, there are both variable and fixed dimensions—simply due to the constraints of the project. In my experience, time and cost are the most commonly fixed dimensions. When defining a Working View, it’s important to define the fixed dimensions and then drill down and add further detail to the variable dimensions. The fundamental idea is to map variable dimensions into concrete, detailed, and meaningful attributes that can guide the team’s priority decisions.

**Steps to Establishing a Working View**

**Step 1: Identify Your Project Stakeholders**

This list should include all cross-functional project participants. It should also include leaders from your core departments (for example, software development, quality, and marketing, on a typical software project). Finally, include senior leadership and project sponsors, as appropriate. The key is to get all of the pertinent stakeholders and decision-makers together to ensure you achieve a quick agreement.

**Step 2: Set the Stage**

If this is your initial effort to define the Working View, then this step is the establishment of the Working View. If you are in a redefi-
nition phase, then in this step, you should highlight what is changing and more importantly why it is changing. In both cases, this is the step where you set the tone for the effort. If the project is struggling to get started, then say so. If you are way off schedule, then say so. The clearer the team is on the current state of the project, the easier it will be for them to define a Working View.

Step 3: Project Vision, Essence, and Release Criteria

Establish the primary reasons for the project’s existence. These are the high-level drivers that will dictate the priorities of your effort. Keep in mind that this isn’t the time for priority negotiation—that is best left for later, after a more detailed analysis. Take more of a “capture and move on” strategy at this point, discussing and considering the following:

1. Are there any critical historical and business agreements or commitments? (Be sure to include internal commitments here. For example, I worked on a project where QA staff members were promised a clean-up effort on the next release if they loosened the quality requirements on the current release.)

2. What is the essence of the business, the product, and the customer? (These are areas where there can be no compromise, for example, print quality or ease of setup and installation in laser printers.)

3. Identify release targets—schedule, alpha and beta commitments, customer expectations—and time to market: key market windows, trade shows, and annual events. (The more concrete information you gather as to why time is critical, the easier it will be for your team to understand that it’s not arbitrary!)

4. What are the specific stakeholder drivers? (If there are any, ensure that they are very specific and accurately mapped to a specific dimension.)

5. Are there other dimensional drivers, such as cost, quality, scope, or team? (You should at least have some team drivers—for example, “We will not resort to eighty-hour workweeks for the next six months in order to meet our Scope requirements.”)
6. What are the key (or golden or brass ring) features of the product? What features can customers not do without? (Word processors have literally hundreds of features; however, there are probably only a critical 10 percent of these that cannot be compromised.)

Once you’ve answered these questions, try and get a sense of priority from the stakeholders. Encourage them to think in terms of ranking and interrelationships. The quality of your issue segmentation, ranking, and prioritization will directly relate to the quality of your Working View.

**Step 4: Explore Product and Project Dimensions**

This is basically a brainstorming session during which the stakeholder team is assembled to identify the problems, map them to the affected dimensions, and brainstorm appropriate changes for moving the effort toward feasibility (see Chapter 12 for some advice on specific techniques). There are several heuristics that will help your exploration:

1. Using some sort of graphical representation for attribute comparison and ranking is extremely helpful. As I mentioned earlier, I find Kiviat or spider charts particularly useful. Use one of these spider charts per dimension and map it with four to six attributes. Use multiple charts for more attributes. Just make sure you reconcile priority across the charts.

2. It is also helpful to create a “worded view” as documentation. It should fully support the graphical representation and accurately characterize the intended prioritization.

3. If ranking attributes, agree on simple scaling rules. For example, score the attributes on a scale of 1 to 10, with 10 being the highest priority; try to score the attributes so that there is one that is clearly the highest priority; strive for two degrees of separation between attributes.

4. Don’t be afraid to clarify attributes at increasing levels of detail. They need to be detailed enough to be useful in ranking, comparison, decision-making, and guiding your team. If you find the team struggling with a particular
dimension attribute or point, it usually implies that you need to reduce it to finer granularity and detail.

5. The exercise is very similar to requirements writing in that your attributes need to be complete, correct, feasible, necessary, prioritized, unambiguous, and verifiable. Karl Wiegers’s book *Software Requirements* [Wiegers, 1999] is an excellent place to explore effective techniques to insure correct attribute definition.

6. Reaching an agreement is sometimes difficult. However, such an agreement is the core of the process that the team needs in order to accomplish its goals. There are many decision models available for different teams, cultures, and situations. I prefer a team-consensus-based approach for most Working View definition sessions. Taking into account your environment, decide on an effective decision-making approach and stick with it.

7. Document your Working View both in written and graphic form (using charts and figures, for example). Then distribute it among your cross-functional team. I always prefer posting the current Working View materials in the project war room. Not only does this identify the Working View as an important project artifact, it helps initiate the necessary changes within the team.

8. Finally, insure that action assignment and tracking are being performed. Then link them to your change management and change control mechanisms.

**Examples of Working View Application Scenarios**

Now we are going to go through a real-life case study of a project, which will show you how to apply Working Views. First, the project background information:

- The company is a European-based provider of network analysis and test equipment.
- It takes pride in utilizing the best *engineering* possible in the production of its products, which customers view as among the best available.
- A set of the company’s component products has been out in the field for three-to-five years.
• The products have evolved separately, each having received between two and four major, and many minor, releases.
• The market is moving toward suite-based products, and the company hasn’t made a major release to any of the components in more than a year.

The company has embarked on an initiative to create a network analysis suite from its disparate products, and there is tremendous pressure to get it to customers soon. The project is currently within the endgame and is struggling to achieve a successful release.

The scenario examines a problem with release criteria. There are conflicting goals within the team. To be specific, the time and quality dimensions are at odds within the project endgame. There is tremendous pressure to release the software, in conjunction with similar pressure to release with minimal to zero defects. These conflicting goals generated opposing forces within the team and little progress is being made. In the first part of the scenario, the team conducts a Working View development exercise to flesh out the conflict and to come to an agreement with the project’s sponsors on the right balance across the conflicting dimensions.

In the second part of the scenario, the team conducts another Working View development exercise, this time to fine-tune the impact on the quality dimension of the higher-level view and to add granularity to the view along this dimension—so the team better understands the testing focus.

This workflow is indicative of the normal processes associated with Working View development and highlights a difference between the Working View and release criteria. Usually, the Working View is not developed in a single, succinct event. You normally redefine the Working View at the highest or project level and then negotiate the dimensional impacts with increasingly detailed and refined Working View exercises on each of the affected or changed dimensions. You iterate into more detail on each dimension until the team is clear on the change and the necessary adjustments and supports these changes.
Software Endgames

Project Application: Example #1

The Problem

The problem analysis point came during endgame testing for the initial product release. Testing staff discovered many interoperability issues not covered in the requirements, while significant ad hoc testing was exacerbating this trend.

The test teams were pushing for close to zero defects at release, which was partly due to the culture and partly due to previous commitments—the testing team was aiming to improve on the product’s quality in the next release.

Marketing and executive leadership were creating tremendous release pressure. Also, to make things worse, there was a significant lack of experienced development resources. CCB meetings were becoming very contentious—we couldn’t fix everything, we couldn’t seem to make balanced decisions, and we were spinning out of control.

As it turned out, our executive leadership’s priorities were also out of synch. Our marketing and engineering VPs were pushing for immediate release while the quality VP was emphasizing zero defects to his team. The project team was caught in the middle of these opposing forces.

The Solution

The company needed better clarity on balance across time and quality project dimensions—it needed to rank-order the key drivers!

Participants

Include the product manager, project manager, VPs from engineering, marketing, testing, and QA.

Dimensions of the Problem

- Acquiring experienced resources is a challenge.
- Time to market (TTM) is our ultimate priority. Our customers and the business need the release.
• Some **overtime** will be required (give extra effort).
• Existing **functionality** must operate as previously designed—even in the interoperable cases—and we must verify all field-based severity 1 and 2 repairs.

**Ranking**

TTM = 10 (fixed), Cost = 10 (fixed), Scope = 7, Quality = 7, Team = 5. Figure 3.2 displays these rankings in a spider diagram.

**High-Level Project Working View**

![Spider diagram showing rankings for Time to Market, Team, Cost, Quality, and Scope.]

*Figure 3.2: High-Level Project Working View.*

**Worded Working View**

We must deliver this release on April 1, 2002, using our existing resources. Overtime may be necessary to meet this release date. We must deliver critical content and can't regress in functionality. However, when pushed, we will compromise quality first (testing time and focus), and then features.

**Agreement**

We agreed that this was our high-level priority compass for the remainder of the effort. We can easily generate release criteria from this Working View:
As I said in the problem definition, our primary problem was inconsistency in quality and schedule priorities, particularly at the executive level. This exercise helped to align both dimensions and gain balance across the two perspectives—that of the VP of quality and the VPs of marketing and software development.

We then had to take this high-level view and develop more detail along the quality dimension to insure that we were operating properly within the testing team.

**PROJECT APPLICATION: EXAMPLE #2**

**The Problem**

While the above exercise was helpful in achieving consensus on priority drivers within our leadership structure, we still had some work to do within the team. We needed to socialize the above Working View into our testing team—and to sort our test focus for the remainder of the endgame. How were we to support the statement that "We can't regress deployed functionality—even in the interoperable cases—and we must include all field-based priority 1 and 2 repairs"?

**The Solution**

We need to drill down into the key quality dimensions for the project and rank-order them.

**Participants**

Include the product manager, project manager, and team leads from development and testing functions.

**Dimensions of the Problem**

- Insure we have full regression tests for deployed functionality—continue to run tests and report results.
Three • Developing Release Criteria and Working Views

- Extend existing regression tests to insure interoperability is covered.
- There are new additions to the regression suite.
- Repair verifications may lag behind.
- We can’t perform any ad hoc testing.

Ranking

Previous Version Regressions = 10, Interoperability Regressions = 10, Priority 1 and 2 Repair Verification = 8, General Verifications = 6, and Ad Hoc Testing = 2. Again, we display the quality dimension expansion in Figure 3.3.

Worded Working View

Our highest priority in testing is to insure that we deliver working repairs for reported field defects at severity levels 1 and 2 without regressing already-deployed functionality. We must also extend regression testing to account for interoperability among the point products. We may lose sight of some low-priority and low-risk verifications when trying to catch them in regression. We will have no time for ad hoc testing.
Agreement

We agreed that this was our high-level-priority Working View for the remainder of the effort. The following needs were captured as part of the exercise:

1. We need to define the component interoperability requirements (marketing).
2. We need to understand the current level of coverage for regression testing (test and development).

The two steps in the example—the high-level alignment with the executives and the lower-level definition of testing focus—helped us immensely in our CCB meeting and processes. Together, they meant that we all essentially viewed the release criteria and project priorities in the same way.

Example Results

We had been spinning for about three months in this state, unable to agree on priority and focus for release drivers and conducting a never-ending endgame. All of the executives were in a state of panic and looking for problems and solutions in black-and-white terms, hoping to find a scapegoat. What was interesting is that they were responsible for the vast amount of project churn and didn’t even realize it.

There was a powerful side effect of getting the executives to agree on a balanced view of priority. It wasn’t easy, but it was necessary. It was also surprising at the time. You would expect a handful of senior leaders in a company to be able to synchronize their decisions relatively easily. However, the reality proved to be quite the opposite. Therefore, the Working View exercise serves not only to align the team, but also to synchronize the view horizontally across the various functional organizations.

Once we aligned ourselves and our Working View, the CCB meetings and our decision-making began to go much more smoothly. We turned the project around and delivered to beta testers in six weeks. As part of our post mortem analysis, we recognized this realignment of the release criteria as one of the defining moments in getting back on track.
Another technique for documenting a Working View dimension is to list high-priority and low-priority focus points. The idea is to produce enough contrast for the team to understand where its priorities and focus should lie. Again, you need to establish enough detail to create clarity in decision-making.

Using the Working View from the introductory example, Figure 3.4 contrasts the following attributes for quality.

<table>
<thead>
<tr>
<th>High-Priority Attributes</th>
<th>Low-Priority Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Existing functionality cannot be affected by new changes (functional regression testing).</td>
<td>• Interfaces beyond 10/100/1000 Ethernet and ATM are lower priority.</td>
</tr>
<tr>
<td>• Existing performance may not be degraded by new changes (performance regression testing—we also lacked a performance benchmark).</td>
<td>• Existing performance may not be degraded by new changes—even when running multiple components, don’t get hung up on improvement.</td>
</tr>
<tr>
<td>• New functionality must work.</td>
<td>• New functionality must work, except new reports that do not map to older reports.</td>
</tr>
<tr>
<td>• Component interoperability without performance regression.</td>
<td>• Component interoperability across all permutations—we can identify (n) key configurations for initial release.</td>
</tr>
<tr>
<td>• Installation framework must create correct initial environment.</td>
<td>• Installation framework needn’t accommodate all previous installation environments.</td>
</tr>
<tr>
<td></td>
<td>• Ad hoc testing, early is better; later—not at all.</td>
</tr>
</tbody>
</table>

Figure 3.4: High- vs. Low-Priority Contrast Working View.

Resetting Your Working View

Setting your Working View is not a static exercise for defining project release criteria. It will probably change frequently throughout the project, particularly in the endgame. To give you a flavor for reset events, here are a few sample drivers for a Working View reset:

- departure of team resources (attrition, vacations, illness)
- schedule slips due to internal dynamics (underestimation) or external dynamics (management-driven schedules)
SOFTWARE ENDCOMES

• feasibility discoveries as part of prototyping (architecture, design, and performance)
• defect find/fix ratios as part of endgame testing
• regression testing progress
• additional features added to the product, with or without any schedule “relief”
• choosing to reduce functionality in order to meet time requirements

You initiate the same process to reset the Working View, simply highlighting differences or changes that have occurred and coming to a new agreement. As a general rule, you should not add or extend without deleting or contracting attributes within your Working View.

It's also a good idea to map all changes to the root cause or problem, just so that it’s clear what drove you to the reset and why. Finally, you should calculate the impact the change makes to insure you’re getting desired results. For example, does the reduction in quality or scope targets actually meet the required release time frame?

It’s important to note that whenever a reset occurs, there should be a mechanism to notify the team of the reset. Acceptable mechanisms for this include

• informal socialization
• team e-mails
• team meetings
• posting the new view in your project meeting or war room

WRAP-UP: ADDITIONS TO YOUR ENDCOMES TOOLBOX

This approach and model can help in other areas as well—leading the team, providing project mission and vision, and generally documenting the important bits that should be driving your efforts.

The approach can also be adapted to support other activities in the project life cycle, such as

• defining system architecture—where dimensions represent architectural attributes
• contrasting different design approaches
deciding what level of inspections need to occur—where, when, and to what degree
• forming early testing strategies: where to focus, risk areas
• risk analysis
• almost anything that requires clarity of detailed requirements in order to make an informed, collaborative decision

It is extremely important to distribute the Working View among your team members. The views truly become graphical rallying points to insure that the team maintains focus. They also emphasize that you've taken a step beyond simply stating requirements and demands, to truly considering cross-dimensional implications and balancing your priorities accordingly and effectively.

Here are the key points:

• Have a more formal way of capturing problem dimensions and balancing priority.
• Define and rank decision criteria (attributes, dimensions) as a team.
• Make the decisions visual.
• Document the decisions and have a change process.
Acceptance testing, 8, 101
Agile methodologies, 8, 78, 96, 276
Allison, Anna, 156, 295
Alpha testing, 62, 68, 96, 120, 294
Anti-pattern, 85–96, 100

Bach, James, 137, 296
Bays, Michael E., 100, 295
Beck, Kent, 202, 204, 228, 236, 244, 277, 295
Beedle, Mike, 250, 297
Beta testing, 58, 62, 68, 73, 77, 96, 120, 127, 129, 144, 151, 218, 294
Black, Rex, 295
Blockers, 158, 200
Brown, William J., 85, 100, 295
Builds, 12, 26, 66, 67, 76, 98, 101, 104–7, 230, 261

Cards on the Wall, 217–18, 219, 220
Change control, 6–7, 8, 9–10, 13, 16, 19, 22, 24, 26, 52, 90, 91, 100, 101, 269
Change Control Board (CCB), 9–10, 11, 13, 19ff., 30, 37, 38, 39, 40, 54, 58, 72, 96–97, 101, 124, 194, 196, 210, 226, 249, 254, 261, 264, 278, 292
agenda, 32–36, 41
facilitator, 28, 29, 31, 292
participant roles, 26–28, 32
process, 99, 254, 261, 292
Change reduction, 12, 13, 64, 70–71, 78–80, 81, 83, 85, 86, 96–97, 191, 293
anti-patterns, 79, 85, 97
Code complete, 71, 81, 82–83, 84
Code freeze, 34, 71, 72, 79, 80–83, 187, 271, 291, 293
Consensus, 38ff., 45, 52, 56, 124, 194, 219, 258
Construction, 12, 65, 68, 205, 236, 260
Costs, 43, 47ff., 165, 169, 179, 190
Customers, xxi, 3, 8, 26, 39, 43, 44, 48, 50, 51, 52, 64, 65, 78, 93, 95ff., 113–14, 120, 124, 156, 161, 163, 164, 167, 169, 172, 194, 201,
Ownership, 75–76
Plan, 76, 210, 245, 291, 292–93
Endgame team, 236, 246, 249, 260, 261, 262, 264, 267, 268, 273, 278
Character of, 272–73
Distributed, 252–55
Engineers, 94, 124, 127, 132, 147, 157, 159, 179, 180, 181, 184, 185, 190, 196, 211, 227, 235, 243, 245
Categories of, 237
Enhancements, 114, 126, 134, 144, 145, 155, 162, 167
Entry criteria, 73ff., 103, 200, 253, 293, 294
Estimates, xxiv, 67, 89, 191, 211, 212, 215, 220, 221
Estimation, 15, 133, 208–9, 209–11, 214–17, 217–20, 222, 280
Collaborative, 208, 209, 220, 286–87
Planning, 214, 216, 235
Sticky note, 209, 217, 222
Exit criteria, 73, 74, 253, 293, 294
Expectations, 26, 27, 32, 44, 50, 68, 69, 91, 150, 191, 229, 230, 233, 239, 253, 272
Extreme Programming, 8, 63, 78, 91, 97, 202, 204, 205, 228, 236, 244, 250, 277, 278
Falk, Jack, 118, 125, 296
Feature creep, xx, 72, 78, 86, 91
Features, 4, 78, 86, 134, 206
Adding, 60, 91, 158
Functionality, xx, 49, 55, 56, 62ff., 70, 73, 74, 80ff., 86, 90, 100, 106, 118, 149, 151, 158, 173ff., 181, 185, 201, 202ff., 218, 235, 261, 269, 278–79, 292, 293
Partial, 66
Reducing, 60, 94
Removing, 161, 173–74
Stabilizing, 270
Verifying, 65
Functional testing, 75, 120, 279
General Availability, 44, 68, 69, 128, 144, 145, 273
Granularity, 64, 76–77, 90, 98, 100–101, 105, 144, 146, 149, 264, 290
Hohmann, Luke, 238, 296
Howard, Alan, 237, 238, 296
Humphrey, Watts S., 20, 21, 211, 214, 296
Integration, 66, 73, 145, 153, 199, 216, 217, 235
Testing, 67, 73, 102, 103, 158
Kaner, Cem, 118, 125, 137, 296
Kaner, Sam, 36, 38n., 296
Kerth, Norman L., 265, 266, 267, 296
Keywords, 116, 127–29, 138, 144–46, 151
Kickoff meeting, 214, 216
Kiviat charts, 47, 51
Lister, Timothy, 244, 296
Logisticians, 26, 27
Management, 15, 151, 225, 238, 242, 252
McCarthy, Jim, 7, 297
McCormick, Hays W., 85, 100, 295
Methodologies, xxi, xxii, 3, 8, 19, 62, 63, 76–77, 79, 86, 92, 96, 138, 153–55, 244, 280
Metrics, 13, 14, 33, 35, 45, 138, 151, 156, 159, 257, 292
Metrics Analysis Questions, 139, 156–58, 159

301
INDEX

Morale, 49, 192, 193, 247, 258
Myers-Briggs Type Indicator, 238

Necaise, Cindy, 261, 297
Nguyen, Hung Quoc, 118, 125, 296

Overtime, 55, 78, 231, 241, 242, 244–45, 262, 272

Packaging, 14, 22, 27, 64, 69, 167, 179, 180, 186–90, 276, 278
 costs, 190–91
 themes, 187–90

Pareto Analysis, 151, 152, 261, 271

Personality types, 29, 46, 226, 237–39, 240

Personal Software Process, 209, 211–14, 222

Petersen, Erik, 151, 297
Pettichord, Bret, 137, 296
Phillips, Dwayne, 100, 213, 217, 297
Planning, 64, 104, 181, 188, 201, 202, 257, 258, 259, 280

Prioritization, 24, 26, 51, 58, 202, 203, 204, 206, 207
Priority, 93, 122f., 138, 144, 150, 151, 161, 162, 163, 166, 182, 194, 210
 changing, 123, 124, 160, 161, 163–64, 201
 levels, 124–26

 external, 174–75
 freeze, 80
 grading, 261
 instability, 165–66
 key features, 51
 life cycle, 10
 maturation, 70, 75, 155
 stability, 3, 4, 87, 95, 97, 104, 105, 114, 140, 142, 152, 158, 292
 stabilization rate, 270
 type, 95
 Project dimensions, 47–49, 51–52, 54
 conflicting, 53
 drivers, 42, 43, 50
 dynamics, 47, 68, 76, 136–37,
 259, 270
 fixed, 49
 life cycle, 3, 60
 planning, xxiv, 231, 254
 ranking, 51, 54, 55, 56
 Project management, xxi, 47, 85, 100, 114, 124, 155, 159, 181, 182, 218, 228, 249, 250, 277, 280
 Project manager, xx, xxi, 4, 5, 10, 14, 23, 26, 31, 54, 56, 68, 75, 80, 97, 102, 103, 124, 129, 135, 181, 198, 217ff., 226, 230, 233, 247, 252ff., 260, 261
 Proxy-Based Estimation, 209, 211–14, 222

Quality, 43, 47, 48, 50, 53, 54, 55, 56, 57, 59, 60, 67, 70, 71, 73, 74, 105, 107, 114, 124, 163, 173, 181, 186, 234, 258, 260, 261, 270
 Quality assurance (QA), 54, 75, 194, 256
 testing and, 105

Regression, 34, 66, 78, 88, 90, 97, 120, 148, 155, 166, 188, 191, 198, 205, 213, 293
 testing, 56, 57, 60, 62, 67, 73, 74, 77, 120, 153, 190, 279

 closure, 79
deferring, 95
 knowing when to, 255–57
 plans, 11, 24, 211
 point, 63, 103, 108, 141, 256, 259
 schedules, 62, 70
 targets, 50, 77, 80, 90, 101, 166
to testing, 12, 62, 64, 66, 74, 77, 81, 86, 245
 Release criteria, 10, 11, 13, 24, 26, 36, 42ff., 48, 50, 53, 55, 58, 59, 67, 70, 79, 85, 86, 92, 97, 136, 164, 190, 244, 245, 253ff., 261, 263, 268, 269, 276, 278
 adjustments to, 34–35, 43, 76, 227
cycle, 139, 141, 153, 154, 270, 293
Release hand-off, xxiv, 102, 103, 108, 190
Repair, xxiii, 10, 32, 39, 55, 70, 73, 79, 84, 89, 90, 93, 97, 102, 103, 104, 114, 117, 126, 127, 128, 131, 134, 137, 142, 147, 154, 158, 161, 165, 171, 172, 173, 175, 180, 186, 188, 194, 199-201, 204, 205, 206, 217, 235, 238, 255, 258, 279, 293
alternatives, 160, 175
deferring, 160, 165-67
designing, 31-32
effort, 24, 124, 210
estimates, 12, 14, 117, 197, 198, 222
planning, 19, 186, 198, 216
scheduling, 6, 11-12, 22, 132, 195, 199, 200, 203, 207
verification, 12, 26, 77, 80, 129, 130, 148, 149, 153, 200, 271
Requirements, 61, 63, 68, 87, 91, 92, 95, 97, 111, 133, 161, 164, 167, 170ff., 188, 233, 235, 236, 257, 269, 293
change, 22, 91, 92, 167-68, 272
elicitation, 95
specification, 75
traceability, 49
Retrospectives, xxiv, 15, 58, 250, 264, 265-68, 268-72, 292
Prime Directive, 266
Rework, 3, 66, 67, 72, 73, 78, 80, 86, 89, 179, 180, 191, 197
Risk, 61, 64ff., 73, 87, 90, 102, 107, 149, 162, 166, 168, 179, 180, 191, 221, 241, 247
management, 87, 132, 151, 230
reducing, 134
Rothman, Johanna, 43, 45, 297
Sabourin, Robert, 20, 71, 122, 124, 126, 163, 297
Schedule, 26, 34, 43, 47, 48, 50, 56, 63, 67, 75, 85, 91, 94, 96, 108, 111, 128, 143, 144, 146, 150, 155, 158, 166, 186, 192, 229, 245, 261
changes, 76, 92, 102, 103
slips, 59, 86, 192, 199, 257, 272
Scheduling, xxiv, 15, 32, 62, 72, 111, 195, 201, 202, 204, 211, 213
Schoor, Bruce, 83, 84, 297
Schwaber, Ken, 250, 297
Scrum, 250, 251, 277
Selby, Richard W., 81, 83, 126, 295
Settling time, 135, 188, 211, 269, 270
Severity, 122-23, 123-24, 126-27, 161, 162, 163, 166, 183, 194, 210
changing, 123, 124, 160, 161, 163-64
Showstopper, 126, 184, 187, 188, 200, 218, 245
Silver, Denise, 217, 298
SMART objectives, 44-45
Smoke testing, 12, 26, 66, 73, 74, 75, 98, 101, 104-7, 158
Software, 80, 111, 114, 123, 142, 143, 161ff., 171, 201, 206, 234, 255, 259
Sticky note brainstorming, 288-90
Subject-matter experts, 24, 215
Team, 43, 45, 47, 49, 50, 65, 74, 76, 77, 80, 87, 99, 101, 120, 137, 139, 140, 144, 157, 159, 168, 180, 185, 191, 195, 202, 208, 217, 221, 229, 234, 242, 249, 258
colocating, 242
dynamics, 12, 225, 239, 263
history, 63, 64
roles, 131
size, 189, 228
Team leaders, 22, 93, 98, 102-3, 131, 132, 208, 222, 225, 226-29, 242, 248, 249, 258, 271
finding, 227-28
Team members, 26, 27, 37, 38, 65, 68, 89, 92, 129, 181, 184, 198, 199, 201, 205, 215, 220, 222, 226ff., 232, 236, 239, 245, 248, 251, 256, 260, 272, 274, 278
colocating, 254-55
sign-up, 228-31, 278
Teamwork, 15, 209, 218, 233, 245, 251, 254, 255
INDEX

Technical leads, 91, 180, 182
Tension points, 225, 233–34, 239–40
Testing, xix, xx, xxiv, 3, 4, 8, 14, 44, 53ff., 62, 66, 68, 70ff., 81, 83, 84, 87, 93, 103, 111, 120, 121, 136, 140, 144, 148, 149, 151, 152, 154, 158, 159, 162, 166, 169, 172, 185–86, 190, 193, 196, 197, 199, 201, 203, 222, 226ff., 249, 253, 255, 269, 271, 293
ad hoc, 54, 57, 87, 88
cycle, 67, 70, 71, 72, 105, 143, 154, 269, 291, 292, 293
cycle time, 153, 270
efficiency, 180
external, 86
guidance, 117
iterative, 63
plan, 64, 86, 91, 94, 152, 171
schedule, 94, 197
strategies, 61, 64, 66, 71, 73, 87, 148
Test leads, 102, 103, 211
Test manager, xxii, 75, 135, 198
Test plan, 63, 88, 103
Thomas, Scott W., 85, 100, 295
Time, 53, 166, 208, 220
to market, 48, 50, 54
to test, 210, 211
to verify, 210, 211
Time to repair, 117, 132, 161, 183, 210, 211
actual, 117, 133, 270
Top 10 list, 89, 184, 230, 246, 252, 256
Trending, 13, 34, 75, 119, 139, 143, 146, 151–56, 157, 159, 244, 269, 270–71, 292
Triage, xxiv, 5, 7, 8–12, 13ff., 20ff., 27, 29, 32, 34, 46, 72, 73, 92, 119, 127, 129, 132, 136, 137, 144, 146, 147, 180, 194, 196, 210, 245, 261, 263
Unit testing, 21, 73, 74, 89, 90, 93, 102, 103, 105, 117, 119, 134, 143, 145, 158, 162, 197, 236
Verification, 57, 73, 74, 107, 153, 279
Versions, 115, 189, 193, 201, 215
Voting, 38–39
War room, 11, 15, 26, 35, 52, 60, 72, 76, 230, 242, 243, 246–47, 252, 255, 262, 278, 279
virtual, 254
Weigers, Karl, 52, 214, 298
Wideband Delphi, 209, 214–17, 218, 220
Wood, Jane, 217, 298
Workarounds, 26, 89, 126ff., 134, 144, 161, 163, 168, 171–73, 175, 194, 201, 216, 219, 271
Workflow, 8, 10, 11, 53, 64, 72, 76, 81, 104, 129, 148, 180, 181, 186, 188, 207, 268, 270, 293
adjusting, 93, 227
application, 52–53
Work queues, 12, 14, 94, 117, 131–33, 136, 162, 179ff., 189, 193, 245, 276
loading, 184–85, 278