Professional Java Tools for Extreme Programming
Ant, XDoclet, JUnit, Cactus, and Maven

Richard Hightower
Warner Onstine
Paul Visan
Damon Payne
Joseph D. Gradecki
Kate Rhodes
Robert Watkins
Erik Meade

Wiley Publishing, Inc.
Professional Java Tools for Extreme Programming
Ant, XDoclet, JUnit, Cactus, and Maven
Credits

Authors
Richard Hightower
Warner Onstine
Paul Visan
Damon Payne
Joseph D. Gradecki
Kate Rhodes
Robert Watkins
Erik Meade

Executive Editor
Robert Elliott

Book Producer
Ryan Publishing Group, Inc.

Copy Editors
Liz Welch
Linda Recktenwald

Compositor
Gina Rexrode

Vice President & Executive Group Publisher
Richard Swadley

Vice President & Executive Publisher
Robert Ipsen

Vice President and Publisher
Joseph B. Wikert

Executive Editorial Director
Mary Bednarek

Editorial Manager
Kathryn A. Malm
Richard Hightower

Richard Hightower is Chief Mentor/Consultant for ArcMind, a full-service software development company. Rick is a software engineer by training who specializes in software development tools and processes, and developing enterprise applications using J2EE, XML, UML, JDBC, SQL, and open source technologies. Formerly he was the Senior Software Engineer for Java Architecture at Intel’s Enterprise Architecture Lab. Rick is the co-author of Professional Jakarta Struts, and he contributed two chapters to the book Mastering Tomcat.

Rick has spoken at a variety of industry conferences and events, including JavaOne, TheServerSide.com Software Symposium JDJEdge, WebServicesEdge, and the Complete Programmer Network software symposiums.

Warner Onstine

Warner Onstine is a founder and CTO of Interface Guru, a leading Web Usability firm where he consults on back-end technology issues with specific emphasis on how technology and usability work together to present the user with an easy-to-use interface. Warner also runs his own custom development shop, Sandcast Software, which focuses on community and team-oriented software.

Warner got his first computer, a TI-99 4/A, when he was 9 and almost immediately attempted to program a game in Basic on it, which did not work. He stubbornly refuses to get rid of that machine though, along with his trusty NeXT Turbo MonoStation, upon which he got his first taste of Objective-C. This eventually led to Java, skipping right over C++.

His strong community background soon led him to open source, where he has had the chance to meet and interact with several incredible individuals from Jakarta and other open source communities. This also has helped him to keep an open eye on the trends that will soon shape the new landscape—one of his specialties.

Another skill he has is in assisting companies with making the right choices at the right time, utilizing XP, in-depth knowledge of their subject area, and the currently available tools germane to their problem. Warner is also a co-founder and President of the Tucson Java Users Group, which he helped form in 2001, which keeps him rather busy at times. Previously, Warner worked at eBlox, Inc. (a Web development company), Intalio, Inc. (a bay-area Business Process Management Server company), and the University of Arizona Main Library on the Digital Library Team.

Paul Visan

Paul Visan is an expert J2EE developer. He serves as a Principal Software Engineer for eBlox, Inc, where he finds that open source tools are invaluable to his work. Paul is a proud Romanian native, now living in the heart of Tucson Arizona. He is a regular contributor to IBM’s developerWorks, for which he has written a series of tutorials on Java Web Services. Paul would like to thank Andrew Barton, Nicholas Lesiecki, Tim Ryan, and Victoria McSherry for helping with this book.
Damon Payne

Damon Payne currently works as the Microsoft Business Group manager for Centare Group, Ltd. in Milwaukee, WI. Damon is very passionate about open source technology in the Java and Microsoft .NET worlds. His other professional interests include Mobile development, data persistence patterns, and product development. When not programming or speaking Damon enjoys raising his wonderful daughter, Brooke, home theater, and music.

Joseph D. Gradecki

Joseph D. Gradecki is a software engineer at Comprehensive Software Solutions, where he works on their SABIL product, a enterprise-level securities processing system. He has built numerous dynamic, enterprise applications using Java, AspectJ, servlets, JSPs, Resin, MySQL, BroadVision, XML, and more. He is the author of Mastering JXTA and the coauthor of MySQL and Java Developer's Guide (with Mark Matthews and Jim Cole). Joe holds Bachelors and Masters degrees in Computer Science and is currently pursuing a Ph.D.

Kate Rhodes

Kate Rhodes is a self-taught programmer, serial entrepreneur, and occasional musician with a Japanese nickname and an unstoppable urge to learn new things, solve new problems, and write open source apps. She’s got too many pets, too many projects, and too little time. If she could afford an army of programming minions she’d have one.

She’s ranked better than most and worse than some. She’s worked in impressive places you’ve possibly heard of and unimpressive places you’ve never heard of. She’s done some stuff she’s very proud of and some she isn’t. And so long as life keeps offering up interesting new challenges for her to learn from, she’s happy. If you’re overcome with curiosity you can always see what Kate and her wife are up to at www.masukomi.org.

Robert Watkins

Robert Watkins is a Brisbane, Australia-based software developer of nine years’ experience. He’s been called a programmer, a software engineer, an architect, and a few other things (many unprintable), but mostly ignores the titles and does what he has to do. These days he is mostly working in J2EE and related technologies, and is a passionate advocate of agile development techniques. When not working, he spends most of his time with his wife and twin children, and when he gets a chance, he takes time out to read Terry Pratchett novels. Robert is also one of the developers on the CruiseControl project.

Erik Meade

Erik Meade is an employee of Wells Fargo, who attended XPImmersionTwo, interviewed at XPImmersionThree, coached at XPImmersionFour, and hung out on the evenings of XPImmersionFive. He is the editor of junit.org and an evangelist of JUnit, Ant, CruiseControl, HttpUnit, Tomcat, and open source in general. He is also a contributor to JUnitPerf.
# Contents

**Introduction**  
xxv

## Part I: Key Concepts

### Chapter 1: Introduction to Extreme Programming  
1

- **XP Overview**  
  1
  - Four Values of XP  
  2
  - Five Principles of XP  
  3
  - Twelve Practices of XP  
  4
- **Adopting XP?**  
  9
- **Summary**  
  10

### Chapter 2: J2EE Deployment Concepts  
11

- **The JAR File**  
  13
  - JAR Files and Applets  
  14
  - Executable JAR Files  
  15
- **Web Applications and the WAR File**  
  15
- **Enterprise Beans and the EJB JAR File**  
  17
- **Enterprise Applications and the EAR File**  
  20
- **Conclusion**  
  21

### Chapter 3: Storing and Managing Code with CVS  
23

- **The Benefits of Versioning**  
  23
- **About CVS**  
  24
- **CVS Foundations**  
  24
  - The CVS Local Server  
  24
  - CVS Commands  
  26
- **Setting Up CVS on the Server**  
  33
- **CVS Pitfalls and Solutions**  
  34
  - Up-to-Date Check Failed for "file"  
  34
  - Recovering a File Deleted with "cvs remove"  
  34
  - CVS Files in Source Drops  
  35
<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Configure Ant</strong></td>
<td>134</td>
</tr>
<tr>
<td><strong>A Simple Example</strong></td>
<td>136</td>
</tr>
<tr>
<td>Sample Ant Build File</td>
<td>140</td>
</tr>
<tr>
<td><strong>XDoclet Tasks, Subtasks, and Templates</strong></td>
<td>141</td>
</tr>
<tr>
<td>Core Tasks</td>
<td>141</td>
</tr>
<tr>
<td>Subtasks</td>
<td>141</td>
</tr>
<tr>
<td>Templates</td>
<td>142</td>
</tr>
<tr>
<td><strong>XDoclet Merging</strong></td>
<td>143</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>144</td>
</tr>
<tr>
<td><strong>Chapter 8: Building J2EE Web Components with Ant and XDoclet</strong></td>
<td>145</td>
</tr>
<tr>
<td><strong>Servlet Example</strong></td>
<td>145</td>
</tr>
<tr>
<td><strong>Webdoclet Tags</strong></td>
<td>150</td>
</tr>
<tr>
<td><strong>Using Ant for the Configuration</strong></td>
<td>152</td>
</tr>
<tr>
<td><strong>Using Ant Properties, Init-Params, and Filtering</strong></td>
<td>154</td>
</tr>
<tr>
<td><strong>Creating Taglibs</strong></td>
<td>154</td>
</tr>
<tr>
<td>Testing the Custom Tag</td>
<td>157</td>
</tr>
<tr>
<td><strong>Using Eclipse with Webdoclet</strong></td>
<td>157</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>167</td>
</tr>
<tr>
<td><strong>Chapter 9: Building EJBs with Ant and XDoclet</strong></td>
<td>169</td>
</tr>
<tr>
<td><strong>Generating XDoclet EJBs</strong></td>
<td>169</td>
</tr>
<tr>
<td>Database Setup</td>
<td>169</td>
</tr>
<tr>
<td>Creating Entity Beans</td>
<td>170</td>
</tr>
<tr>
<td>Relations</td>
<td>176</td>
</tr>
<tr>
<td>Student Bean Code</td>
<td>177</td>
</tr>
<tr>
<td>Creating a Session Bean</td>
<td>180</td>
</tr>
<tr>
<td>Ant Target for XDoclet</td>
<td>182</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>185</td>
</tr>
<tr>
<td><strong>Chapter 10: Building Struts Apps with Ant and XDoclet</strong></td>
<td>187</td>
</tr>
<tr>
<td><strong>Brief Introduction to Struts</strong></td>
<td>187</td>
</tr>
<tr>
<td><strong>Installing Struts</strong></td>
<td>188</td>
</tr>
<tr>
<td><strong>A Sample Application</strong></td>
<td>188</td>
</tr>
<tr>
<td>Creating the ActionForm</td>
<td>188</td>
</tr>
<tr>
<td>Building the Action</td>
<td>189</td>
</tr>
<tr>
<td>Configuring Struts Config</td>
<td>190</td>
</tr>
<tr>
<td>Creating the Web.xml File</td>
<td>191</td>
</tr>
<tr>
<td>Building the Success and Failure Pages</td>
<td>192</td>
</tr>
<tr>
<td>Building the Register Page</td>
<td>193</td>
</tr>
<tr>
<td>Setup</td>
<td>194</td>
</tr>
</tbody>
</table>
Contents

Compile 195
Run 195
Struts and XDoclet 195
Summary 198

Chapter 11: Creating Custom Ant Tasks 199
How Ant Works 199
An Example Task 200
A Skeleton Task Example 201
Building a New Task 203
Accessing Properties 203
Working with <classpath> 204
Setting Up the Task 204
Using Nested Elements 205
Using the New Task 208
Summary 212

Chapter 12: Creating XDoclet Custom Tags and Templates 213
XDoclet Architecture 213
A Simple Template 216
Building an XDoclet Template 217
Summary 220

Part III: Automated Java Testing

Chapter 13: Unit Testing with JUnit 221
Overview of JUnit 222
Writing a Test Case 223
Integrating JUnit with Ant 229
Case Study: Adding an Entity Bean to the Pet Store 234
Overview of Existing JUnit Tests 234
Adding an Enterprise JavaBean to the Pet Store 243
Creating an Ant Buildfile to Deploy Our Entity Bean 254
Modifying the Test Buildfile to Test Our Entity Bean 260
Case Study Conclusion 262
Summary 262
## Contents

### Chapter 14: Load Testing with JUnitPerf 263

- JUnitPerf Concepts 264
- TimedTest Example 264
- LoadTest Example 271
- Case Study 272
  - HttpUnit Test 273
  - JMeter Configuration 274
  - Putting It All Together 275
- Summary 278

### Chapter 15: Defect Tracking with Bugzilla 279

- Enter Bugzilla 280
  - How Does Bugzilla Work? 280
- Bugzilla Setup 280
  - Installation 280
  - Configuration 284
- Bugs in Bugzilla 285
  - Setting Up the Sample Product 285
  - Entering Bugs 287
  - Maintaining Bugs–The Lifecycle of a Bug 290
  - Searching for Bugs 291
- Security 293
  - Advanced Configuration 294
  - Voting 295
- Summary 296

### Part IV: Automated J2EE Testing

### Chapter 16: Functional Testing with HttpUnit 297

- Why Functional Testing? 297
- Why HttpUnit? 298
- HttpUnit Basics 299
  - WebClient Functionality 299
  - Response Inspection and Multiple Pages 300
- Advanced Topics in HttpUnit 309
  - DOM Inspection 309
  - Headers and Cookies 310
  - Frames 311
  - SSL 311
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HttpSessionOptions</td>
<td>311</td>
</tr>
<tr>
<td>Technical Limitations</td>
<td>311</td>
</tr>
<tr>
<td><strong>Spider Example</strong></td>
<td>312</td>
</tr>
<tr>
<td>Spider Development: First Iteration</td>
<td>312</td>
</tr>
<tr>
<td>Spider Development: Second Iteration</td>
<td>314</td>
</tr>
<tr>
<td>Future Work on the Spider</td>
<td>317</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>318</td>
</tr>
</tbody>
</table>

**Chapter 17: Performance Testing with JMeter**  319

| Overview of JMeter                                                    | 319  |
| JMeter Concepts                                                       | 320  |
| **Using JMeter to Test a Web Application Navigation**                 | 322  |
| **Using JMeter to Test the Web Application's Form Entry**             | 327  |
| **Using JMeter to Test Performance of Our RDBMS**                     | 334  |
| **Case Study: The Pet Store**                                         | 335  |
| Business Need                                                         | 335  |
| Creating the Test                                                     | 337  |
| Analysis                                                              | 341  |
| Conclusion                                                            | 341  |
| **Summary**                                                           | 342  |

**Chapter 18: Cactus Primer**                                           343

| Why In-Container Testing?                                             | 343  |
| Testing in Isolation: Mock Objects                                   | 343  |
| Testing in Context: Integration Testing                              | 344  |
| A Blend of Approaches                                                | 345  |
| **Why Cactus?**                                                       | 346  |
| **Installing Cactus**                                                 | 347  |
| Server-Side Installation                                              | 347  |
| Client-Side Installation                                              | 348  |
| A Simple Example                                                      | 349  |
| **Cactus Architecture**                                               | 350  |
| Extending the Appropriate Class                                       | 351  |
| Beginning Execution                                                   | 351  |
| The beginXXX() Method                                                 | 351  |
| Calling the Redirector Servlet                                        | 353  |
| Server-Side Setup                                                     | 353  |
| The Implicit Objects                                                  | 354  |
| Server-Side Execution                                                 | 355  |
| setUp() and tearDown()                                                | 355  |
## Contents

- testXXX() 356
  - Results Collection and Postprocessing 356
- endXXX() 357
- **Cactus with Ant** 359
  - The Strategy 359
- **Summary** 360

### Chapter 19: Testing Servlets and Filters with Cactus 361

- **Testing Servlets** 361
  - The Code Under Test 362
- **Testing Filters** 365
  - An Example 366
- **Summary** 369

### Chapter 20: JspTestCases and Testing Custom Tags with Cactus 371

- **Testing Simple Tags** 372
- **Testing Tag Attributes and Page Interaction** 373
  - Managing Information in Scopes 375
- **Testing Body Tags** 376
  - productLink Tag Example 376
- **Testing Iteration Tags** 378
  - Repeat Tag Example 378
- **Server-Side Assertion Facilities** 380
- **Working with Nested Tags** 381
- **Testing Auxiliary Tag Components** 384
  - TagExtraInfo Classes 384
  - Tag Library Descriptor Testing 384
- **Case Study: The Pet Store with Custom Tags** 385
  - The Business Need 386
  - Finding a Starting Point 386
  - The TestCase 386
- **Summary** 391

### Chapter 21: Testing EJBs with Cactus 393

- **Creating an Ant Buildfile to Run the Test** 396
- **Case Study: Adding an Entity Bean to the Pet Store** 397
  - Adding an Enterprise JavaBean to the Pet Store 397
- **Summary** 408
Chapter 22: Code Coverage with jcoverage 409

Why Code Coverage? 410
  Types of Coverage 411
jcoverage Primer 411
  jcoverage as an Ant task 411
  jcoverage as a Maven Plugin 416
100% Go Code and 80% Exception Handling 416
  Statement or Line Coverage 417
  Branch Coverage 417
  Exception Handling 419
  Diminishing Return 421
  Setting Goals 422
Other Tools: GroboUtils 422
Conclusion 423

Part V: Automated Swing Testing

Chapter 23: Swing Testing with Jemmy 425

Jemmy Primer 426
Testing a Swing Application with Jemmy 427
Summary 436

Chapter 24: Swing Testing with jfcUnit 437

jfcUnit Primer 438
Testing a Swing Application with jfcUnit 439
Summary 445

Chapter 25: Swing Testing with Abbot 447

Abbot and Costello Primer 448
Recording and Using Our Test 452
  Cleaning Up 454
  Putting it All Together 454
Advantages of Abbot 455
# Contents

## Part VI: Continuous Integration, Project Management, and IDEs

### Chapter 26: Managing Projects with Maven

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maven Primer</td>
<td>458</td>
</tr>
<tr>
<td>Installing on Unix</td>
<td>459</td>
</tr>
<tr>
<td>Installing on Windows</td>
<td>459</td>
</tr>
<tr>
<td>Verifying the Install</td>
<td>459</td>
</tr>
<tr>
<td>Installing Plug-ins</td>
<td>461</td>
</tr>
<tr>
<td>The Project Object Model (POM)</td>
<td>462</td>
</tr>
<tr>
<td>Project Properties</td>
<td>467</td>
</tr>
<tr>
<td>maven.xml</td>
<td>467</td>
</tr>
<tr>
<td>Project Layout</td>
<td>467</td>
</tr>
<tr>
<td>Maven Architecture</td>
<td>469</td>
</tr>
<tr>
<td>Maven Properties</td>
<td>471</td>
</tr>
<tr>
<td>Understanding Jelly</td>
<td>472</td>
</tr>
<tr>
<td>Project Management with Maven</td>
<td>475</td>
</tr>
<tr>
<td>Defining Your Project Structure</td>
<td>475</td>
</tr>
<tr>
<td>Defining Your Root Project Descriptor</td>
<td>476</td>
</tr>
<tr>
<td>Your project.properties File</td>
<td>478</td>
</tr>
<tr>
<td>Your maven.xml File</td>
<td>478</td>
</tr>
<tr>
<td>Maven multiproject (reactor) and Subprojects</td>
<td>478</td>
</tr>
<tr>
<td>Calling Subprojects from the Main Project Using reactor</td>
<td>478</td>
</tr>
<tr>
<td>Building the Web Site</td>
<td>479</td>
</tr>
<tr>
<td>Case Study: Managing Our Pet Store Project with Maven</td>
<td>480</td>
</tr>
<tr>
<td>The New Directory Structure</td>
<td>480</td>
</tr>
<tr>
<td>Defining Our Dependencies</td>
<td>481</td>
</tr>
<tr>
<td>The New Project Descriptors and How They Work</td>
<td>482</td>
</tr>
<tr>
<td>Using the Eclipse Plug-in</td>
<td>484</td>
</tr>
<tr>
<td>Building the Respective Pieces</td>
<td>484</td>
</tr>
<tr>
<td>Adding In Custom Deployment Options Using Jelly</td>
<td>485</td>
</tr>
<tr>
<td>Deploying the Web Application to Tomcat or Other Containers</td>
<td>486</td>
</tr>
<tr>
<td>Building the Web Site</td>
<td>486</td>
</tr>
<tr>
<td>Summary</td>
<td>488</td>
</tr>
</tbody>
</table>

### Chapter 27: Automating Continuous Integration with CruiseControl

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basics of Using CruiseControl</td>
<td>491</td>
</tr>
<tr>
<td>The MVCHelloWorld Project</td>
<td>491</td>
</tr>
<tr>
<td>CruiseControl Components</td>
<td>497</td>
</tr>
<tr>
<td>Source Control Systems</td>
<td>498</td>
</tr>
</tbody>
</table>
Contents

Ant Key Tasks 532
  Ant Task 532
  Antcall Task 532
  Available Task 533
  Echo Task 534
  Parameters 534
  Fail Task 534
  Property Task 534
  Taskdef Task 535

Directory and File System Tasks 535
  Chmod Task 535
  Copy Task 536
  Delete Task 537
  Mkdir Task 538
  Move Task 538
  Touch Task 538

External Execution Tasks 539
  Apply Task 539
  Exec Task 540
  Execon Task 541
  Java Task 542
  Javac Task 543
  Sql Task 544

File Library Tasks 545
  Jar Task 545
  War Task 546
  Zip Task 547
  Unjar, Unzip, and Unwar Tasks 548

File Readers, Writers, and Tokenizers 548
  Replace Task 548
  Mail Task 549

Source Control Tasks 549
  Cvs Task 550
  Get Task 550

Some Key Optional Tasks 551
  Junit Task 551
  JunitReport Task 552

Frequently Used Nested Parameters and Elements 553
  Classpath Element 553
  Mapper Element 554
  Fileset Element 555
  Patternset Element 555
  Zipfileset Element 556
Chapter 32: Cactus API Reference 607

Package org.apache.cactus 607
  Class AbstractTestCase 607
  Class Cookie 610
  FilterTestCase 614
  Class JspTestCase 615
  Class ServiceDefinition 616
  Class ServiceEnumeration 616
  Class ServletTestCase 616
  Class ServletTestRequest Deprecated 620
  Class ServletURL 621
  Class WebRequest 622
  ClassWebResponse 625
  Class WebTestResult 626

Package org.apache.cactus.util 627
  Class AssertUtils Deprecated 627
  Class ClientCookie Deprecated 628

Package org.apache.cactus.server 629
  Class FilterConfigWrapper 629
  Class HttpServletRequestWrapper 630
  Class PageContextWrapper 631
  Class RequestDispatcherWrapper 631
  Class ServletConfigWrapper 632
  Class ServletContextWrapper 633

Chapter 33: HttpUnit API Reference 635

Package com.meterware.httpunit 635
  Class AuthorizationRequiredException 636
  Class GetMethodWebRequest 636
  Class HTMLSegment 637
  Class HttpException 639
  Class HttpInternalErrorException 640
  Class HttpNotFoundException 640
  Class HttpUnitOptions 640
  Class HttpUnitUtils 642
  Class IllegalRequestParameterException 643
  Class MessageBodyWebRequest 644
  Class PostMethodWebRequest 644
  Class PutMethodWebRequest 645
  Class SubmitButton 646
Contents

Class TableCell 646
Class WebClient 647
Class WebConversation 651
Class WebForm 651
Class WebLink 654
Class WebRequest 655
ClassWebResponse 659
Class WebTable 662

Chapter 34: JUnitPerf API Reference 665

Package com.clarkware.junitperf 665
Class ConstantTimer 665
Class LoadTest 666
Class RandomTimer 669
Class ThreadBarrier 670
Class ThreadedTest 671
Class ThreadedTestGroup 671
Class TimedTest 672
Interface Timer 674

Part VIII: Appendix

Appendix A: Example Applications Used in This Book 675

Simple Example: Model 2 Hello World 676
Case Study Example: The Pet Store 676
Baseline Version of the Pet Store 677
Model Classes (Public Interface) 678
dbmodel Classes (Implementation) 679
Database Schema 680
Web Interface 681
Build System 683
Setting Up the Database with Test Data: A Small Taste Of Ant 687

Complete Listings 692
Model Package Complete Listings 692
dbmodel Package Complete Listings 695
Test Package Complete Listings 701
JSPs and Web.xml Complete Listings 707

Conclusion 712

Index 713
Introduction

This book describes techniques for implementing the Extreme Programming practices of automated testing and continuous integration using open source tools.

Let’s unpack that statement. Automated testing and continuous integration are two of the twelve core practices of the Extreme Programming (XP) software development methodology. Extreme Programming is a lightweight software development process that focuses on feedback, communication, simplicity, and courage. The full XP process is summarized in Chapter 1; suffice it to say for now that it consists of common-sense development practices practiced religiously and in concert.

Two of these common-sense practices are testing and frequent integration. Almost no software development shop would consider leaving these steps out of its process entirely—after all, a system has to be integrated to ship, and it must be tested to ensure that the customers accept the shipment. Thanks to the dot-com shakeout, most of the shops that did skip these practices are now out of business. Still, many software companies either struggle with implementing these processes, or acknowledge that they should be done but claim that “things are just too busy right now” to do them. This book explains and demonstrates the use of software tools to help put these valuable practices into place.

Why Spend So Much Time on the Tools?

We focus on tools, ironically enough, because XP is a human-centric development philosophy. It recognizes that the key challenges of writing software are human challenges—such as getting people to work together, helping programmers learn, and managing emotions. Its four core values (communication, feedback, simplicity, and courage) are human values. Most books published on XP so far have focused on the human issues: outlining the philosophy, spreading the ideology (Extreme Programming Explained was described by Kent Beck as a manifesto), and talking about the feeling of writing software. By doing so, Kent Beck and the originators of XP have followed their own philosophy: Solve the most pressing problems first. However, the current books do not cover the technical details of implementing some of their practices. That’s where books like this one come in.

We will explain how to set up continuous integration and automated testing in a Java environment (specifically J2EE, although most of the tools apply generally). Technical detail will be addressed, and we will offer loads of examples to show the tools in action. Specifically, we will cover how to use Abbot, Ant, AntHill, Bugzilla, Cactus, CruiseControl, CVS, Eclipse, HttpUnit, Jemmy, jfcUnit, JMeter, JUnit, Maven, and other tools to write automated tests and achieve continuous integration.
Who Should Read this Book

Although this book speaks from an XP perspective, you need not practice XP to benefit from it. Anyone who needs help automating testing and integration can benefit from the tools and practices outlined herein. If you know nothing about Extreme Programming, you should probably read the rest of this Introduction, along with Chapter 1 to get a sense of the practices covered in this book, both alone and in their XP context. In particular the Introduction touches on the value of automated testing and continuous integration for all developers.

This book assumes you are at least a moderately experienced Java developer. Because it covers the application of testing and integration tools to the J2EE platform, this book also expects familiarity with J2EE technologies and development practices. Those who are not interested in J2EE applications will still find plenty of worthwhile material, because most of these tools can be applied to almost any Java (or, in the case of JMeter and HttpUnit, even non-Java) software project. Developers who aren’t familiar with J2EE but who want to apply these tools and techniques to a J2EE application may also want to pick up a comprehensive J2EE book like Expert One-on-One: J2EE Design and Development by Rod Johnson.

Why Open Source?

It is hard to miss the growing prominence of open source development in software engineering. Open source development tools offer compelling advantages over traditional tools—especially for XP development. The advantages fall into two categories. First, open source tools are practical. Second, the open source philosophy is closely aligned with XP.

Open source tool offer several practical advantages:

The price is right. Open source software can almost always be obtained for free; all the tools we cover in this book can be downloaded at no cost from the Internet. Free software means no immediate overhead for yourself or your company, which is always a benefit, but in this case not the major one. The major benefit in the case of these tools is that their adoption will not be hampered by corporate red tape or management worried about buying into the latest fad. Once you have downloaded JUnit, for example, and you’ve fallen in love with it and spread it to your team—speeding development and improving quality—no one will want to throw roadblocks in your way. Starting the adoption of XP by asking for $7,500 worth of whiz-bang deployment tools might invite skepticism.

The tools are high quality. Programmers use open source development tools every day. Because improving the tool means improving their immediate situation, open source development tools often receive many contributions and bug fixes. Improvement and features come fast and furious.

The tools are the standard. Especially in the case of JUnit and Ant, the tools covered in this book are the standards in their field. Countless open source projects use Ant, and JUnit (upon which several of the tools are based) was written by Kent Beck (the godfather of XP) and Erich Gamma (co-author of the OO classic Design Patterns: Elements of Reusable Object-Oriented Software).
Synergy Between XP and Open Source

Extreme Programming and open source development are closely aligned ideologically. Both foster an open, giving style of collaborative development—they share a certain *vibe*, if you will. Both philosophies acknowledge human weakness—no code is perfect, and the assistance of others in finding and fixing problems is gratefully acknowledged. All open source code is commonly owned (as XP would have it). Many open source projects use and benefit from automated testing, which is especially important when code from a wide variety of sources must be integrated. Both systems demand small, incremental releases. Of course, both philosophies also focus heavily on the code—open source is founded on the premise that reading code is enjoyable, educational, and helpful.

The list could continue for quite a while. By using open source tools (and by giving back to the open source community in the form of feedback, assistance, and code) you practice some of the values and processes that make XP great.

**Read the Source**

If you are looking for more information than this book provides on any of the tools, the best place to start is the source code. In addition to containing the Javadoc (another handy reference), the source code is the definitive authority on the tool’s behavior. Open-source software exists because (in addition to liking free stuff) programmers value the ability to dig into the work of fellow coders. By reading the source carefully, you can gain insight into how the program works, insight into the domain, and, if you are lucky, insight into the arcane art of programming itself. If you are unlucky enough to encounter a bug while using the tool, having the source handy can help you determine where the bug lies.

**Automated Testing: A Summary**

XP regards testing as central to the activity of software development. To quote Dan Rawsthorne from the afterword of *Extreme Programming Installed*, “XP works because it is validation-centric rather than product-centric.” Testing software continuously validates that the software works and that it meets the customer’s requirements. Automating the tests ensures that testing will in fact be continuous. Without testing, a team is just guessing that its software meets those requirements. XP cannot be done without automated testing, nor can development be done successfully without it. All software projects need to satisfy the intended customer and to be free of defects.

**Tests and Refactoring**

Another core XP practice is refactoring (changing existing code for simplicity, clarity, and/or feature addition). Refactoring cannot be accomplished without tests. If you don’t practice XP, you may not be refactoring religiously. Even the most stable or difficult-to-change projects require occasional modification. To do it right, programmers will have to change the existing design. That’s where automated testing comes in.
Object-oriented programming (and, to a lesser extent, other programming styles) separates interface from implementation. In theory, this means you can change the underlying logic behind a class or method, and dependent code will handle the change seamlessly. Entire books have been written about this powerful abstraction. However, if in practice the programmers are scared to change the underlying logic for fear of disturbing code that interacts with the interface, then this separation might as well not exist. Comprehensive tests (run frequently) verify how the system should work and allow the underlying behavior to change freely. Any problems introduced during a change are caught by the tests. If Design A and Design B produce equivalent results when tested, the code can be migrated from one to the other freely. With testing in place, programmers refactor with confidence, the code works, and the tests prove it.

Types of Automated Tests

Unit tests are the most talked-about test in XP; however, they are only a part of the testing picture. Unit tests cooperate with integration tests, functional tests, and auxiliary tests (performance tests, regression tests, and so on) to ensure that the system works totally.

Unit Tests: JUnit

Unit tests are the first (and perhaps the most critical) line of tests in the XP repertoire. Writing a unit test involves taking a unit of code and testing everything that could possibly break. A unit test usually exercises all the methods in the public interface of a class. Good unit tests do not necessarily test every possible permutation of class behavior, nor do they test ultra-simple methods (simple accessors come to mind); rather, they provide a common-sense verification that the code unit behaves as expected. With this verification, the public interface gains meaning. This approach makes changing unit behavior easier, and also provides a convenient (and verifiable) guide to the behavior of the unit. Developers can consult a test to discover the intended use of a class or method.

In XP, unit tests become part of the cycle of everyday coding. Ideally, programmers write tests before the code, and use the test as a guide to assist in implementation. The authors both work in this mode, and we find ourselves unable to live without the guidance and corrective influence of unit tests. After a unit is complete, the team adds the test to the project’s test suite. This suite of unit tests runs multiple times per day, and all the tests always pass. This sounds extreme; however, a 100 percent pass rate on unit tests is far more sane than the alternative: a piece of vital production code that does not work. (If the code isn’t vital, why is it in the project?)

Verifying each class builds a higher-quality system because it ensures that the building blocks work. Unit tests also lead the way toward clean architecture. If a developer writes a test three times for the same code in different locations, laziness and irritation will compel her to move the code to a separate location.

JUnit is a lightweight testing framework written by Erich Gamma and Kent Beck (one of the chief proponents of XP). The authors based its design on SUnit, a successful and popular unit-testing framework written by Beck for Smalltalk. The simplicity of the framework lends itself to rapid adoption and extension. All the testing tools covered in this book (with the exception of JMeter, a GUI tool) interact with or extend the JUnit frame.