

---

# Agile Software Development

*Principles, Patterns, and Practices*

*Robert Cecil Martin*

Alan Apt Series



Pearson Education, Inc.  
Upper Saddle River, New Jersey 07458

---

# Brief Contents

<b><i>Section 1</i></b>	<b>Agile Development</b>	<b>1</b>
<b><i>Chapter 1</i></b>	<b>Agile Practices</b>	<b>3</b>
<b><i>Chapter 2</i></b>	<b>Overview of Extreme Programming</b>	<b>11</b>
<b><i>Chapter 3</i></b>	<b>Planning</b>	<b>19</b>
<b><i>Chapter 4</i></b>	<b>Testing</b>	<b>23</b>
<b><i>Chapter 5</i></b>	<b>Refactoring</b>	<b>31</b>
<b><i>Chapter 6</i></b>	<b>A Programming Episode</b>	<b>43</b>
<b><i>Section 2</i></b>	<b>Agile Design</b>	<b>85</b>
<b><i>Chapter 7</i></b>	<b>What Is Agile Design?</b>	<b>87</b>
<b><i>Chapter 8</i></b>	<b>SRP: The Single-Responsibility Principle</b>	<b>95</b>
<b><i>Chapter 9</i></b>	<b>OCP: The Open–Closed Principle</b>	<b>99</b>
<b><i>Chapter 10</i></b>	<b>LSP: The Liskov Substitution Principle</b>	<b>111</b>
<b><i>Chapter 11</i></b>	<b>DIP: The Dependency-Inversion Principle</b>	<b>127</b>
<b><i>Chapter 12</i></b>	<b>ISP: The Interface-Segregation Principle</b>	<b>135</b>
<b><i>Section 3</i></b>	<b>The Payroll Case Study</b>	<b>147</b>
<b><i>Chapter 13</i></b>	<b>COMMAND and ACTIVE OBJECT</b>	<b>151</b>
<b><i>Chapter 14</i></b>	<b>TEMPLATE METHOD &amp; STRATEGY: Inheritance vs. Delegation</b>	<b>161</b>
<b><i>Chapter 15</i></b>	<b>FACADE and MEDIATOR</b>	<b>173</b>
<b><i>Chapter 16</i></b>	<b>SINGLETON and MONOSTATE</b>	<b>177</b>

<i>Chapter 17</i>	<b>NULL OBJECT</b>	<b>189</b>
<i>Chapter 18</i>	<b>The Payroll Case Study: Iteration One Begins</b>	<b>193</b>
<i>Chapter 19</i>	<b>The Payroll Case Study: Implementation</b>	<b>205</b>
<i>Section 4</i>	<b>Packaging the Payroll System</b>	<b>251</b>
<i>Chapter 20</i>	<b>Principles of Package Design</b>	<b>253</b>
<i>Chapter 21</i>	<b>FACTORY</b>	<b>269</b>
<i>Chapter 22</i>	<b>The Payroll Case Study (Part 2)</b>	<b>275</b>
<i>Section 5</i>	<b>The Weather Station Case Study</b>	<b>291</b>
<i>Chapter 23</i>	<b>COMPOSITE</b>	<b>293</b>
<i>Chapter 24</i>	<b>OBSERVER—Backing into a Pattern</b>	<b>297</b>
<i>Chapter 25</i>	<b>ABSTRACT SERVER, ADAPTER, and BRIDGE</b>	<b>317</b>
<i>Chapter 26</i>	<b>PROXY and STAIRWAY TO HEAVEN: Managing Third Party APIs</b>	<b>327</b>
<i>Chapter 27</i>	<b>Case Study: Weather Station</b>	<b>355</b>
<i>Section 6</i>	<b>The ETS Case Study</b>	<b>385</b>
<i>Chapter 28</i>	<b>VISITOR</b>	<b>387</b>
<i>Chapter 29</i>	<b>STATE</b>	<b>419</b>
<i>Chapter 30</i>	<b>The ETS Framework</b>	<b>443</b>
<i>Appendix A</i>	<b>UML Notation I: The CGI Example</b>	<b>467</b>
<i>Appendix B</i>	<b>UML Notation II: The STATMUX</b>	<b>489</b>
<i>Appendix C</i>	<b>A Satire of Two Companies</b>	<b>507</b>
<i>Appendix D</i>	<b>The Source Code Is the Design</b>	<b>517</b>
<b>Index</b>		<b>525</b>

---

# Contents

<b>Foreword</b>	<b>iii</b>
<b>Preface</b>	<b>iv</b>
<b>About the Authors</b>	<b>ix</b>
<b>List of Design Patterns</b>	<b>xxii</b>
<hr/>	
<b><i>Section 1</i> Agile Development</b>	<b>1</b>
<hr/>	
<b><i>Chapter 1</i> Agile Practices</b>	<b>3</b>
The Agile Alliance	4
The Manifesto of the Agile Alliance	4
Principles	6
Conclusion	8
Bibliography	9
 <b><i>Chapter 2</i> Overview of Extreme Programming</b>	 <b>11</b>
The Practices of Extreme Programming	11
Customer Team Member	11
User Stories	12
Short Cycles	12
Acceptance Tests	13
Pair Programming	13
Test-Driven Development	14
Collective Ownership	14
Continuous Integration	14
Sustainable Pace	15
Open Workspace	15
The Planning Game	15
Simple Design	15
Refactoring	16
Metaphor	16
Conclusion	17
Bibliography	17

---

<b>Chapter 3</b>	<b>Planning</b>	<b>19</b>
	Initial Exploration	20
	Spiking, Splitting, and Velocity	20
	Release Planning	20
	Iteration Planning	21
	Task Planning	21
	The Halfway Point	22
	Iterating	22
	Conclusion	22
	Bibliography	22
<b>Chapter 4</b>	<b>Testing</b>	<b>23</b>
	Test Driven Development	23
	An Example of Test-First Design	24
	Test Isolation	25
	Serendipitous Decoupling	26
	Acceptance Tests	27
	Example of Acceptance Testing	27
	Serendipitous Architecture	29
	Conclusion	29
	Bibliography	29
<b>Chapter 5</b>	<b>Refactoring</b>	<b>31</b>
	Generating Primes: A Simple Example of Refactoring	32
	The Final Reread	38
	Conclusion	42
	Bibliography	42
<b>Chapter 6</b>	<b>A Programming Episode</b>	<b>43</b>
	The Bowling Game	44
	Conclusion	82
<hr/> <b>Section 2 Agile Design</b>		<b>85</b>
	Symptoms of Poor Design	85
	Principles	86
	Smells and Principles	86
	Bibliography	86
<b>Chapter 7</b>	<b>What Is Agile Design?</b>	<b>87</b>
	What Goes Wrong with Software?	87
	Design Smells—The Odors of Rotting Software	88
	What Stimulates the Software to Rot?	89
	Agile Teams Don't Allow the Software to Rot	90
	The “Copy” Program	90
	Agile Design of the Copy Example	93
	How Did the Agile Developers Know What to Do?	94
	Keeping the Design As Good As It Can Be	94
	Conclusion	94
	Bibliography	94

<b>Chapter 8</b>	<b>SRP: The Single-Responsibility Principle</b>	<b>95</b>
	<i>A CLASS SHOULD HAVE ONLY ONE REASON TO CHANGE.</i>	
	SRP: The Single-Responsibility Principle	95
	What Is a Responsibility?	97
	Separating Coupled Responsibilities	97
	Persistence	98
	Conclusion	98
	Bibliography	98
<b>Chapter 9</b>	<b>OCP: The Open–Closed Principle</b>	<b>99</b>
	<i>SOFTWARE ENTITIES (CLASSES, MODULES, FUNCTIONS, ETC.) SHOULD BE OPEN FOR EXTENSION, BUT CLOSED FOR MODIFICATION.</i>	
	OCP: The Open–Closed Principle	99
	Description	100
	Abstraction Is the Key	100
	The Shape Application	101
	Violating the OCP	101
	Conforming to the OCP	103
	OK, I Lied	104
	Anticipation and “Natural” Structure	105
	Putting the “Hooks” In	105
	Using Abstraction to Gain Explicit Closure	106
	Using a “Data-Driven” Approach to Achieve Closure	107
	Conclusion	108
	Bibliography	109
<b>Chapter 10</b>	<b>LSP: The Liskov Substitution Principle</b>	<b>111</b>
	<i>SUBTYPES MUST BE SUBSTITUTABLE FOR THEIR BASE TYPES.</i>	
	LSP: The Liskov Substitution Principle	111
	A Simple Example of a Violation of the LSP	112
	Square and Rectangle, a More Subtle Violation	113
	The Real Problem	115
	Validity Is Not Intrinsic	116
	ISA Is about Behavior	116
	Design by Contract	117
	Specifying Contracts in Unit Tests	117
	A Real Example	117
	Motivation	118
	Problem	119
	A Solution That Does <i>Not</i> Conform to the LSP	120
	An LSP-Compliant Solution	120
	Factoring Instead of Deriving	121
	Heuristics and Conventions	124
	Degenerate Functions in Derivatives	124
	Throwing Exceptions from Derivatives	124
	Conclusion	125
	Bibliography	125

<b>Chapter 11 DIP: The Dependency-Inversion Principle</b>	<b>127</b>
<i>A. HIGH-LEVEL MODULES SHOULD NOT DEPEND UPON LOW-LEVEL MODULES. BOTH SHOULD DEPEND ON ABSTRACTIONS.</i>	
<i>B. ABSTRACTIONS SHOULD NOT DEPEND ON DETAILS. DETAILS SHOULD DEPEND ON ABSTRACTIONS.</i>	
DIP: The Dependency-Inversion Principle	127
Layering	128
An Inversion of Ownership	128
Depend on Abstractions	129
A Simple Example	130
Finding the Underlying Abstraction	131
The Furnace Example	132
Dynamic v. Static Polymorphism	133
Conclusion	134
Bibliography	134
<b>Chapter 12 ISP: The Interface-Segregation Principle</b>	<b>135</b>
Interface Pollution	135
Separate Clients Mean Separate Interfaces	137
The Backwards Force Applied by Clients Upon Interfaces	137
<i>CLIENTS SHOULD NOT BE FORCED TO DEPEND ON METHODS THAT THEY DO NOT USE.</i>	
ISP: The Interface-Segregation Principle	137
Class Interfaces v. Object Interfaces	138
Separation through Delegation	138
Separation through Multiple Inheritance	139
The ATM User Interface Example	139
The Polyad v. the Monad	144
Conclusion	145
Bibliography	145
<b>Section 3 The Payroll Case Study</b>	<b>147</b>
Rudimentary Specification of the Payroll System	148
Exercise	148
Use Case 1: Add New Employee	148
Use Case 2: Deleting an Employee	149
Use Case 3: Post a Time Card	149
Use Case 4: Posting a Sales Receipt	149
Use Case 5: Posting a Union Service Charge	150
Use Case 6: Changing Employee Details	150
Use Case 7: Run the Payroll for Today	150
<b>Chapter 13 COMMAND and ACTIVE OBJECT</b>	<b>151</b>
Simple Commands	152
Transactions	153
Physical and Temporal Decoupling	154
Temporal Decoupling	154
UNDO	154

ACTIVE OBJECT	155
Conclusion	159
Bibliography	159
<b>Chapter 14 TEMPLATE METHOD &amp; STRATEGY: Inheritance vs. Delegation</b>	<b>161</b>
TEMPLATE METHOD	162
Pattern Abuse	164
Bubble Sort	165
STRATEGY	168
Sorting Again	170
Conclusion	172
Bibliography	172
<b>Chapter 15 FACADE and MEDIATOR</b>	<b>173</b>
FACADE	173
MEDIATOR	174
Conclusion	176
Bibliography	176
<b>Chapter 16 SINGLETON and MONOSTATE</b>	<b>177</b>
SINGLETON	178
Benefits of the SINGLETON	179
Costs of the SINGLETON	179
SINGLETON in Action	179
MONOSTATE	180
Benefits of MONOSTATE	182
Costs of MONOSTATE	182
MONOSTATE in Action	182
Conclusion	187
Bibliography	187
<b>Chapter 17 NULL OBJECT</b>	<b>189</b>
Conclusion	192
Bibliography	192
<b>Chapter 18 The Payroll Case Study: Iteration One Begins</b>	<b>193</b>
Introduction	193
Specification	193
Analysis by Use Cases	194
Adding Employees	195
Deleting Employees	196
Posting Time Cards	196
Posting Sales Receipts	197
Posting a Union Service Charge	197
Changing Employee Details	198
Payday	199
Reflection: What Have We Learned?	201
Finding the Underlying Abstractions	201
The Schedule Abstraction	201



Payment Methods	202
Affiliations	202
Conclusion	203
Bibliography	203
<b>Chapter 19 The Payroll Case Study: Implementation</b>	<b>205</b>
Adding Employees	206
The Payroll Database	207
Using TEMPLATE METHOD to Add Employees	209
Deleting Employees	212
Global Variables	213
Time Cards, Sales Receipts, and Service Charges	214
Changing Employees	220
Changing Classification	224
What Was I Smoking?	229
Paying Employees	233
Do We Want Developers Making Business Decisions?	235
Paying Salaried Employees	235
Paying Hourly Employees	237
Pay Periods: A Design Problem	241
Main Program	248
The Database	248
Summary of Payroll Design	249
History	249
Resources	250
Bibliography	250
<b>Section 4 Packaging the Payroll System</b>	<b>251</b>
<b>Chapter 20 Principles of Package Design</b>	<b>253</b>
Designing with Packages?	253
Granularity: The Principles of Package Cohesion	254
The Reuse–Release Equivalence Principle (REP)	254
<i>THE GRANULE OF REUSE IS THE GRANULE OF RELEASE.</i>	
The Common-Reuse Principle (CRP)	255
<i>THE CLASSES IN A PACKAGE ARE REUSED TOGETHER. IF YOU REUSE ONE OF THE CLASSES IN A PACKAGE, YOU REUSE THEM ALL.</i>	
The Common-Closure Principle (CCP)	256
<i>THE CLASSES IN A PACKAGE SHOULD BE CLOSED TOGETHER AGAINST THE SAME KINDS OF CHANGES. A CHANGE THAT AFFECTS A PACKAGE AFFECTS ALL THE CLASSES IN THAT PACKAGE AND NO OTHER PACKAGES.</i>	
Summary of Package Cohesion	256
Stability: The Principles of Package Coupling	256
The Acyclic-Dependencies Principle (ADP)	256
<i>ALLOW NO CYCLES IN THE PACKAGE DEPENDENCY GRAPH.</i>	
The Weekly Build	257
Eliminating Dependency Cycles	257
The Effect of a Cycle in the Package Dependency Graph	258

Breaking the Cycle	259
The “Jitters”	259
Top-Down Design	260
The Stable-Dependencies Principle (SDP)	261
<i>DEPEND IN THE DIRECTION OF STABILITY.</i>	
Stability	261
Stability Metrics	262
Not All Packages Should Be Stable	263
Where Do We Put the High-level Design?	264
The Stable-Abstractions Principle (SAP)	264
<i>A PACKAGE SHOULD BE AS ABSTRACT AS IT IS STABLE.</i>	
Measuring Abstraction	265
The Main Sequence	265
Distance from the Main Sequence	266
Conclusion	268
<b>Chapter 21 FACTORY</b>	<b>269</b>
A Dependency Cycle	271
Substitutable Factories	272
Using Factories for Test Fixtures	273
How Important Is It to Use Factories?	274
Conclusion	274
Bibliography	274
<b>Chapter 22 The Payroll Case Study (Part 2)</b>	<b>275</b>
Package Structure and Notation	276
Applying the Common Closure Principle (CCP)	277
Applying the Reuse–Release Equivalency Principle (REP)	278
Coupling and Encapsulation	279
Metrics	281
Applying the Metrics to the Payroll Application	282
Object Factories	285
The Object Factory for <code>TransactionImplementation</code>	286
Initializing the Factories	286
Rethinking the Cohesion Boundaries	287
The Final Package Structure	287
Conclusion	290
Bibliography	290
<hr/> <b>Section 5 The Weather Station Case Study</b>	<hr/> <b>291</b>
<b>Chapter 23 COMPOSITE</b>	<b>293</b>
Example: Composite Commands	294
Multiplicity or Not Multiplicity	295
<b>Chapter 24 OBSERVER—Backing into a Pattern</b>	<b>297</b>
The Digital Clock	297

Conclusion	314
The Use of Diagrams in this Chapter	314
The OBSERVER Pattern	315
How OBSERVER Manages the Principles of OOD	316
Bibliography	316
<b>Chapter 25 ABSTRACT SERVER, ADAPTER, and BRIDGE</b>	<b>317</b>
ABSTRACT SERVER	318
Who Owns the Interface?	318
Adapter	319
The Class Form of ADAPTER	319
The Modern Problem, ADAPTERS and LSP	320
BRIDGE	322
Conclusion	324
Bibliography	325
<b>Chapter 26 PROXY and STAIRWAY TO HEAVEN: Managing Third Party APIs</b>	<b>327</b>
PROXY	327
Proxifying the Shopping Cart	332
Summary of PROXY	344
Dealing with Databases, Middleware, and Other Third Party Interfaces	345
STAIRWAY TO HEAVEN	347
Example of STAIRWAY TO HEAVEN	348
Other Patterns That Can Be Used with Databases	353
Conclusion	354
Bibliography	354
<b>Chapter 27 Case Study: Weather Station</b>	<b>355</b>
The Cloud Company	355
The WMS-LC Software	356
Language Selection	357
Nimbus-LC Software Design	357
24-Hour History and Persistence	368
Implementing the HiLo Algorithms	371
Conclusion	379
Bibliography	379
Nimbus-LC Requirements Overview	379
Usage Requirements	379
24-Hour History	379
User Setup	379
Administrative Requirements	380
Nimbus-LC Use Cases	380
Actors	380
Use Cases	380
Measurement History	380
Setup	381
Administration	381
Nimbus-LC Release Plan	381
Introduction	381
Release I	381

	Risks	382
	Deliverable(s)	382
	Release II	382
	Use Cases Implemented	382
	Risks	383
	Deliverable(s)	383
	Release III	383
	Use Cases Implemented	383
	Risks	383
	Deliverable(s)	383
<b>Section 6</b>	<b>The ETS Case Study</b>	<b>385</b>
<b>Chapter 28</b>	<b>VISITOR</b>	<b>387</b>
	The VISITOR Family of Design Patterns	388
	VISITOR	388
	VISITOR is Like a Matrix	391
	ACYCLIC VISITOR	391
	ACYCLIC VISITOR Is Like a Sparse Matrix	396
	Using VISITOR in Report Generators	396
	Other Uses of VISITOR	402
	DECORATOR	403
	Multiple Decorators	406
	EXTENSION OBJECT	408
	Conclusion	418
	Reminder	418
	Bibliography	418
<b>Chapter 29</b>	<b>STATE</b>	<b>419</b>
	Overview of Finite State Automata	419
	Implementation Techniques	421
	Nested Switch/Case Statements	421
	Interpreting Transition Tables	424
	The STATE Pattern	426
	SMC—The State-Machine Compiler	429
	Where Should State Machines be Used?	432
	High-Level Application Policies for GUIs	432
	GUI Interaction Controllers	433
	Distributed Processing	433
	Conclusion	434
	Listings	434
	Turnstile.java Using Table Interpretation	434
	Turnstile.java Generated by SMC, and Other Support Files	437
	Bibliography	441
<b>Chapter 30</b>	<b>The ETS Framework</b>	<b>443</b>
	Introduction	443
	Project Overview	443

Early History 1993–1994	445
Framework?	445
Framework!	446
The 1994 Team	446
The Deadline	446
The Strategy	446
Results	447
Framework Design	448
The Common Requirements of the Scoring Applications	448
The Design of the Scoring Framework	450
A Case for TEMPLATE METHOD	453
Write a Loop Once	454
The Common Requirements of the Delivery Applications	456
The Design of the Delivery Framework	457
The Taskmaster Architecture	462
Conclusion	465
Bibliography	466
<b>Appendix A UML Notation I: The CGI Example</b>	<b>467</b>
Course Enrollment System: Problem Description	468
Actors	469
Use Cases	469
The Domain Model	472
The Architecture	476
Abstract Classes and Interfaces in Sequence Diagrams	485
Summary	486
Bibliography	487
<b>Appendix B UML Notation II: The STATMUX</b>	<b>489</b>
The Statistical Multiplexor Definition	489
The Software Environment	490
The Real-time Constraints	490
The Input Interrupt Service Routine	491
The Output Service Interrupt Routine	495
The Communications Protocol	496
Conclusion	506
Bibliography	506
<b>Appendix C A Satire of Two Companies</b>	<b>507</b>
Rufus, Inc.	
Project Kickoff	507
Rupert Industries	
Project: ~Alpha~	507
<b>Appendix D The Source Code Is the Design</b>	<b>517</b>
What Is Software Design?	517
Afterword	523
<b>Index</b>	<b>525</b>