PRENTICE HALL

Robert C. Martin Series

Clean Architecture

A Craftsman's Guide to Software Structure and Design

Robert C. Martin

With contributions by James Grenning *and* Simon Brown

Foreword by Kevlin Henney
Afterword by Jason Gorman

Scanned with CamScanner

Clean Architecture

A Craftsman's Guide to Software Structure and Design

Robert C. Martin

PRENTICE HALL

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Scanned with CamScanner

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Visit us on the Web: informit.com

Library of Congress Control Number: 2017945537

Copyright © 2018 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearsoned.com/permissions/.

ISBN-13: 978-0-13-449416-6
ISBN-10: 0-13-449416-4

3 18

Scanned with CamScanner

This book is dedicated to my lovely wife, my four spectacular children, and their families, including my quiver full of five grandchildren—who are the dessert of my life.

Scanned with CamScanner

Contents

Foreword		xv
Preface		xix
Acknowledgments		xxiii
About the Author		xxv
PART I	Introduction	1
Chapter 1	What Is Design and Architecture?	3
	The Goal?	4
	Case Study	5
	Conclusion	12
Chapter 2	A Tale of Two Values	13
	Behavior	14
	Architecture	14
	The Greater Value	15
	Eisenhower's Matrix	16
	Fight for the Architecture	18

Scanned with CamScanner

PART II	Starting with the Bricks: Programming Paradigms	19
Chapter 3	Paradigm Overview	21
	Structured Programming	22
	Object-Oriented Programming	22
	Functional Programming	22
	Food for Thought	23
	Conclusion	24
Chapter 4	Structured Programming	25
	Proof	27
	A Harmful Proclamation	28
	Functional Decomposition	29
	No Formal Proofs	30
	Science to the Rescue	30
	Tests	31
	Conclusion	31
Chapter 5	Object-Oriented Programming	33
	Encapsulation?	34
	Inheritance?	37
	Polymorphism?	40
	Conclusion	47
Chapter 6	Functional Programming	49
	Squares of Integers	50
	Immutability and Architecture	52
	Segregation of Mutability	52
	Event Sourcing	54
	Conclusion	56
PART III	Design Principles	57
Chapter 7	SRP: The Single Responsibility Principle	61
	Symptom 1: Accidental Duplication	63
	Symptom 2: Merges	65
	Solutions	66
	Conclusion	67

Scanned with CamScanner

Chapter 8	OCP: The Open-Closed Principle	69
	A Thought Experiment	70
	Directional Control	74
	Information Hiding	74
	Conclusion	75
Chapter 9	LSP: The Liskov Substitution Principle	77
	Guiding the Use of Inheritance	78
	The Square/Rectangle Problem	79
	LSP and Architecture	80
	Example LSP Violation	80
	Conclusion	82
Chapter 10	ISP: The Interface Segregation Principle	83
	ISP and Language	85
	ISP and Architecture	86
	Conclusion	86
Chapter 11	DIP: The Dependency Inversion Principle	87
	Stable Abstractions	88
	Factories	89
	Concrete Components	91
	Conclusion	91
PART IV	Component Principles	93
Chapter 12	Components	95
	A Brief History of Components	96
	Relocatability	99
	Linkers	100
	Conclusion	102
Chapter 13	Component Cohesion	103
	The Reuse/Release Equivalence Principle	104
	The Common Closure Principle	105
	The Common Reuse Principle	107
	The Tension Diagram for Component Cohesion	108
	Conclusion	110

Scanned with CamScanner

Chapter 14	**Component Coupling**	**111**
	The Acyclic Dependencies Principle	112
	Top-Down Design	118
	The Stable Dependencies Principle	120
	The Stable Abstractions Principle	126
	Conclusion	132
PART V	**Architecture**	**133**
Chapter 15	**What Is Architecture?**	**135**
	Development	137
	Deployment	138
	Operation	138
	Maintenance	139
	Keeping Options Open	140
	Device Independence	142
	Junk Mail	144
	Physical Addressing	145
	Conclusion	146
Chapter 16	**Independence**	**147**
	Use Cases	148
	Operation	149
	Development	149
	Deployment	150
	Leaving Options Open	150
	Decoupling Layers	151
	Decoupling Use Cases	152
	Decoupling Mode	153
	Independent Develop-ability	153
	Independent Deployability	154
	Duplication	154
	Decoupling Modes (Again)	155
	Conclusion	158

Scanned with CamScanner

Chapter 17 Boundaries: Drawing Lines 159
A Couple of Sad Stories 160
FitNesse 163
Which Lines Do You Draw, and When Do You Draw Them? 165
What About Input and Output? 169
Plugin Architecture 170
The Plugin Argument 172
Conclusion 173

Chapter 18 Boundary Anatomy 175
Boundary Crossing 176
The Dreaded Monolith 176
Deployment Components 178
Threads 179
Local Processes 179
Services 180
Conclusion 181

Chapter 19 Policy and Level 183
Level 184
Conclusion 187

Chapter 20 Business Rules 189
Entities 190
Use Cases 191
Request and Response Models 193
Conclusion 194

Chapter 21 Screaming Architecture 195
The Theme of an Architecture 196
The Purpose of an Architecture 197
But What About the Web? 197
Frameworks Are Tools, Not Ways of Life 198
Testable Architectures 198
Conclusion 199

Scanned with CamScanner

Chapter 22	**The Clean Architecture**	**201**
	The Dependency Rule	203
	A Typical Scenario	207
	Conclusion	209
Chapter 23	**Presenters and Humble Objects**	**211**
	The Humble Object Pattern	212
	Presenters and Views	212
	Testing and Architecture	213
	Database Gateways	214
	Data Mappers	214
	Service Listeners	215
	Conclusion	215
Chapter 24	**Partial Boundaries**	**217**
	Skip the Last Step	218
	One-Dimensional Boundaries	219
	Facades	220
	Conclusion	220
Chapter 25	**Layers and Boundaries**	**221**
	Hunt the Wumpus	222
	Clean Architecture?	223
	Crossing the Streams	226
	Splitting the Streams	227
	Conclusion	228
Chapter 26	**The Main Component**	**231**
	The Ultimate Detail	232
	Conclusion	237
Chapter 27	**Services: Great and Small**	**239**
	Service Architecture?	240
	Service Benefits?	240
	The Kitty Problem	242
	Objects to the Rescue	244

Scanned with CamScanner

	Component-Based Services	245
	Cross-Cutting Concerns	246
	Conclusion	247
Chapter 28	The Test Boundary	249
	Tests as System Components	250
	Design for Testability	251
	The Testing API	252
	Conclusion	253
Chapter 29	Clean Embedded Architecture	255
	App-titude Test	258
	The Target-Hardware Bottleneck	261
	Conclusion	273
PART VI	Details	275
Chapter 30	The Database Is a Detail	277
	Relational Databases	278
	Why Are Database Systems So Prevalent?	279
	What If There Were No Disk?	280
	Details	281
	But What about Performance?	281
	Anecdote	281
	Conclusion	283
Chapter 31	The Web Is a Detail	285
	The Endless Pendulum	286
	The Upshot	288
	Conclusion	289
Chapter 32	Frameworks Are Details	291
	Framework Authors	292
	Asymmetric Marriage	292
	The Risks	293
	The Solution	294

Scanned with CamScanner

	I Now Pronounce You …	295
	Conclusion	295
Chapter 33	Case Study: Video Sales	297
	The Product	298
	Use Case Analysis	298
	Component Architecture	300
	Dependency Management	302
	Conclusion	302
Chapter 34	The Missing Chapter	303
	Package by Layer	304
	Package by Feature	306
	Ports and Adapters	308
	Package by Component	310
	The Devil Is in the Implementation Details	315
	Organization versus Encapsulation	316
	Other Decoupling Modes	319
	Conclusion: The Missing Advice	321
Afterword		323
PART VII	Appendix	327
Appendix A	Architecture Archaeology	329
Index		379

Scanned with CamScanner

Introduction

It doesn't take a huge amount of knowledge and skill to get a program working. Kids in high school do it all the time. Young men and women in college start billion-dollar businesses based on scrabbling together a few lines of PHP or Ruby. Hoards of junior programmers in cube farms around the world slog through massive requirements documents held in huge issue tracking systems to get their systems to "work" by the sheer brute force of *will*. The code they produce may not be pretty; but it works. It works because getting something to work—once—just isn't that hard.

Getting it right is another matter entirely. Getting software right is *hard*. It takes knowledge and skills that most young programmers haven't yet acquired. It requires thought and insight that most programmers don't take the time to develop. It requires a level of discipline and dedication that most programmers never dreamed they'd need. Mostly, it takes a passion for the craft and the desire to be a professional.

And when you get software right, something magical happens: You don't need hordes of programmers to keep it working. You don't need massive requirements documents and huge issue tracking systems. You don't need global cube farms and 24/7 programming.

Scanned with CamScanner

When software is done right, it requires a fraction of the human resources to create and maintain. Changes are simple and rapid. Defects are few and far between. Effort is minimized, and functionality and flexibility are maximized.

Yes, this vision sounds a bit utopian. But I've been there; I've seen it happen. I've worked in projects where the design and architecture of the system made it easy to write and easy to maintain. I've experienced projects that required a fraction of the anticipated human resources. I've worked on systems that had extremely low defect rates. I've seen the extraordinary effect that good software architecture can have on a system, a project, and a team. I've been to the promised land.

But don't take my word for it. Look at your own experience. Have you experienced the opposite? Have you worked on systems that are so interconnected and intricately coupled that every change, regardless of how trivial, takes weeks and involves huge risks? Have you experienced the impedance of bad code and rotten design? Has the design of the systems you've worked on had a huge negative effect on the morale of the team, the trust of the customers, and the patience of the managers? Have you seen teams, departments, and even companies that have been brought down by the rotten structure of their software? Have you been to programming hell?

I have—and to some extent, most of the rest of us have, too. It is far more common to fight your way through terrible software designs than it is to enjoy the pleasure of working with a good one.

Scanned with CamScanner

Software Development/Architecture

Clean Architecture

Practical Software Architecture Solutions from the Legendary Robert C. Martin ("Uncle Bob")

By applying universal rules of software architecture, you can dramatically improve developer productivity throughout the life of any software system. Now, building upon the success of his best-selling books *Clean Code* and *The Clean Coder*, legendary software craftsman Robert C. Martin ("Uncle Bob") reveals those rules and helps you apply them.

Martin's ***Clean Architecture*** doesn't merely present options. Drawing on over a half-century of experience in software environments of every imaginable type, Martin tells you what choices to make and why they are critical to your success. As you've come to expect from Uncle Bob, this book is packed with direct, no-nonsense solutions for the real challenges you'll face—the ones that will make or break your projects.

- Learn what software architects need to achieve—and core disciplines and practices for achieving it
- Master essential software design principles for addressing function, component separation, and data management
- See how programming paradigms impose discipline by restricting what developers can do
- Understand what's critically important and what's merely a "detail"
- Implement optimal, high-level structures for web, database, thick-client, console, and embedded applications
- Define appropriate boundaries and layers, and organize components and services
- See why designs and architectures go wrong, and how to prevent (or fix) these failures

Clean Architecture is essential reading for every current or aspiring software architect, systems analyst, system designer, and software manager—and for every programmer who must execute someone else's designs.

Robert C. Martin ("Uncle Bob") has been a programmer since 1970. An acclaimed speaker at conferences worldwide, his books include *The Clean Coder, Clean Code, Agile Software Development,* and *UML for Java Programmers.* Martin is founder of Uncle Bob Consulting, LLC, and cofounder (with his son Micah Martin) of The Clean Coders LLC. He has served as editor-in-chief of *The C++ Report*, as the first chairman of the Agile Alliance, and as co-founder and leader of Object Mentor, Inc.

Register Your Product ▸▸▸ at **informit.com/register** for convenient access to downloads, updates, and/or corrections as they become available.

informit.com/martinseries | cleancoders.com

Cover image: © Vadim Sadovski/ShutterStock

Text printed on recycled paper

ISBN-13: 978-0-13-449416-6
ISBN-10: 0-13-449416-4

53499

9 780134 494166

Pearson

$34.99 US • $43.99 CANADA

Scanned with CamScanner