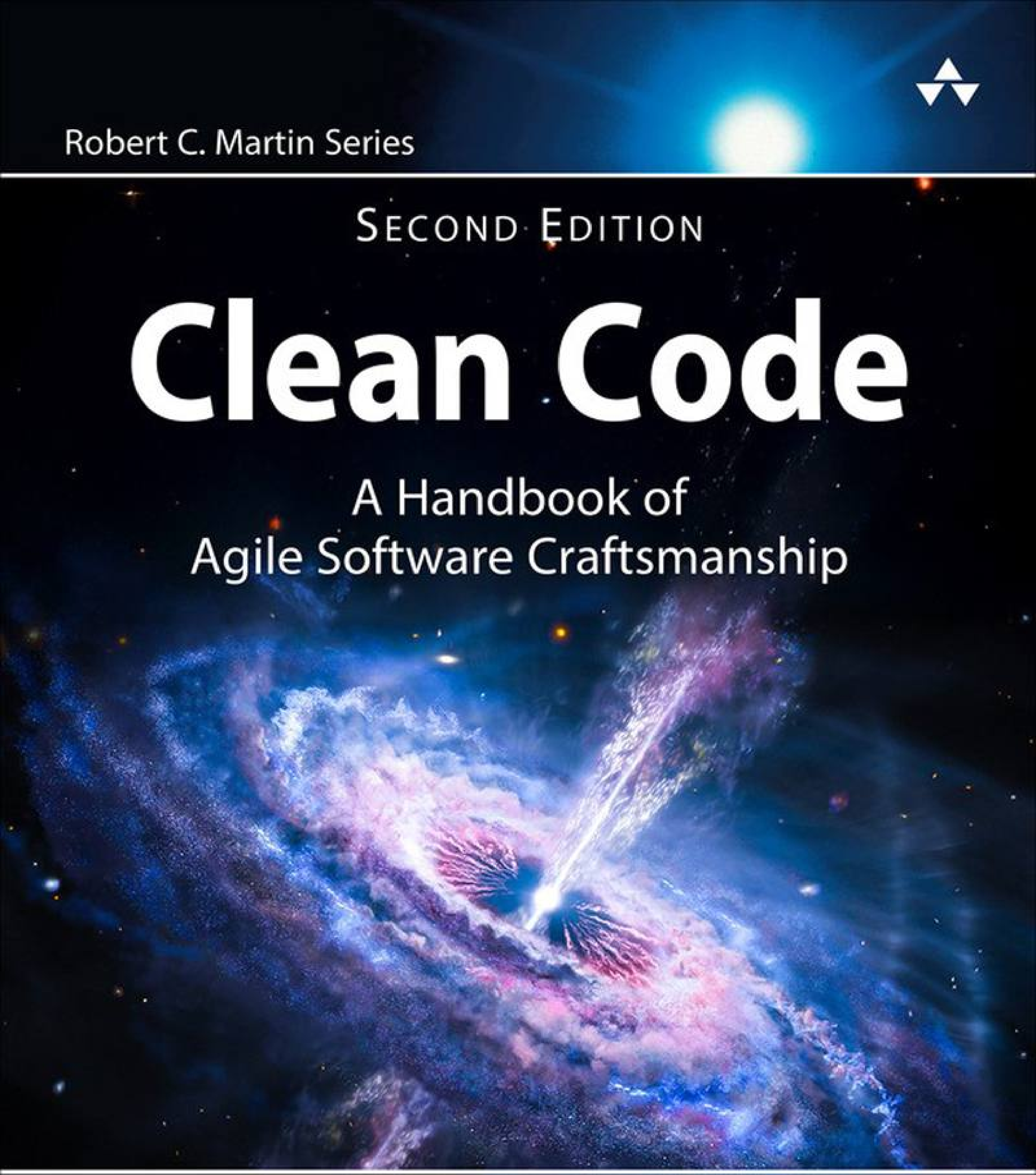
Robert C. Martin Series

SECOND EDITION

Clean Code

A Handbook of
Agile Software Craftsmanship



Robert C. Martin

Foreword by **Micah D. Martin**
Afterword by **Justin M. Martin**

Robert C. Martin Series

SECOND EDITION

Clean Code

A Handbook of
Agile Software Craftsmanship

Robert C. Martin

Foreword by Micah D. Martin

Afterword by Justin M. Martin

CleanCode:AHandbookofAgile SoftwareCraftsmanship

SecondEdition

RobertC.Martin

Contents

Foreword

Introduction

Introduction (from Long Ago)

Chapter 1: Clean Code

Part I: Code

Chapter 2: Clean That Code!

Chapter 3: First Principles

Chapter 4: Meaningful Names

Chapter 5: Comments

Chapter 6: Formatting

Chapter 7: Clean Functions

Chapter 8: Function Heuristics

Chapter 9: The Clean Method

Chapter 10: One Thing

Chapter 11: Be Polite

Chapter 12: Objects and Data Structures

Chapter 13: Clean Classes

Chapter 14: Testing Disciplines

Chapter 15: Clean Tests

Chapter 16: Acceptance Testing

Chapter 17: AIs, LLMs, and God knows What

Part II: Design

Chapter 18: Simple Design

Chapter 19: The SOLID Principles

Chapter 20: Component Principles

Chapter 21: Continuous Design

Chapter 22: Concurrency

Part III: Architecture

Chapter 23: The Two Values of Software

Chapter 24: Independence

Chapter 25: Architectural Boundaries

Chapter 26: Clean Boundaries

Chapter 27: The Clean Architecture

Part IV: Craftsmanship

Chapter 28: Harm

Chapter 29: No Defect in Behavior or Structure

Chapter 30: Repeatable Proof

Chapter 31: Small Cycles

Chapter 32: Relentless Improvement

Chapter 33: Maintain High Productivity

Chapter 34: Work as a Team

Chapter 35: Estimate Honestly and Fairly

Chapter 36: Respect for Fellow Programmers

Chapter 37: Never Stop Learning

Afterword

Appendix

Bibliography

Table of Contents

Foreword

Introduction

- A Note About Older Chapters
- The Structure of This Book

Introduction (from Long Ago)

Chapter 1. Clean Code

- There Will Be Code
- Bad Code
- The Art of Clean Code?
- Putting All This Together
- We Read More Than We Write
- The Boy Scout Rule

Part I: Code

Chapter 2. Clean That Code!

The Cleaning Process

Conclusion

Postscript: Future Bob Playing with Grok3

Postscript Conclusion

Chapter 3. First Principles

Everything Small, Well Named, Organized, and Ordered

A More Significant Example

A Final Thought

Chapter 4. Meaningful Names

Use Intention-Revealing Names

Final Words

Chapter 5. Comments

Compensating for Failure

Good Comments

Bad Comments

Conclusion

Chapter 6. Formatting

The Purpose of Formatting

Vertical Formatting

Team Rules

Uncle Bob’s Formatting Rules

Chapter 7. Clean Functions

Small!

Reading Code from Top to Bottom: The Stepdown Rule

Switch Statements

Clean Functions: A Deeper Look

Conclusion

Chapter 8. Function Heuristics

Function Arguments

Command Query Separation

Prefer Exceptions to Returning Error Codes

Structured Programming

Conclusion

Chapter 9. The Clean Method

Make It Right

Example

Conclusion

Chapter 10. One Thing

Extract Method Refactoring

What Are Large Functions Anyway?

Conclusion

Chapter 11. Be Polite

The Newspaper Metaphor

The Stepdown Rule: Once Again

The Abstraction Roller Coaster

This Is How We Write, but Not How We Want to Read

Chapter 12. Objects and Data Structures

What Is an Object?

Data Abstraction

Data/Object Antisymmetry

The Law of Demeter

Data Transfer Objects

The OO/Procedural Trade-off

But What About Performance?

Conclusion

Chapter 13. Clean Classes

Classes and Modules versus Files

What Should a Class Contain?

Chapter 14. Testing Disciplines

Discipline 1: Test-Driven Development (TDD)

Discipline 2: Test && Commit || Revert (TCR)

Discipline 3: Small Bundles

Design

Discipline

Keeping Tests Clean

Tests Enable the -ilities

Chapter 15. Clean Tests

Domain-Specific Testing Language

F.I.R.S.T.

Test Design

Conclusion

Chapter 16. Acceptance Testing

The Acceptance Testing Discipline

Conclusion

Chapter 17. AIs, LLMs, and God Knows What

Programming by Prompt

Conclusion

Part II: Design

Chapter 18. Simple Design

YAGNI

Covered by Tests

Maximize Expression

Minimize Duplication

Chapter 19. The SOLID Principles

SRP: The Single Responsibility Principle

OCP: The Open–Closed Principle

LSP: The Liskov Substitution Principle

ISP: The Interface Segregation Principle

DIP: The Dependency Inversion Principle

Chapter 20. Component Principles

Components

A Brief History of Components

Component Cohesion

Component Coupling

Conclusion

Chapter 21. Continuous Design

Continuous Change

Continuous Design

Sailing on the Four Cs of Continuous Design

When Else Do We Design?

Chapter 22. Concurrency

Why Concurrency?

Concurrency Defense Principles

2025 Update and Report from the Field

Conclusion

Part III: Architecture

Chapter 23. The Two Values of Software

Keeping Options Open

Chapter 24. Independence

Use Cases

Operation

Development

Deployment

Leaving Options Open

Chapter 25. Architectural Boundaries

What Lines Do You Draw, and When?

Plug-in Architecture

Case Study: FitNesse

Conclusion

Chapter 26. Clean Boundaries

Third-Party IoT Framework: Lots o’ Boundaries

UI/Application Boundary

Clean Boundaries

Chapter 27. The Clean Architecture

The Dependency Rule

Entities

Conclusion

Part IV: Craftmanship

Chapter 28. Harm

No Harm to Society

Harm to Function

No Harm to Structure

Soft

Tests

Chapter 29. No Defect in Behavior or Structure

Making It Right

Programmers Are Stakeholders

Do Your Best

Chapter 30. Repeatable Proof

Dijkstra

Structured Programming

Functional Decomposition

Test-Driven Development et al.

Chapter 31. Small Cycles

The History of Source Code Control

Continuous Integration

Branches versus Toggles

Continuous Deployment

Continuous Build

Chapter 32. Relentless Improvement

Test Coverage

Mutation Testing

Semantic Stability

Cleaning

Creations

Chapter 33. Maintain High Productivity

Viscosity

Managing Distractions

Time Management

Chapter 34. Work as a Team

Collaborative Programming

Open/Virtual Office

Chapter 35. Estimate Honestly and Fairly

Lies

Honesty, Accuracy, Precision

Lessons from Me

Accuracy and Precision

Aggregation

Honesty

Pressure

Chapter 36. Respect for Fellow Programmers

Chapter 37. Never Stop Learning

Afterword

Appendix. The Clean Code Debate

Introductions

Method Length

Method Length Summary

Comments

Comments Summary

John’s Rewrite of PrimeGenerator

A Tale of Two Programmers

Bob’s Rewrite of PrimeGenerator2

Test-Driven Development

TDD Summary

Closing Remarks

Bibliography

Foreword

Micah Daniel Martin

You hold in your hand the secrets for building high-quality software—secrets that will make you better than you are, secrets that will help you become a master craftsman.

Like Antonio Stradivari (perhaps the best luthier ever), Uncle Bob has spent a lifetime curating and developing the best techniques for crafting code. Unlike Stradivari, who hoarded his secrets so that only he could craft the best violins, Uncle Bob shares his secrets willingly, openly, hoping that you too will craft masterful code.

This is the Way: the Way of Clean Code. These are my disciplines, my principles, my techniques. I strive to master them all. Why? Because they make me better. And, this is the way my father taught me.

Yes, Uncle Bob is my father. He bestowed upon me the honor of writing this foreword. Furthermore, please allow me to introduce my brother, Justin, also a Clean Coder, who wrote the afterword. How appropriate that two second editions of Uncle Bob get to sandwich his second edition of *Clean Code*. (Yeah . . . that sounded cheesy as I wrote it, but I'm going with it.)

Let me tell you a little about Uncle Bob, or at least the version of him that I know as "Dad."

One of my earliest memories is riding on my dad's shoulders, playing Robot. The rules are simple. My dad was the robot, and he did whatever I told him to do . . . to a T.

Me: "Go to my room."

Dad: "I don't know how to do that."

Me: Thinking . . . "Turn."

Dad: "Which way?"

Me: "Turn right."

Dad: Starts turning clockwise. 90°. . . 180°. . . 360°. . . He doesn't stop turning.

Me: "STOP!"

Dad: Stops. Stands still, facing the absolute wrong direction.

Me: "Turn right . . . STOP!"

Dad: Turns and stops obediently. We are now facing down the hallway, the right direction.

Me: "Start walking."

Dad: "I don't know how to walk."

For the next few minutes, I teach the robot how to "walk" by lifting a leg the appropriate amount and falling forward, then repeating with the other leg. We agree to name this activity "walking."

Me: "Start walking."

Dad: Starts walking.

Me: Feeling exhausted by this ignorant robot, but also victorious that I successfully made it do my bidding, I prepare for the final maneuver, a left turn at the end of the hall. "Turn left!"

Dad: "I am still walking" . . . and CRASH we go into the wall. Apparently the robot was single threaded. We bounce off the wall and the robot continues to walk . . . CRASH we go into the wall again . . . and again.

Me: Laughing uncontrollably.

At that tender age of 4 or 5 years old, my dad was teaching me how to program with the Robot game. It was fun. Fast-forward 5 years, and my dad brings home a Commodore 64. “What is it?” I inquired. “A computer,” my dad responded. “What is it for?” I asked, still confused. “Well, with a computer you can blah blah blah blah, blah blah, you can even play games.”

“Games? Show me!” I adopted that Commodore 64, or maybe my dad gave it to me . . . regardless, it lived in my room and I played *lots* of games on it. There was a kid in my neighborhood who knew how to pirate games, and we’d do a weekly floppy disk exchange. Other than loading games from disks, I didn’t really know how to use the computer, and anytime something went wrong, like “PRESS PLAY ON TAPE” . . . I’d shout “DAD! HELP!”

At one point, he sat me down at the Commodore and introduced me to Logo, a programming language with turtle graphics. He showed me how to make the turtle turn, move forward. It was the Robot game! I played with Logo for a while and it was . . . boring. Back to gaming.

Then, I observed my dad working with Logo for what seemed like forever but was probably just a couple days. When he was finally done monopolizing my computer, he showed me his Lunar Lander game that he’d just built. It was a jaw-dropping moment as I realized this is how games are built. Wow! Lunar Lander was inducted into my game rotation, and I proudly showed it off to my friends.

A couple of years later, my dad brought home a brand-new Apple Macintosh. It was AMAZING. “What’s that thing?” “This is a mouse.” So novel. And it had more games. Better games! The Mac lived in my dad’s office (basement), but when he was at work, the Mac was mine. Similar to the Lunar Lander incident, my dad spent weeks monopolizing the Mac to create a game he called “Pharaoh.” Pharaoh was much bigger than Lunar Lander. You were the pharaoh, and your goal was to buy enough oxen, plant enough crops, and sustain enough workers every year to build a pyramid before you died. It was a fun game, and I could tell my dad had fun building it.

Programming is so much fun! That’s the conclusion I had when I reached college age. For my whole childhood, if I wasn’t having fun playing on the computers, my dad was having fun programming them. Why would I choose any other major than computer science? Programming in college was . . . not fun. Upon graduating, I was offered a job at ThoughtWorks, and, thank goodness, my dad offered me a job working with him at Object Mentor. Of course I accepted my dad’s offer, where I got to work with some of the biggest names and best software guys in the industry. Once again, programming was fun. It was at Object Mentor where I learned the way of Clean Code from my dad, building lots of internal tools and open source projects like FitNesse, that you’ll see later in this book.

Object Mentor was primarily a training and consulting business. We told people how to build good software. We didn’t really write the software for them. I’ll admit that after several years as a trainer, I got an itch to build software again, instead of teaching how to build software.

An opportunity arose, and Object Mentor started developing software for a client. It was great, and it turned out to be a solid business model. So, I convinced my dad that Object Mentor should do more software development contracts. He agreed. And then I tried to convince him that I should run that branch of the business. Oops . . . that was one step too far. His response hit me like a truck. “If you want to run things, you should start your own business.” Wow! Was that his way of putting me in my place? Or was he actually suggesting I start my own business? I concluded it was the latter, so I told him, “I quit.”

I created a company called 8th Light. It was (and still is) a contract software development company. Clients hired us to build software for them. What was unique about 8th Light was that we practiced the Way of Clean Code. We didn't call it that, because the first edition of *Clean Code* hadn't been written yet. But we put into practice everything that my dad had taught me. The Uncle Bob Way of Coding doesn't have the same ring. Moreover, 8th Light adopted apprenticeship, which is something we had dabbled with at Object Mentor. Everyone who joined 8th Light had to go through a 3-to-12-month apprenticeship where they learned the Way of Clean Code. We were ALL-IN on Clean Code. Did it work? Hell yeah! Demand for Clean Code went through the roof. We couldn't train apprentices fast enough. By the time of my departure from 8th Light in 2015, we had nearly a hundred team members, most of them "Clean Code" Craftsmen working on our clients' projects.

But maybe 8th Light was a fluke. In 2020, my dad and I decided to start Clean Coders Studio. This is a branch off our existing partnership, Clean Coders, where we sell educational software videos (https://cleancoders.com). However, "Studio" is our contract software development branch of the business, and yes, I do run it. Once again, we are ALL-IN on Clean Code, more than ever. Is it working? Hell yeah. Once again, we cannot train apprentices fast enough to meet demand.

The industry wants Clean Code. The industry NEEDS Clean Code. This book is your guide into the world of Clean Code.

I have some advice for you on how to best use this book.

1. **Don't rush it.** There is a lot of ground to cover in these pages. Apprentices at Clean Coders Studio take months to internalize all the material. Don't feel like you need to cram it all in a week.

2. **Use it as a reference.** You might want to stick some tabs on some of the pages. For example, the SOLID principles; you'll likely want to jump back to that section when you're trying to debate a code smell with your peers.

3. **Read the code examples.** When Uncle Bob walks through code changes, like the Video Store, you might be tempted to skip to the end. Don't. Follow along. Ideally, you'd have Uncle Bob sitting next to you, pairing with you on this code. His walk-throughs are the next best thing.

4. **Practice.** "Calculus is not a spectator sport." I can still hear my professor recite that when he assigned homework. Coding is not a spectator sport either. You need to practice it to internalize it. Especially TDD.

5. **Have fun!** Uncle Bob has fun when he codes. Trust me, he does. So do I. *Clean Code* should help you have more fun writing code too.

—Micah Daniel Martin

Introduction

"Code Quality is a mindset that I've seen correlate with wealth and great outcomes over and over again."

—David Heinemeier Hansson (DHH)

As I type this, it has been over a decade and a half since *Clean Code* was first published. Since then, I have participated in many discussions, podcasts, interviews, blog postings, and social network flame wars inspired by that first edition. Recently it occurred to me that there are quite a few more things I could say; and several more that I would say differently than I did in 2008.

I first proposed the *Clean Code* book in April of 2006. This was before there were iPhones. Mobile internet was mostly GPRS, or maybe EDGE. Nobody was talking about the Internet of Things. YouTube was barely one year old, and hardly anyone had heard of it—fewer took it seriously. Languages like Kotlin, Swift, Go, Dart, Rust, Elm, Elixir, and Clojure had not yet appeared. *Meet the Fockers* was the top-grossing movie up to that point.

Yeah, that was a long time ago—especially in internet time. So, it's time to take a new look at the old message of *Clean Code*. The essence of that message has not changed; but after a decade and a half our industry has—

and remarkably so. After so much change, it would be irresponsible to just dust off the old manuscript and make a few simple adjustments. So, in light of that decade and a half of progress, learning, and change, I decided to significantly restate and expand the original message.

I have added many new chapters, new ideas, and new disciplines. I have added newer languages, and newer paradigms. I have also revamped the older chapters with a decade and a half of newer insights and knowledge.

I have added chapters on design, architecture, and even ethics. These were missing from the first edition, and I felt it necessary to repair that omission. In some cases, these chapters are heavily edited and abridged excerpts from my more recent books, *Clean Architecture* and *Clean Craftsmanship*; others are entirely new.

In the original edition, I was reluctant to assert my ideas as anything more than one programmer's way of looking at things. Now, many years later, I'm a bit more confident than that. The ideas in that original edition have withstood the test of time and have quite a bit more staying power.

On the other hand, there are other very experienced programmers who have different ideas, and I thought it wise to include some of their contrary viewpoints. After all, I'm not the only one who has ideas about what it means for code to be clean. I like my ideas best, but I think all creditable ideas should be heard and considered.

On yet another hand, in the process of reviewing the newer chapters months after they had been written, I was surprised to find that I had issues with some of the examples I had posed. Rather than erasing that surprise by further "cleaning" the examples, I added some Future Bob comments so that you could see my own reaction.

Why is all this cleanliness important? What benefits do these ideas about clean code bring? Why should we be worried about code cleanliness at all?

Our civilization depends upon us. Nothing happens in our society without computers being intimately involved in virtually every aspect of our lives. Nowadays we cannot microwave a hot dog, watch TV, make a phone call, drive to the store, wash our dishes or our clothes, buy anything, sell anything, or even tell the time of day without software being smack in the middle of it all.

You and I, we programmers, rule the world. Other people think they rule the world, but then they hand those rules to us, and *we* write the rules—the software—that executes in the machines that sit in the middle of the details of billions of people's lives.

And some of the software we wrote has caused two fully loaded passenger jets to slam into the ground at near the speed of sound, cars to accelerate out of control and crash, billions of dollars to be lost in seconds, rockets to explode, spacecraft to crash into Mars, and noxious fumes to be emitted by tens of thousands of cars. Just for starters.

And with such a burden of responsibility and failure sitting upon us, we have to ask ourselves this question: Are we professionals?

A professional is someone who professes a set of standards, disciplines, and ethics. Do we? Do you? If so, can you recite them?

My goal for this book is to provide some standards of code cleanliness, some disciplines of code creation and maintenance, and the beginnings of an ethical framework to help us define a true profession. Because if we don't define and adopt that profession, others who are less able but more powerful

—the politicians of the world—will define it for us and force it down our throats.

I hope there's still time for us to get there before that idea occurs to them.

A Note About Older Chapters

I have taken a great deal of liberty with the older chapters for which I was the sole author. In many cases they have been completely rewritten, or very substantially edited.

Most of the authors of the chapters that I did not write have submitted second editions of their chapters to this effort.

The Structure of This Book

Possibly the greatest change between the first and second editions is the structure of the material. I have divided the book into four parts: Code, Design, Architecture, and Craftsmanship.

These parts are not independent of each other.

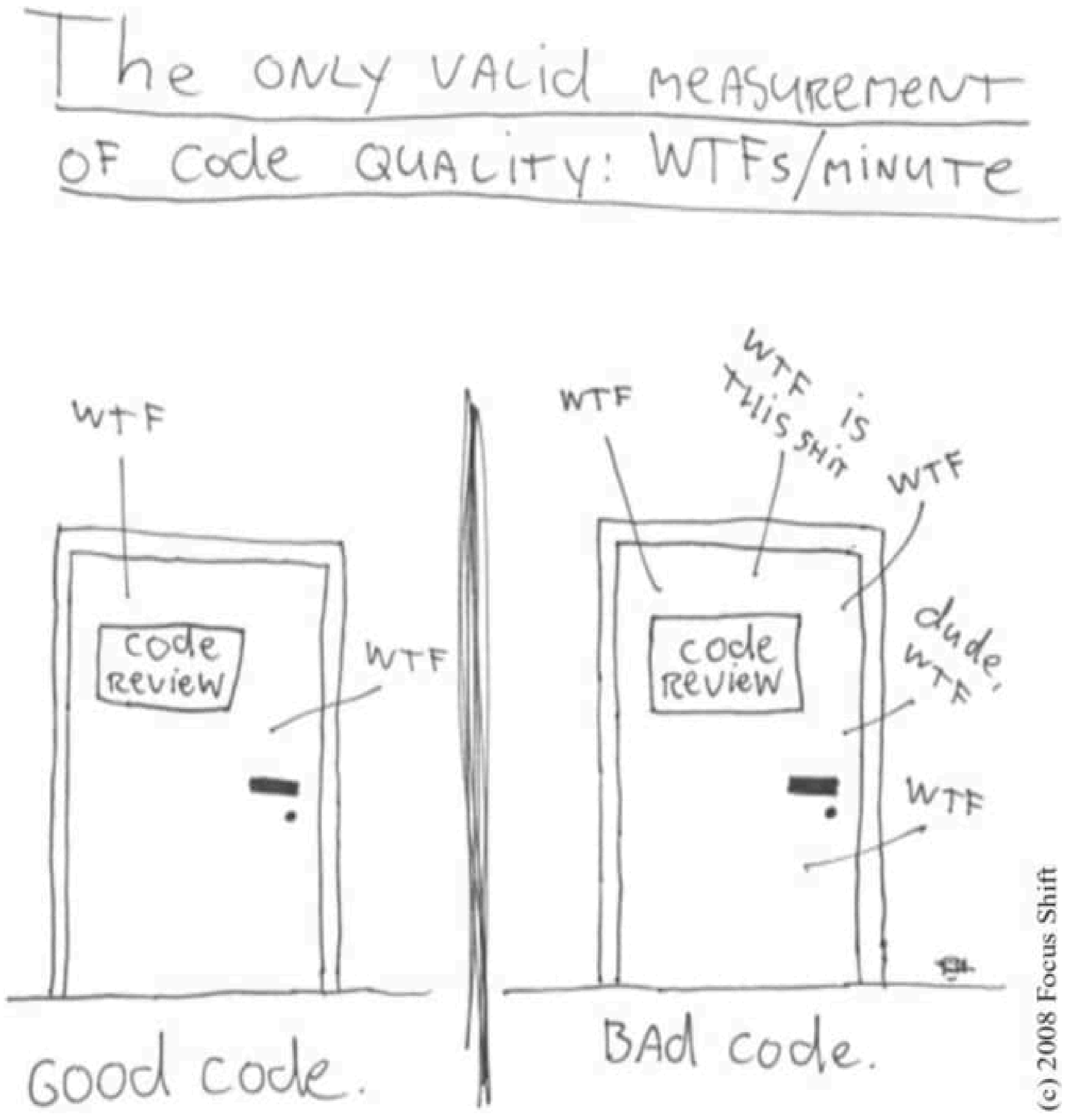
Part I: Code strongly depends upon Part II: Design. As you read Part I, you will find many references to design principles and design strategies that are described in Part II. I encourage you to treat Part II as a reference that you can consult while reading Part I. Then, when you are done with Part I, you should read Part II in its entirety.

Part III: Architecture is a set of deeply edited and abridged excerpts from my book *Clean Architecture*. Again, consider it a reference to assist in your understanding of Part I. However, it also stands alone as a simple introduction to higher-level architectural principles.

Part IV: Craftsmanship is, once again, a set of deeply edited and abridged excerpts from my book *Clean Craftsmanship*—specifically the chapters that have to do with the ethics of software development. It is included as a way to underscore that there is an ethical imperative to writing clean code.

Introduction (from Long Ago)

(This is the Introduction to the first edition. Edited and abridged.)



Reproduced with the kind permission of Thom Holwerda (www.osnews.com/story/19266/WTFs_m).

Look at the drawing above. Which door represents your code? Which door represents your team or your company? Why are we in that room? Is this just a normal code review or have we found a stream of horrible problems shortly after going live?

Are we debugging in a panic, poring over code that we thought worked? Are customers leaving in droves and managers breathing down our necks? How can we make sure we wind up behind the right door when the going gets tough? The answer is: craftsmanship.

There are two parts to learning craftsmanship: knowledge and work. You must gain the knowledge of principles, patterns, practices, and heuristics that a craftsman knows, and you must also grind that knowledge into your fingers, eyes, and gut by working hard and practicing.

I can teach you the physics of riding a bicycle. Indeed, the classical mathematics is relatively straightforward. Gravity, friction, angular momentum, center of mass, and so forth can be demonstrated with less than a page full of equations. Given those formulae, I could prove to you that bicycle riding is practical and give you all the knowledge you needed to make it work.

And you'd still fall down the first time you climbed on that bike.

Coding is no different. We could write down all the "feel good" principles of clean code and then trust you to do the work (in other words, let you fall down when you get on the bike), but then what kind of teachers would that make us, and what kind of student would that make you?

No. That's not the way this book is going to work.

Learning to write clean code is hard work. It requires more than just the knowledge of principles and patterns. You must *sweat* over it. You must practice it yourself, and watch yourself fail. You must watch others practice it and fail. You must see them stumble and retrace their steps. You must see them agonize over decisions and see the price they pay for making those decisions the wrong way.

Be prepared to work hard while reading this book. This is not a “feel good” book that you can read on an airplane and finish before you land. This book will make you work, *and work hard*. What kind of work will you be doing? You’ll be reading code—lots of code. And you will be challenged to think about what’s right about that code and what’s wrong with it. You’ll be asked to follow along as we (the authors) take modules apart and put them back together again. This will take time and effort; but we think it will be worth it.

Chapter 1. Clean Code

"'Clean' isn't a ding at anyone. Our notion of 'unclean' is a way of characterizing code that demands more effort from the average developer to understand or maintain."

—Jeff Langr