

20<sup>th</sup> ANNIVERSARY EDITION



# The Pragmatic Programmer

  
your journey to mastery

DAVID THOMAS

ANDREW HUNT





---

# The Pragmatic Programmer

---

**YOUR JOURNEY TO MASTERY  
BY DAVE THOMAS, ANDY HUNT**

Version: P1.0 (September 13, 2019)

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 Pragmatic Programmer" and the linking g device are trademarks of The Pragmatic Programmers, LLC.

The authors 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](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact [governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com). For questions about sales outside the U.S., please contact [intlcs@pearson.com](mailto:intlcs@pearson.com). Visit us on the Web:

[informit.com/aw](http://informit.com/aw)

Library of Congress Control Number: 2019944178

Copyright © 2020 Pearson Education, Inc.

Cover images: Mihalec/Shutterstock,  
Stockish/Shutterstock

All rights reserved. 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](http://www.pearsoned.com/permissions).

ISBN-13: 978-0-13-595705-9

ISBN-10: 0-13-595705-2

*For Juliet and Ellie,  
Zachary and Elizabeth,  
Henry and Stuart*

---

# Table of Contents

---

1. **Foreword**

---
2. **Preface to the Second Edition**

---

  1. **How the Book Is Organized**

---
  2. **What's in a Name?**

---
  3. **Source Code and Other Resources**

---
  4. **Send Us Feedback**

---
  5. **Second Edition Acknowledgments**

---
3. **From the Preface to the First Edition**

---

  1. **Who Should Read This Book?**

---
  2. **What Makes a Pragmatic Programmer?**

---
  3. **Individual Pragmatists, Large Teams**

---
  4. **It's a Continuous Process**

---
4. **1. A Pragmatic Philosophy**

---

  1. **Topic 1. It's Your Life**

---
  2. **Topic 2. The Cat Ate My Source Code**

---
  3. **Topic 3. Software Entropy**

---
  4. **Topic 4. Stone Soup and Boiled Frogs**

---

5. Topic 5. Good-Enough Software
6. Topic 6. Your Knowledge Portfolio
7. Topic 7. Communicate!

## 5. 2. A Pragmatic Approach

1. Topic 8. The Essence of Good Design
2. Topic 9. DRY—The Evils of Duplication
3. Topic 10. Orthogonality
4. Topic 11. Reversibility
5. Topic 12. Tracer Bullets
6. Topic 13. Prototypes and Post-it Notes
7. Topic 14. Domain Languages
8. Topic 15. Estimating

## 6. 3. The Basic Tools

1. Topic 16. The Power of Plain Text
2. Topic 17. Shell Games
3. Topic 18. Power Editing
4. Topic 19. Version Control
5. Topic 20. Debugging
6. Topic 21. Text Manipulation
7. Topic 22. Engineering Daybooks

## 7. 4. Pragmatic Paranoia

1. Topic 23. Design by Contract
2. Topic 24. Dead Programs Tell No Lies

3. Topic 25. Assertive Programming
4. Topic 26. How to Balance Resources
5. Topic 27. Don't Outrun Your Headlights

## 8. **5. Bend, or Break**

1. Topic 28. Decoupling
2. Topic 29. Juggling the Real World
3. Topic 30. Transforming Programming
4. Topic 31. Inheritance Tax
5. Topic 32. Configuration

## 9. **6. Concurrency**

1. Topic 33. Breaking Temporal Coupling
2. Topic 34. Shared State Is Incorrect State
3. Topic 35. Actors and Processes
4. Topic 36. Blackboards

## 0. **7. While You Are Coding**

1. Topic 37. Listen to Your Lizard Brain
2. Topic 38. Programming by Coincidence
3. Topic 39. Algorithm Speed
4. Topic 40. Refactoring
5. Topic 41. Test to Code
6. Topic 42. Property-Based Testing
7. Topic 43. Stay Safe Out There
8. Topic 44. Naming Things



## 1. **8. Before the Project**

1. **Topic 45. The Requirements Pit**
2. **Topic 46. Solving Impossible Puzzles**
3. **Topic 47. Working Together**
4. **Topic 48. The Essence of Agility**

## 2. **9. Pragmatic Projects**

1. **Topic 49. Pragmatic Teams**
2. **Topic 50. Coconuts Don't Cut It**
3. **Topic 51. Pragmatic Starter Kit**
4. **Topic 52. Delight Your Users**
5. **Topic 53. Pride and Prejudice**

## 3. **10. Postface**

## 4. **A1. Bibliography**

## 5. **A2. Possible Answers to the Exercises**

# Praise for the second edition of *The Pragmatic Programmer*

Some say that with *The Pragmatic Programmer*, Andy and Dave captured lightning in a bottle; that it's unlikely anyone will soon write a book that can move an entire industry as it did. Sometimes, though, lightning does strike twice, and this book is proof. The updated content ensures that it will stay at the top of "best books in software development" lists for another 20 years, right where it belongs.

— VM (Vicky) Brasseur

*Director of Open Source Strategy, Juniper Networks*

If you want your software to be easy to modernize and maintain, keep a copy of *The Pragmatic Programmer* close. It's filled with practical advice, both technical and professional, that will serve you and your projects well for years to come.

— Andrea Goulet

*CEO, Corgibytes; Founder, LegacyCode.Rocks*

*The Pragmatic Programmer* is the one book I can point to that completely dislodged the existing trajectory of my career in software and pointed me in the direction of success. Reading it opened my mind to the possibilities of being a craftsman, not just a cog in a big machine. One of the most significant books in my life.

— Obie Fernandez

*Author, The Rails Way*

First-time readers can look forward to an enthralling induction into the modern world of software practice, a world that the first edition played a major role in shaping. Readers of the first edition will rediscover here the insights and practical wisdom that made the book so significant in the first place, expertly curated and updated, along with much that's new.

— David A. Black

*Author, The Well-Founded Rubyist*

I have an old paper copy of the original *Pragmatic Programmer* on my bookshelf. It has been read and re-read and a long time ago it changed everything about how I approached my job as a programmer. In the new edition everything and nothing has changed: I now read it on my iPad and the code examples use modern programming

languages—but the underlying concepts, ideas, and attitudes are timeless and universally applicable. Twenty years later, the book is as relevant as ever. It makes me happy to know that current and future developers will have the same opportunity to learn from Andy and Dave’s profound insights as I did back in the day.

— Sandy Mamoli

*Agile coach, author of [How Self-Selection Lets People Excel](#)*

Twenty years ago, the first edition of *The Pragmatic Programmer* completely changed the trajectory of my career. This new edition could do the same for yours.

— Mike Cohn

*Author of [Succeeding with Agile](#),  
[Agile Estimating and Planning](#), and  
[User Stories Applied](#)*

# Foreword

I remember when Dave and Andy first tweeted about the new edition of this book. It was big news. I watched as the coding community responded with excitement. My feed buzzed with anticipation. After twenty years, *The Pragmatic Programmer* is just as relevant today as it was back then.

It says a lot that a book with such history had such a reaction. I had the privilege of reading an unreleased copy to write this foreword, and I understood why it created such a stir. While it's a technical book, calling it that does it a disservice. Technical books often intimidate. They're stuffed with big words, obscure terms, convoluted examples that, unintentionally, make you feel stupid. The more experienced the author, the easier it is to forget what it's like to learn new concepts, to be a beginner.

Despite their decades of programming experience, Dave and Andy have conquered the difficult challenge of writing with the same excitement of people who've just learned these lessons. They don't talk down to you. They don't assume you are an expert. They don't even assume you've read the first edition. They take you as you are—programmers who just want to be better. They spend the pages of this book helping you get there, one actionable step at a time.



To be fair, they'd already done this before. The original release was full of tangible examples, new ideas, and practical tips to build your coding muscles and develop your coding brain that still apply today. But this updated edition makes two improvements on the book.

The first is the obvious one: it removes some of the older references, the out-of-date examples, and replaces them with fresh, modern content. You won't find examples of loop invariants or build machines. Dave and Andy have taken their powerful content and made sure the lessons still come through, free of the distractions of old examples. It dusts off old ideas like DRY (don't repeat yourself) and gives them a fresh coat of paint, really making them shine.

But the second is what makes this release truly exciting. After writing the first edition, they had the chance to reflect on what they were trying to say, what they wanted their readers to take away, and how it was being received. They got feedback on those lessons. They saw what stuck, what needed refining, what was misunderstood. In the twenty years that this book has made its way through the hands and hearts of programmers all over the world, Dave and Andy have studied this response and formulated new ideas, new concepts.

They've learned the importance of agency and recognized that developers have arguably more agency than most other professionals. They start this book with the simple but profound message: "it's your life." It reminds us of our own power in our code base, in our jobs, in our careers. It sets the tone for everything else in the book—that it's more than just another technical book filled with code examples.

What makes it truly stand out among the shelves of technical books is that it understands what it means to be a programmer. Programming is about trying to make the future less painful. It's about making things easier for our teammates. It's about getting things wrong and being able to bounce back. It's about forming good habits. It's about understanding your toolset. Coding is just part of the world of being a programmer, and this book explores that world.

I spend a lot of time thinking about the coding journey. I didn't grow up coding; I didn't study it in college. I didn't spend my teenage years tinkering with tech. I entered the coding world in my mid-twenties and had to learn what it meant to be a programmer. This community is very different from others I'd been a part of. There is a unique dedication to learning and practicality that is both refreshing and intimidating.

For me, it really does feel like entering a new world. A new town, at least. I had to get to know the neighbors, pick my grocery store, find the best coffee shops. It took a while to get the lay of the land, to find the most efficient routes, to avoid the streets with the heaviest traffic, to know when traffic was likely to hit. The weather is different, I needed a new wardrobe.

The first few weeks, even months, in a new town can be scary. Wouldn't it be wonderful to have a friendly, knowledgeable neighbor who'd been living there a while? Who can give you a tour, show you those coffee shops? Someone who'd been there long enough to know the culture, understand the pulse of the town, so you not only feel at home, but become a contributing member as well? Dave and Andy are those neighbors.

As a relative newcomer, it's easy to be overwhelmed not by the

act of programming but the process of becoming a programmer. There is an entire mindset shift that needs to happen—a change in habits, behaviors, and expectations. The process of becoming a better programmer doesn't just happen because you know how to code; it must be met with intention and deliberate practice. This book is a guide to becoming a better programmer efficiently.

But make no mistake—it doesn't tell you how programming should be. It's not philosophical or judgmental in that way. It tells you, plain and simple, what a Pragmatic Programmer is—how they operate, and how they approach code. They leave it up to you to decide if you want to be one. If you feel it's not for you, they won't hold it against you. But if you decide it is, they're your friendly neighbors, there to show you the way.

► *Saron Yitbarek*

Founder & CEO of CodeNewbie

Host of Command Line Heroes

# Preface to the Second Edition

Back in the 1990s, we worked with companies whose projects were having problems. We found ourselves saying the same things to each: maybe you should test that before you ship it; why does the code only build on Mary's machine? Why didn't anyone ask the users?

To save time with new clients, we started jotting down notes. And those notes became *The Pragmatic Programmer*. To our surprise the book seemed to strike a chord, and it has continued to be popular these last 20 years.

But 20 years is many lifetimes in terms of software. Take a developer from 1999 and drop them into a team today, and they'd struggle in this strange new world. But the world of the 1990s is equally foreign to today's developer. The book's references to things such as CORBA, CASE tools, and indexed loops were at best quaint and more likely confusing.

At the same time, 20 years has had no impact whatsoever on common sense. Technology may have changed, but people haven't. Practices and approaches that were a good idea then remain a good idea now. Those aspects of the book aged well.

So when it came time to create this *20<sup>th</sup> Anniversary Edition*,

we had to make a decision. We could go through and update the technologies we reference and call it a day. Or we could reexamine the assumptions behind the practices we recommended in the light of an additional two decades' worth of experience.

In the end, we did both.

As a result, this book is something of a *Ship of Theseus*.<sup>[1]</sup> Roughly one-third of the topics in the book are brand new. Of the rest, the majority have been rewritten, either partially or totally. Our intent was to make things clearer, more relevant, and hopefully somewhat timeless.

We made some difficult decisions. We dropped the *Resources* appendix, both because it would be impossible to keep up-to-date and because it's easier to search for what you want. We reorganized and rewrote topics to do with concurrency, given the current abundance of parallel hardware and the dearth of good ways of dealing with it. We added content to reflect changing attitudes and environments, from the agile movement which we helped launch, to the rising acceptance of functional programming idioms and the growing need to consider privacy and security.

Interestingly, though, there was considerably less debate between us on the content of this edition than there was when we wrote the first. We both felt that the stuff that was important was easier to identify.

Anyway, this book is the result. Please enjoy it. Maybe adopt some new practices. Maybe decide that some of the stuff we suggest is wrong. Get involved in your craft. Give us feedback.



But, most important, remember to make it fun.

## How the Book Is Organized

This book is written as a collection of short topics. Each topic is self-contained, and addresses a particular theme. You'll find numerous cross references, which help put each topic in context. Feel free to read the topics in any order—this isn't a book you need to read front-to-back.

Occasionally you'll come across a box labeled *Tip nn* (such as Tip 1, *Care About Your Craft*). As well as emphasizing points in the text, we feel the tips have a life of their own—we live by them daily. You'll find a summary of all the tips on a pull-out card inside the back cover.

We've included exercises and challenges where appropriate. Exercises normally have relatively straightforward answers, while the challenges are more open-ended. To give you an idea of our thinking, we've included our answers to the exercises in an appendix, but very few have a single *correct* solution. The challenges might form the basis of group discussions or essay work in advanced programming courses.

There's also a short bibliography listing the books and articles we explicitly reference.

## What's in a Name?

Scattered throughout the book you'll find various bits of jargon —either perfectly good English words that have been corrupted to mean something technical, or horrendous made-up words that have been assigned meanings by computer scientists with a grudge against the language. The first time we use each of these jargon words, we try to define it, or at least give a hint to its meaning. However, we're sure that some have fallen through the cracks, and others, such as *object* and *relational database*, are in common enough usage that adding a definition would be boring. If you *do* come across a term you haven't seen before, please don't just skip over it. Take time to look it up, perhaps on the web, or maybe in a computer science textbook. And, if you get a chance, drop us an email and complain, so we can add a definition to the next edition.

Having said all this, we decided to get revenge against the computer scientists. Sometimes, there are perfectly good jargon words for concepts, words that we've decided to ignore. Why? Because the existing jargon is normally restricted to a particular problem domain, or to a particular phase of development. However, one of the basic philosophies of this book is that most of the techniques we're recommending are universal: modularity applies to code, designs, documentation, and team organization, for instance. When we wanted to use the conventional jargon word in a broader context, it got confusing —we couldn't seem to overcome the baggage the original term brought with it. When this happened, we contributed to the decline of the language by inventing our own terms.

## Source Code and Other Resources

Most of the code shown in this book is extracted from compilable source files, available for download from our website.<sup>[2]</sup>

There you'll also find links to resources we find useful, along with updates to the book and news of other Pragmatic Programmer developments.

## **Send Us Feedback**

We'd appreciate hearing from you. Email us at [ppbook@pragprog.com](mailto:ppbook@pragprog.com).



## Second Edition Acknowledgments

We have enjoyed literally thousands of interesting conversations about programming over the last 20 years, meeting people at conferences, at courses, and sometimes even on the plane. Each one of these has added to our understanding of the development process, and has contributed to the updates in this edition. Thank you all (and keep telling us when we're wrong).

Thanks to the participants in the book's beta process. Your questions and comments helped us explain things better.

Before we went beta, we shared the book with a few folks for comments. Thanks to VM (Vicky) Brasseur, Jeff Langr, and Kim Shrier for your detailed comments, and to José Valim and Nick Cuthbert for your technical reviews.

Thanks to Ron Jeffries for letting us use the Sudoku example.

Much gratitude to the folks at Pearson who agreed to let us create this book our way.

A special thanks to the indispensable Janet Furlow, who masters whatever she takes on and keeps us in line.

And, finally, a shout out to all the Pragmatic Programmers out there who have been making programming better for everyone for the last twenty years. Here's to twenty more.

[1] If, over the years, every component of a ship is replaced as it fails, is the resulting vessel the same ship?

[2] <https://pragprog.com/titles/tpp20>

# From the Preface to the First Edition

This book will help you become a better programmer.

You could be a lone developer, a member of a large project team, or a consultant working with many clients at once. It doesn't matter; this book will help you, as an individual, to do better work. This book isn't theoretical—we concentrate on practical topics, on using your experience to make more informed decisions. The word *pragmatic* comes from the Latin *pragmaticus*—"skilled in business"—which in turn is derived from the Greek πραγματικός, meaning "fit for use."

This is a book about doing.

Programming is a craft. At its simplest, it comes down to getting a computer to do what you want it to do (or what your user wants it to do). As a programmer, you are part listener, part advisor, part interpreter, and part dictator. You try to capture elusive requirements and find a way of expressing them so that a mere machine can do them justice. You try to document your work so that others can understand it, and you try to engineer your work so that others can build on it. What's more, you try to do all this against the relentless ticking of the project clock. You

work small miracles every day.

It's a difficult job.

There are many people offering you help. Tool vendors tout the miracles their products perform. Methodology gurus promise that their techniques guarantee results. Everyone claims that their programming language is the best, and every operating system is the answer to all conceivable ills.

Of course, none of this is true. There are no easy answers. There is no *best* solution, be it a tool, a language, or an operating system. There can only be systems that are more appropriate in a particular set of circumstances.

This is where pragmatism comes in. You shouldn't be wedded to any particular technology, but have a broad enough background and experience base to allow you to choose good solutions in particular situations. Your background stems from an understanding of the basic principles of computer science, and your experience comes from a wide range of practical projects. Theory and practice combine to make you strong.

You adjust your approach to suit the current circumstances and environment. You judge the relative importance of all the factors affecting a project and use your experience to produce appropriate solutions. And you do this continuously as the work progresses. Pragmatic Programmers get the job done, and do it well.

## Who Should Read This Book?

This book is aimed at people who want to become more effective and more productive programmers. Perhaps you feel frustrated that you don't seem to be achieving your potential. Perhaps you look at colleagues who seem to be using tools to make themselves more productive than you. Maybe your current job uses older technologies, and you want to know how newer ideas can be applied to what you do.

We don't pretend to have all (or even most) of the answers, nor are all of our ideas applicable in all situations. All we can say is that if you follow our approach, you'll gain experience rapidly, your productivity will increase, and you'll have a better understanding of the entire development process. And you'll write better software.



## What Makes a Pragmatic Programmer?

Each developer is unique, with individual strengths and weaknesses, preferences and dislikes. Over time, each will craft their own personal environment. That environment will reflect the programmer's individuality just as forcefully as his or her hobbies, clothing, or haircut. However, if you're a Pragmatic Programmer, you'll share many of the following characteristics:

### *Early adopter/fast adapter*

You have an instinct for technologies and techniques, and you love trying things out. When given something new, you can grasp it quickly and integrate it with the rest of your knowledge. Your confidence is born of experience.

### *Inquisitive*

You tend to ask questions. *That's neat—how did you do that? Did you have problems with that library? What's this quantum computing I've heard about? How are symbolic links implemented?* You are a pack rat for little facts, each of which may affect some decision years from now.

### *Critical thinker*

You rarely take things as given without first getting the facts. When colleagues say "because that's the way it's done," or a vendor promises the solution to all your problems, you smell a challenge.

### *Realistic*

You try to understand the underlying nature of each problem you face. This realism gives you a good feel for

how difficult things are, and how long things will take. Deeply understanding that a process *should* be difficult or *will* take a while to complete gives you the stamina to keep at it.

*Jack of all trades*

You try hard to be familiar with a broad range of technologies and environments, and you work to keep abreast of new developments. Although your current job may require you to be a specialist, you will always be able to move on to new areas and new challenges.

We've left the most basic characteristics until last. All Pragmatic Programmers share them. They're basic enough to state as tips:

Tip 1

### Care About Your Craft

We feel that there is no point in developing software unless you care about doing it well.

Tip 2

### Think! About Your Work

In order to be a Pragmatic Programmer, we're challenging you to think about what you're doing while you're doing it. This isn't a one-time audit of current practices—it's an ongoing critical appraisal of every decision you make, every day, and on every project. Never run on auto-pilot. Constantly be thinking, critiquing your work in real time. The old IBM corporate motto, *THINK!*, is the Pragmatic Programmer's mantra.

If this sounds like hard work to you, then you're exhibiting the *realistic* characteristic. This is going to take up some of your