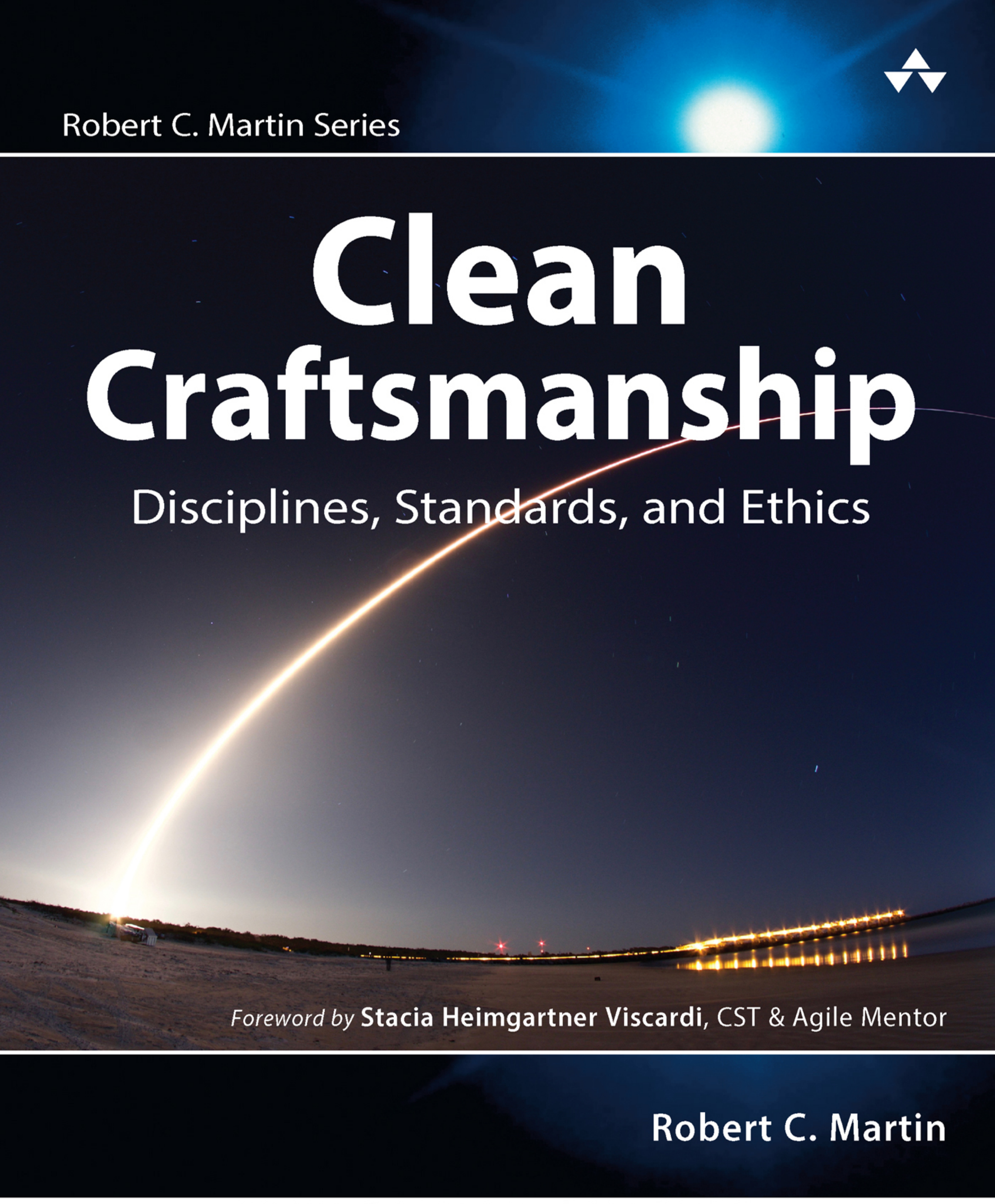
Robert C. Martin Series

# Clean Craftsmanship

Disciplines, Standards, and Ethics

*Foreword by* **Stacia Heimgartner Viscardi**, CST & Agile Mentor

**Robert C. Martin**



# *Praise for* Clean Craftsmanship

"Bob's *Clean Craftsmanship* has done a great job explaining the purposes of agile technical practices, along with a deep historical basis for how they came into existence, as well as positioning for why they will always be important. His involvement in history and formation of agility, thorough understanding of practices, and their purposes reflect vividly throughout the manuscript."

*—Tim Ottinger, well-known Agile Coach and author*

"Bob's writing style is excellent. It is easy to read and the concepts are explained in perfect detail for even a new programmer to follow. Bob even has some funny moments, which pleasantly snap you out of focus. The true value of the book is really in the cry for change, for something better . . . the cry for programmers to be professional . . . the realization that software is everywhere. Additionally, I believe there is a lot of value in all the history Bob provides. I enjoy that he doesn't waste time laying blame for how we got to where we are now. Bob calls people to action, asking them to take responsibility by increasing their standards and level of professionalism, even if that means pushing back sometimes."

*—Heather Kanser*

"As software developers, we have to continually solve important problems for our employers, customers, colleagues, and future selves. Getting the app to work, though difficult, is not enough, it does not make you a craftsman. With an app working, you have passed the *app-titude* test. You may have the aptitude to be a craftsman, but there is more to master. In these pages, Bob expresses clearly the techniques and responsibilities to go beyond the *app-titude* test and shows the way of the serious software craftsman."

*—James Grenning, author of* Test-Driven Development for Embedded C *and Agile Manifesto co-author*

"Bob's one of the very few famous developers with whom I'd like to work on a tech project. It's not because he's a good developer, famous, or a good communicator; it's because Bob helps me be a better developer and a team member. He has spotted every major development trend, years ahead of others, and has been able to explain its importance, which encouraged me to learn. Back when I started—apart from being honest and a good person—the idea of craftsmanship and ethics was completely missing from this field. Now, it seems to be the most important thing professional developers can learn, even ahead of coding itself. I'm happy to see Bob leading the way again. I can't wait to hear his perspective and incorporate it into my own practice."

*—Daniel Markham, Principal, Bedford Technology Group, Inc.*

*This page intentionally left blank*

# Clean Craftsmanship

# Clean Craftsmanship

## Disciplines, Standards, and Ethics

Robert C. Martin

Addison-Wesley

Boston • Columbus • 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

Cover image: Tom Cross/Shutterstock

Page xxix: Author photo courtesy of Robert C. Martin

Page 6: “. . . we shall need . . . what we are doing.” A.M. Turing’s ACE Report of 1946 and Other Papers – Vol. 10, “In the Charles Babbage Institute Reprint Series for the History of Computing”, (B.E. Carpenter, B.W. Doran, eds.). The MIT Press, 1986.

Page 62: Illustration by Angela Brooks

Page 227: “You aren’t gonna need it.” Jeffries, Ron, Ann Anderson, and Chet Hendrickson. Extreme Programming Installed. Addison-Wesley, 2001.

Page 280: “We shall need . . . work of this kind to be done.” A.M. Turing’s ACE Report of 1946 and Other Papers – Vol. 10, “In the Charles Babbage Institute Reprint Series for the History of Computing”, (B.E. Carpenter, B.W. Doran, eds.). The MIT Press, 1986.

Page 281: “One of our difficulties . . . what we are doing.” A.M. Turing’s ACE Report of 1946 and Other Papers – Vol. 10, “In the Charles Babbage Institute Reprint Series for the History of Computing”, (B.E. Carpenter, B.W. Doran, eds.). The MIT Press, 1986.

Page 289: “This was a couple . . . for whatever reasons.” Volkswagen North America CEO Michael Horn prior to testifying before the House Energy and Commerce Committee in Washington, October 8, 2015.

Page 310: “I have two . . . are never urgent.” In a 1954 speech to the Second Assembly of the World Council of Churches, former U.S. President Dwight D. Eisenhower, who was quoting Dr J. Roscoe Miller, president of Northwestern University.

Page 310: Dwight D. Eisenhower, President of the United States, photo, February 1959. Niday Picture Library/Alamy Stock Photo.

Page 319: “Of course I . . . any size at all.” Edsger W. Dijkstra: Notes on Structured Programming in Pursuit of Simplicity; the manuscripts of Edsger W. Dijkstra, Ed. Texas; 1969-1970; http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF

Page 328: Photo courtesy of Robert C. Martin

Page 331: Photo courtesy of Robert C. Martin

Library of Congress Control Number: 2021942499

ISBN-13: 978-0-13-691571-3
ISBN-10: 0-13-691571-X

ScoutAutomatedPrintCode

*In memory of Mike Beedle*

*This page intentionally left blank*

# Contents

*This page intentionally left blank*

# Foreword

I remember meeting Uncle Bob in the spring of 2003, soon after Scrum was introduced to our company and technology teams. As a skeptical, fledgling ScrumMaster, I remember listening to Bob teach us about TDD and a little tool called FitNesse, and I remember thinking to myself, "Why would we *ever* write test cases that fail first? Doesn't testing come *after* coding?" I often walked away scratching my head, as did many of my team members, and yet to this day I distinctly remember Bob's palpable exuberance for code craftsmanship like it was only yesterday. I recall his directness one day as he was looking at our bug backlog and asking us why on earth we would make such poor decisions about software systems that we did not in fact own—"These systems are *company assets*, not your own *personal assets*." His passion piqued our curiosity, and a year and half later, we had refactored our way to about 80 percent automated test coverage and a clean code base that made pivoting much easier, resulting in much happier customers—and happier teams. We moved lightning fast after that, wielding our definition of *done* like armor to protect us from the always-lurking code goblins; we had learned, in essence, how to protect us from ourselves. Over time, we developed a warmth for Uncle Bob, who came to truly feel like an uncle to us—a warm, determined, and courageous man who would over time help us learn to stand up for ourselves and do what was right. While some kids' Uncle

Bobs taught them how to ride bicycles or fish, our Uncle Bob taught us to not compromise our integrity—and to this day, the ability and desire to show up to every situation with courage and curiosity has been the best lesson of my career.

I brought Bob's early lessons with me on my journey as I ventured into the world as an agile coach and quickly observed for myself that the best product development teams figured out how to package up their own best practices for their unique contexts, for their particular customers, in their respective industries. I remembered Bob's lessons when I observed that the best development tools in the world were only as good as their human operators—the teams who would figure out the best *applications* of those tools within their own domains. I observed that, sure, teams can reach high percentages of unit test coverage to check the box and meet the metric, only to find that a large percentage of those tests are flaky—metric was met, but value was not delivered. The best teams didn't really need to care about metrics; they had purpose, discipline, pride, and responsibility—and the metrics in every case spoke for themselves. *Clean Craftsmanship* weaves all of these lessons and principles into practical code examples and experiences to illustrate the difference between writing something to meet a deadline versus actually building something sustainable for the future.

*Clean Craftsmanship* reminds us to never settle for less, to walk the Earth with *fearless competence*. This book, like an old friend, will remind you of what matters, what works, what doesn't, what creates risk, and what diminishes it. These lessons are timeless. You may find that you already practice some of the techniques contained within, and I bet you'll find something new, or at least something that you dropped because at some point you caved to deadlines or other pressures in your career. If you are new to the development world—whether in business or technology—you will learn from the very best, and even the most practiced and battle weary will find ways to improve themselves. Perhaps this book will help you find your passion again, renew your desire to improve your craft, or rededicate your energy to the search for perfection, regardless of the impediments on your horizon.

*Software developers rule the world,* and Uncle Bob is here again to remind us of the professional discipline of those with such power. He picks up where he left off with *Clean Code;* because software developers literally write the rules of humankind, Uncle Bob reminds us that we must practice a strict code of ethics, a responsibility to know what the code does, how people use it, and where it breaks. Software mistakes cost people their livelihoods—and their lives. Software influences the way we think, the decisions we make, and as a result of artificial intelligence and predictive analytics, it influences social and herd behavior. Therefore, we must be responsible and act with great care and empathy—the health and well-being of people depend on it. Uncle Bob helps us face this responsibility and *become the professionals that our society expects, and demands, us to be.*

As the Agile Manifesto nears its twentieth birthday at the writing of this foreword, this book is a perfect opportunity to go back to basics: a timely and humble reminder of the ever-increasing complexity of our programmatic world and how we owe it to the legacy of humankind—and to ourselves—to practice ethical development. Take your time reading *Clean Craftsmanship*. Let the principles seep into you. Practice them. Improve them. Mentor others. Keep this book on your go-to bookshelf. Let this book be your old friend—*your* Uncle Bob, *your* guide—as you make your way through this world with curiosity and courage.

*—Stacia Heimgartner Viscardi, CST & Agile Mentor*

*This page intentionally left blank*

# PREFACE

Before we begin, there are two issues we need to deal with in order to ensure that you, my gentle reader, understand the frame of reference in which this book is presented.

## ON THE TERM *CRAFTSMANSHIP*

The beginning of the twenty-first century has been marked by some controversy over language. We in the software industry have seen our share of this controversy. One term that is often called out as a failure to be inclusive is *craftsman*.

I've given this issue quite a bit of thought and talked with many people of varying opinions, and I've come to the conclusion that there is no better term to use in the context of this book.

Alternatives to craftsman were considered, including craftsperson, craftsfolk, and crafter, among others. But none of those terms carries the historical gravitas of *craftsman*. And that historical gravitas is important to the message here.

*Craftsman* brings to mind a person who is deeply skilled and accomplished in a particular activity—someone who is comfortable with their tools and their trade, who takes pride in their work, and who can be trusted to behave with the dignity and professionalism of their calling.

It may be that some of you will disagree with my decision. I understand why that might be. I only hope you will not interpret it as an attempt to be exclusive in any way—for that is, by no means, my intent.

## On the One True Path

As you read *Clean Craftsmanship: Disciplines, Standards, and Ethics*, you may get the feeling that this is the *One True Path to Craftsmanship*. It may be that *for me,* but not necessarily for you. I am offering this book to you as an example of *my path*. You will, of course, need to choose your own.

Will we eventually need *One True Path*? I don't know. Perhaps. As you will read in these pages, the pressure for a strict definition of a software profession is mounting. We may be able to get away with several different paths, depending on the criticality of the software being created. But, as you will read in what follows, it may not be so easy to separate critical from noncritical software.

One thing I am certain of. The days of "Judges"[1] are over. It is no longer sufficient that every programmer does what is right in their own eyes. Some disciplines, standards, and ethics *will* come. The decision before us today is whether we programmers will define them for ourselves or have them forced upon us by those who don't know us.

1. A reference to the Old Testament book of Judges.

## Introduction to the Book

This book is written for programmers and for managers of programmers. But in another sense, this book is written for all of human society. For it is we, programmers, who have inadvertently found ourselves at the very fulcrum of that society.

### For Yourself

If you are a programmer of several years' experience, you probably know the satisfaction of getting a system deployed and working. There is a certain pride you feel at having been part of such an accomplishment. You are proud of getting the system out the door.

But are you proud of *the way* you got that system out the door? Is your pride the pride of finishing? Or is your pride the pride of workmanship? Are you proud that the system has been deployed? Or are you proud of the way you built that system?

When you go home after a hard day of writing code, do you look at yourself in the mirror and say, "I did a good job today"? Or do you have to take a shower?

Too many of us feel dirty at the end of the day. Too many of us feel trapped into doing substandard work. Too many of us feel that low quality is expected and is necessary for high speed. Too many of us think that productivity and quality are inversely related.

In this book, I strive to break that mindset. This is a book about *working well*. This is a book about doing a good job. This is a book that describes the disciplines and practices that every programmer should know in order to work fast, be productive, and be proud of what they write every single day.

## For Society

The twenty-first century marks the first time in human history that our society has become dependent, for its survival, on a technology that has acquired virtually no semblance of discipline or control. Software has invaded every facet of modern life, from brewing our morning coffee to providing our evening entertainment, from washing our clothes to driving our cars, from connecting us in a world-spanning network to dividing us socially and politically. There is literally no aspect of life in the modern world that is not dominated by software. And yet those of us who build this software are little more than a ragtag batch of tinkerers who barely have any idea what we are doing.

If we programmers had had a better grasp on what we do, would the 2020 Iowa Caucus results have been ready when promised? Would 346 people have died in the two 737 Max crashes? Would Knight Capital Group have lost $460 million in 45 minutes? Would 89 people have lost their lives in Toyota's unintended acceleration accidents?

Every five years, the number of programmers in the world doubles. Those programmers are taught very little about their craft. They are shown the tools, given a few toy projects to develop, and are then tossed into an exponentially growing workforce to answer the exponentially growing demand for more and more software. Every day, the house of cards that we call software insinuates itself deeper and deeper into our infrastructure, our institutions, our governments, and our lives. And with every day, the risk of catastrophe grows.

Of what catastrophe do I speak? It is not the collapse of our civilization nor the sudden dissolution of all the software systems at once. The house of cards that is due to collapse is not composed of the software systems themselves. Rather, it is the fragile foundation of public trust that is at risk.

Too many more 737 Max incidents, Toyota unintended acceleration incidents, Volkswagen California EPA incidents, or Iowa Caucus incidents—too many more cases of high-profile software failures or malfeasance—and the lack of our discipline, ethics, and standards will become the focus of a distrustful and enraged public. And then the regulations will follow: regulations that none of us should desire; regulations that will cripple our ability to freely explore and expand the craft of software development; regulations that will put severe restrictions on the growth of our technology and economy.

It is not the goal of this book to stop the headlong rush into ever-more software adoption. Nor is it the goal to slow the rate of software production. Such goals would be a waste of effort. Our society needs software, and it will get it no matter what. Attempting to throttle that need will not stop the looming catastrophe of public trust.

Rather, it is the goal of this book to impress upon software developers and their managers the need for discipline and to teach those developers and managers the disciplines, standards, and ethics that are most effective at maximizing their ability to produce robust, fault-tolerant, effective software. It is only by changing the way that we programmers work, by upping our discipline, ethics, and standards, that the house of cards can be shored up and prevented from collapse.

## The Structure of This Book

This book is written in three parts that describe three levels: disciplines, standards, and ethics.

*Disciplines* are the lowest level. This part of the book is pragmatic, technical, and prescriptive. Programmers of all stripes will benefit from reading and understanding this part. Within the pages of this part are several references to videos. These videos show the rhythm of the test-driven development and refactoring disciplines in real time. The written pages of the book try to capture that rhythm as well, but nothing serves quite so well as videos for that purpose.

*Standards* are the middle level. This section outlines the expectations that the world has of our profession. This is a good section for managers to read, so that they know what to expect of professional programmers.

*Ethics* are at the highest level. This section describes the ethical context of the profession of programming. It does so in the form of an oath, or a set of promises. It is laced with a great deal of historical and philosophical discussion. It should be read by programmers and managers alike.

## A Note for Managers

These pages contain a great deal of information that you will find beneficial. They also contain quite a bit of technical information that you probably don't need. My advice is that you read the introduction of each chapter and stop reading when the content becomes more technical than you need. Then go to the next chapter and start again.

Make sure you read Part II, "The Standards," and Part III, "The Ethics." Make sure you read the introductions to each of the five disciplines.

# ACKNOWLEDGMENTS

Thank you to my intrepid reviewers: Damon Poole, Eric Crichlow, Heather Kanser, Tim Ottinger, Jeff Langr, and Stacia Viscardi. They saved me from many a faltering step.

Thank you also to Julie Phifer, Chris Zahn, Menka Mehta, Carol Lallier, and to all those at Pearson who work so tirelessly to make these books work out so well.

As always, thank you to my creative and talented illustrator, Jennifer Kohnke. Her pictures always make me smile.

And, of course, thank you to my lovely wife and wonderful family.

*This page intentionally left blank*

# About the Author

**Robert C. Martin** (**Uncle Bob**) wrote his first line of code at the age of 12 in 1964. He has been employed as a programmer since 1970. He is cofounder of cleancoders.com, offering online video training for software developers, and founder of Uncle Bob Consulting LLC, offering software consulting, training, and skill development services to major corporations worldwide. He served as the Master Craftsman at 8th Light, Inc., a Chicago-based software consulting firm.

Mr. Martin has published dozens of articles in various trade journals and is a regular speaker at international conferences and trade shows. He is also the creator of the acclaimed educational video series at cleancoders.com.

Mr. Martin has authored and edited many books, including the following:

*Designing Object-Oriented C++ Applications Using the Booch Method*
*Patterns Languages of Program Design 3*
*More C++ Gems*
*Extreme Programming in Practice*
*Agile Software Development: Principles, Patterns, and Practices*
*UML for Java Programmers*
*Clean Code*
*The Clean Coder*
*Clean Architecture: A Craftsman's Guide to Software Structure and Design*
*Clean Agile: Back to Basics*

A leader in the industry of software development, Mr. Martin served three years as the editor-in-chief of the *C++ Report*, and he served as the first chairperson of the Agile Alliance.

# 1 Craftsmanship

The dream of flying is almost certainly as old as humanity. The ancient Greek myth describing the flight of Daedalus and Icarus dates from around 1550 BCE. In the millennia that followed, a number of brave, if foolish, souls have strapped ungainly contraptions to their bodies and leapt off cliffs and towers to their doom in pursuit of that dream.

Things began to change about five hundred years ago when Leonardo DaVinci drew sketches of machines that, though not truly capable of flight, showed some reasoned thought. It was DaVinci who realized that flight could be possible because air resistance works in both directions. The resistance caused by pushing down on the air creates lift of the same amount. This is the mechanism by which all modern airplanes fly.

DaVinci's ideas were lost until the middle of the eighteenth century. But then began an almost frantic exploration into the possibility of flight. The eighteenth and nineteenth centuries were a time of intense aeronautical research and experimentation. Unpowered prototypes were built, tried, discarded, and improved. The science of aeronautics began to take shape. The forces of lift, drag, thrust, and gravity were identified and understood. Some brave souls made the attempt.

And some crashed and died.

In the closing years of the eighteenth century, and for the half century that followed, Sir George Cayley, the father of modern aerodynamics, built experimental rigs, prototypes, and full-sized models culminating in the first manned flight of a glider.

And some still crashed and died.

Then came the age of steam and the possibility of powered manned flight. Dozens of prototypes and experiments were performed. Scientists and enthusiasts alike joined the gaggle of people exploring the potential of flight. In 1890, Clément Ader flew a twin-engine steam-powered machine for 50 meters.

And some still crashed and died.

But the internal combustion engine was the real game-changer. In all likelihood, the first powered and controlled manned flight took place in 1901 by Gustave Whitehead. But it was the Wright Brothers who, on December 17, 1903, at Kill Devil Hills, North Carolina, conducted the first truly sustained, powered, and controlled manned flight of a heavier-than-air machine.

And some still crashed and died.

But the world changed overnight. Eleven years later, in 1914, biplanes were dogfighting in the air over Europe.

And though many crashed and died under enemy fire, a similar number crashed and died just learning to fly. The principles of flight might have been mastered, but the *technique* of flight was barely understood.

Another two decades, and the truly terrible fighters and bombers of World War II were wreaking havoc over France and Germany. They flew at extreme altitudes. They bristled with guns. They carried devastating destructive power.

During the war, 65,000 American aircraft were lost. But only 23,000 of those were lost in combat. The pilots flew and died in battle. But more often they flew and died when no one was shooting. We still didn't know *how* to fly.

Another decade saw jet-powered craft, the breaking of the sound barrier, and the explosion of commercial airlines and civilian air travel. It was the beginning of the jet age, when people of means (the so-called jet set) could leap from city to city and country to country in a matter of hours.

And the jet airliners tore themselves to shreds and fell out of the sky in terrifying numbers. There was so much we still didn't understand about making and flying aircraft.

That brings us to the 1950s. Boeing 707s would be flying passengers from here to there across the world by the end of the decade. Two more decades would see the first wide-body jumbo jet, the 747.

Aeronautics and air travel settled down to become the safest and most efficient means of travel in the history of the world. It took a long time, and cost many lives, but we had finally learned how to safely build and fly airplanes.[1]

Chesley Sullenberger was born in 1951 in Denison, Texas. He was a child of the jet age. He learned to fly at age sixteen and eventually flew F4 Phantoms for the Air Force. He became a pilot for US Airways in 1980.

On January 15, 2009, just after departure from LaGuardia, his Airbus A320 carrying 155 souls struck a flock of geese and lost both jet engines. Captain Sullenberger, relying on over twenty thousand hours of experience in the air, guided his disabled craft to a "water landing" in the Hudson River and, through sheer indomitable skill, saved every one of those 155 souls. Captain Sullenberger excelled at his craft. Captain Sullenberger was a craftsman.

---

The dream of fast, reliable computation and data management is almost certainly as old as humanity. Counting on fingers, sticks, and beads dates back thousands of years. People were building and using abacuses over four thousand years ago. Mechanical devices were used to predict the movement of stars and planets some two thousand years ago. Slide rules were invented about four hundred years ago.

In the early nineteenth century, Charles Babbage started building crank-powered calculating machines. These were true digital computers with memory and arithmetic processing. But they were difficult to build with the metalworking technology of the day, and though he built a few prototypes, they were not a commercial success.

1. The 737 Max notwithstanding.

In the mid-1800s, Babbage attempted to build a much more powerful machine. It would have been steam powered and capable of executing true programs. He dubbed it the *Analytical Engine*.

Lord Byron's daughter, Ada—the Countess of Lovelace—translated the notes of a lecture given by Babbage and discerned a fact that had apparently not occurred to anyone else at the time: *the numbers in a computer need not represent numbers at all but can represent things in the real world*. For that insight, she is often called the world's first true programmer.

Problems of precise metalworking continued to frustrate Babbage, and in the end, his project failed. No further progress was made on digital computers throughout the rest of the nineteenth and early twentieth centuries. During that time, however, mechanical *analog* computers reached their heyday.

In 1936, Alan Turing showed that there is no general way to prove that a given Diophantine[2] equation has solutions. He constructed this proof by imagining a simple, if infinite, digital computer and then proving that there were numbers that this computer could not calculate. As a consequence of this proof, he invented finite state machines, machine language, symbolic language, macros, and primitive subroutines. He invented, what we would call today, software.

At almost exactly the same time, Alonzo Church constructed a completely different proof for the same problem and consequently developed the lambda calculus—the core concept of functional programming.

In 1941, Konrad Zuse built the first electromechanical programmable digital computer, the Z3. It consisted of more than 2,000 relays and operated at a clock rate of 5 to 10Hz. The machine used binary arithmetic organized into 22-bit words.

During World War II, Turing was recruited to help the "boffins" at Bletchley Park decrypt the German Enigma codes. The Enigma machine was a simple

2. Equations of integers.

digital computer that randomized the characters of textual messages that were typically broadcast using radio telegraphs. Turing aided in the construction of an electromechanical digital search engine to find the keys to those codes.

After the war, Turing was instrumental in building and programming one of the world's first electronic vacuum tube computers—the Automatic Computing Engine, or ACE. The original prototype used 1,000 vacuum tubes and manipulated binary numbers at a speed of a million bits per second.

In 1947, after writing some programs for this machine and researching its capabilities, Turing gave a lecture in which he made these prescient statements:

> *We shall need a great number of mathematicians of ability [to put the problems] into a form for computation.*
>
> *One of our difficulties will be the maintenance of an appropriate discipline, so that we do not lose track of what we are doing.*

And the world changed overnight.

Within a few years, core memory had been developed. The possibility of having hundreds of thousands, if not millions, of bits of memory accessible within microseconds became a reality. At the same time, mass production of vacuum tubes made computers cheaper and more reliable. Limited mass production was becoming a reality. By 1960, IBM had sold 140 model 70x computers. These were huge vacuum tube machines worth millions of dollars.

Turing had programmed his machine in binary, but everyone understood that was impractical. In 1949, Grace Hopper had coined the word *compiler* and by 1952 had created the first one: A-0. In late 1953, John Bachus submitted the first FORTRAN specification. ALGOL and LISP followed by 1958.

The first working transistor was created by John Bardeen, Walter Brattain, and William Shockley in 1947. They made their way into computers in 1953. Replacing vacuum tubes with transistors changed the game entirely. Computers became smaller, faster, cheaper, and much more reliable.

By 1965, IBM had produced 10,000 model 1401 computers. They rented for $2,500 per month. This was well within the reach of midsized businesses. Those businesses needed programmers, and so the demand for programmers began to accelerate.

Who was programming all these machines? There were no university courses. Nobody went to school to learn to program in 1965. These programmers were drawn from business. They were mature folks who had worked in their businesses for some time. They were in their 30s, 40s, and 50s.

By 1966, IBM was producing 1,000 model 360 computers every month. Businesses could not get enough. These machines had memory sizes that reached 64kB and more. They could execute hundreds of thousands of instructions per second.

That same year, working on a Univac 1107 at the Norwegian Computer Center, Ole-Johan Dahl and Kristen Nygard invented Simula 67, an extension of ALGOL. It was the first object-oriented language.

*Alan Turing's lecture was only two decades in the past!*

Two years later, in March 1968, Edsger Dijkstra wrote his famous letter to the *Communications of the ACM* (*CACM*). The editor gave that letter the title "Go To Statement Considered Harmful."[3] Structured programming was born.

In 1972, at Bell Labs in New Jersey, Ken Thompson and Dennis Ritchie were between projects. They begged time on a PDP 7 from a different project team and invented UNIX and C.

Now the pace picked up to near breakneck speeds. I'm going to give you a few key dates. For each one, ask yourself how many computers are in the

3. Edsger W. Dijkstra, "Go To Statement Considered Harmful," *Communications of the ACM* 11, no. 3 (1968).

world? How many programmers are in the world? And where did those programmers come from?

1970—Digital Equipment Corporation had produced 50,000 PDP-8 computers since 1965.

1970—Winston Royce wrote the “waterfall” paper, “Managing the Development of Large Software Systems.”

1971—Intel released the 4004 single-chip microcomputer.

1974—Intel released the 8080 single-chip microcomputer.

1977—Apple released the Apple II.

1979—Motorola released the 68000, a 16-bit single-chip microcomputer.

1980—Bjarne Stroustrup invented *C with Classe*s (a preprocessor that makes C look like Simula).

1980—Alan Kay invented Smalltalk.

1981—IBM released the IBM PC.

1983—Apple released the 128K Macintosh.

1983—Stroustrup renamed C with Classes to C++.

1985—The US Department of Defense adopted waterfall as the official software process (DOD-STD-2167A).

1986—Stroustrup published *The C++ Programming Language* (Addison-Wesley).

1991—Grady Booch published *Object-Oriented Design with Applications* (Benjamin/Cummings).

1991—James Gosling invented Java (called *Oak* at the time).

1991—Guido Van Rossum released Python.

1995—*Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley) was written by Erich Gamma, Richard Helm, John Vlissides, and Ralph Johnson.

1995—Yukihiro Matsumoto released Ruby.

1995—Brendan Eich created JavaScript.

1996—Sun Microsystems released Java.

1999—Microsoft invented C#/.NET (then called *Cool*).

2000—Y2K! The Millennium Bug.

2001—The Agile Manifesto was written.

Between 1970 and 2000, the clock rates of computers increased by three orders of magnitude. Density increased by four orders of magnitude. Disk space increased by six or seven orders of magnitude. RAM capacity increased by six or seven orders of magnitude. Costs had fallen from dollars per bit to dollars per gigabit. The change in the hardware is hard to visualize, but just summing up all the orders of magnitude I mentioned leads us to about a thirty orders of magnitude increase in capability.

And all this in just over fifty years since Alan Turing's lecture.

How many programmers are there now? How many lines of code have been written? How good is all that code?

Compare this timeline with the aeronautical timeline. Do you see the similarity? Do you see the gradual increase in theory, the rush and failure of the enthusiasts, the gradual increase in competence? The decades of not knowing what we were doing?

And now, with our society depending, for its very existence, on our skills, do we have the Sullenbergers whom our society needs? Have we groomed the programmers who understand their craft as deeply as today's airline pilots understand theirs? Do we have the craftsmen whom we shall certainly require?

---

Craftsmanship is the state of knowing how to do something well and is the outcome of good tutelage and lots of experience. Until recently, the software industry had far too little of either. Programmers tended not to remain programmers for long, because they viewed programming as a steppingstone into management. This meant that there were few programmers who acquired enough experience to teach the craft to others. To make matters worse, the number of new programmers entering the field doubles every five years or so, keeping the ratio of experienced programmers far too low.

The result has been that most programmers never learn the disciplines, standards, and ethics that could define their craft. During their relatively brief career of writing code, they remain unapprenticed novices. And this, of course, means that much of the code produced by those inexperienced programmers is substandard, ill structured, insecure, buggy, and generally a mess.

In this book, I describe the standards, disciplines, and ethics that I believe every programmer should know and follow in order to gradually acquire the knowledge and skill that their craft truly requires.

# Index