Adventures in Software Development Evolution and Revolution

JIM HIGHSMITH

Forewords by **Martin Fowler** *and* **Heidi J. Musser**

FREE SAMPLE CHAPTER

Praise for *Wild West to Agile*

"Jim Highsmith is the Forrest Gump of software development. What made the 1994 movie so entertaining was how frequently Forrest found himself in the right spot as history was being made. Unlike Forrest, though, Jim's actions influenced that history.

"Jim tells us stories from his early-career involvement in the Gemini and Apollo space projects and then working as a leader to bring about the shift to structured methods. From there he outlines how approaches such as Rapid Application Development planted the seeds that became agile software development.

"Throughout, Jim played a part in bringing about the changes that moved software development out of its Wild West beginnings and into its Agile present. The stories Jim tells in this book are entertaining and educational, and will be important to remember as we move into whatever the future holds for software development."

—Mike Cohn, co-founder of the Agile Alliance and the Scrum Alliance; author of *Succeeding with Agile*

"Jim provides a unique perspective not just from the sidelines, but from out on the playing field. If you want to understand the shape of software development today, this is the book for you. If you want to understand how to navigate a turbulent career with grace and style, this is also the book for you. If you enjoy memoirs, ditto. Since I first encountered him in the 1990s, Jim has been my model of a steady, thoughtful leader. Enjoy his story."

—Kent Beck, Chief Scientist, Mechanical Orchard; author, *Extreme Programming Explained: Embrace Change*

"A magnum opus from master storyteller, adventurer, nonconformist, and adaptable agile maven Jim Highsmith. With this braided narrative, Jim defines what it means to be truly agile, both personally and organizationally. A must-read for a fascinating first-person perspective and invaluable insights into the past, present, and future of agility. I haven't enjoyed a book as much in a very long, long time."

—**Sanjiv Augustine**, Founder and CEO, LitheSpeed

"The phrase 'been there, done that' has rarely been as true as it is for Jim Highsmith and his long and varied career in software. Jim is a great storyteller, and this book tells great stories about many of the leaders of our industry. Enjoy the ride!"

—**Rebecca Parsons**, Chief Technology Officer, Thoughtworks

"The evolution of the software industry from its humble beginnings is captured in a way that only Jim Highsmith can. His coupling of the historical evolution of the software industry with his own personal experiences helps to bring greater insight to the reader as he explains the driving business factors that led to the creation and evolution of the agile software movement. This is not just a walk down memory lane. As an

industry leader, Jim clearly demonstrates the next evolution for business and how the software industry is leading the change. Well done, Jim!"

—Ken Delcol, Former Director, Product Development, Sciex;
Advanced Program Manager, MDA

"*Wild West to Agile* is an exhilarating ride through the landscape of IT history. Accompanied by personal stories and humorous anecdotes, Jim Highsmith shares his reflections on the good, the bad, and the ugly of agile software development. By knowing the heroes of days gone by, and understanding what came before, you can prepare yourself for a better future in the technology business."

—Jurgen Appelo, creator of unFIX; author of
Management 3.0 and *Managing for Happiness*

"More than 70% of agile transformations fail, and many more fall short of expectations. In his book, Jim reflects on decades of learnings on technology innovation and encourages business leaders to embark in the *ultimate transformation journey*: the company-wide adoption of an agile mindset to achieve sustainable business success."

—Marcelo De Santis, Chief Digital Officer, Thoughtworks

"What happened over the last six decades in the software field? How did our methods, methodologies, and mindsets evolve? Who played a key role along the way, and where are we headed today? Only a veteran practitioner, industry legend, master storyteller, and agile pioneer like Jim Highsmith could tell this tale with such detail and depth."

—Joshua Kerievsky, CEO, Industrial Logic; author, *Joy of Agility*

"It is not often you get to read a front-lines view of the explosive software industry, running from 1966 to 2023—nearly 60 years of being an eyewitness to history! Not just witnessing history, but also driving it. This is an eminently readable narrative by a figure in our history."

—Alistair Cockburn, co-author, the Agile Manifesto

"More people every day join the digital workforce to enjoy the agile paradise, but they have not experienced a single real day of being inside a waterfall project or experiencing hardcore command-and-control management delivered in inhuman facilities in dark basements.

"What they enjoy today is the result of courageous battles against powerful executives fought by thought leaders like Jim, who believed that a better status quo was possible. Jim's braided approach in Chapters 1 to 6 sheds light on this past and will help the newcomers more meaningfully value the status quo they enjoy today.

"The last chapters of the book contain valuable keys for unlocking future challenges for both digital practitioners and analog C-executives who are willing to unlearn so that they can have a seat at the table in the future."

—Ricard Vilà, Chief Digital Officer, Latam Airlines (Chile)

"An entertaining and insightful book full of nostalgia and advice from a true leader in this space. This book has firsthand experiences and stories of the real struggles of becoming an adaptive organization. Coverage of topics such as the implementation of EDGE is a valuable addition."

—**Linda Luu**, Enterprise Strategy, IBM;
co-author, *EDGE: Value-Driven Digital Transformation*

"Since I just retired from the company I founded 15 years ago, Jim's memoir comes a day late for me. *Wild West to Agile* would have been the perfect vehicle to help this 'courageous executive' address the biggest challenge that I was facing at the time of my departure: the need to evangelize the agile mindset to the next generation of employees, whose focus is primarily on the pedantic pursuit of agile methods.

"Jim has been there and done that, and in *Wild West to Agile* he gives us an educational and entertaining ring-side seat to his historical travels and his many contributions to the software development industry. I'm honored to have played a small part in Jim's matriculation as an agilist and grateful for the mentoring he provided as I was transitioning out of the laboratory and into this century's most exciting business opportunity—creating valuable software for humanity."

—Sam Bayer, Founder and former CEO, Corevist

"This is a valuable retrospective on a journey from Apollo to SpaceX, through the evolution of technologies from vacuum tubes to billions of transistors on a chip, from the perspective of a leader in software methodologies and a signatory of the Agile Manifesto. Jim Highsmith's work with other experts on adaptive approaches to business and technology has turned agile software development from a great idea into an essential tool for business survival in the modern world for all successful technology companies."

—Jeff Sutherland, inventor and co-creator of Scrum and Scale;
signatory of the Agile Manifesto

"I began working with Jim when we both joined Exxon in the early 1970s. He and I were in the same systems group. I was a business guy by education; Jim was an engineer who was already moving forward in the software development world. I eventually became a manager in the accounting department and Jim a key player in implementing a new, quite complex financial system, which was an arduous task in those early days. Jim spent many days and nights as the driving force during implementation. A brilliant software developer, he had the dedication and work ethic to 'get the job done,' and was highly effective at working with all involved, from the accounting folks to the division executive."

—John Fahlberg, executive leadership coach; previous CFO, COO,
and CEO of early-stage growth companies

"Jim Highsmith is truly a pioneer who inspired and led the evolution of software development through the past six decades. *Wild West to Agile* is an enlightening and entertaining trip down memory lane, built on the stories and the people who were lucky enough to collaborate with Jim on that journey."

—Gary Walker, Former Manager, Software Development, MDS Sciex

"I thoroughly enjoyed *Wild West to Agile*, which chronicles Jim Highsmith's fascinating career. When I met Jim in the late 1990s, I felt I had finally met someone who was talking about software management in a way that made sense. His stories of success and failure are engaging and show how he constantly evolved to better ways. Highly recommended."

—Todd Little, Chairman, Kanban University

"Do you like listening to family elders tell their history? If you're a member of the software development community, here is your opportunity to hear its wild and wonderful history from a leader who's spent six decades affecting it."

—Gil Broza, author, *The Agile Mindset*

"Jim Highsmith was literally in the middle of the maelstrom that was the birth and evolution of agile software development. His perspective and stories are fascinating and telling. This book is a must-read for those who prefer to learn from history, rather than repeat it."

—David Robinson, Digital Transformation Partner, Thoughtworks;
co-author *EDGE: Value-Driven Digital Transformation*

"I have hoped for a book like this, an accurate historical accounting of the software development world, for quite some time. And Jim has certainly delivered. Any serious agilist, or serious software professional, needs to breathe in Jim's words. You won't be disappointed."

—Scott Ambler, author

Wild West to Agile

This page intentionally left blank

Wild West to Agile

Adventures in Software Development Evolution and Revolution

JIM HIGHSMITH

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: ShutterOK/Shutterstock

Author photo: Janet Meyer Photography

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2023934207

ISBN-13: 978-0-13-796100-9
ISBN-10: 0-13-796100-6

ScoutAutomatedPrintCode

Pearson's Commitment to Diversity, Equity, and Inclusion

Pearson is dedicated to creating bias-free content that reflects the diversity of all learners. We embrace the many dimensions of diversity, including but not limited to race, ethnicity, gender, socioeconomic status, ability, age, sexual orientation, and religious or political beliefs.

Education is a powerful force for equity and change in our world. It has the potential to deliver opportunities that improve lives and enable economic mobility. As we work with authors to create content for every product and service, we acknowledge our responsibility to demonstrate inclusivity and incorporate diverse scholarship so that everyone can achieve their potential through learning. As the world's leading learning company, we have a duty to help drive change and live up to our purpose to help more people create a better life for themselves and to create a better world.

Our ambition is to purposefully contribute to a world where:

- Everyone has an equitable and lifelong opportunity to succeed through learning.
- Our educational products and services are inclusive and represent the rich diversity of learners.
- Our educational content accurately reflects the histories and experiences of the learners we serve.
- Our educational content prompts deeper discussions with learners and motivates them to expand their own learning (and worldview).

While we work hard to present unbiased content, we want to hear from you about any concerns or needs with this Pearson product so that we can investigate and address them.

- Please contact us with concerns about any potential bias at https://www.pearson.com/report-bias.html.

This page intentionally left blank

For their inspiration, encouragement, and support

Grandkids: Zach, Ellie, Ruby

Daughters: Nikki, Debbie, and goddaughter Amy

My life partner, Wendie, who kept me sane by periodically prying me from my writing desk.

This page intentionally left blank

Contents

Foreword

I STILL REMEMBER WHEN I first saw Jim, in the late 1990s on a stage at a software conference in far-away Wellington, New Zealand. As someone immersed in Extreme Programming, I didn't expect much from someone steeped in the traditional software engineering processes of the time. Yet I experienced a talk full of refreshing thinking, giving credible reasons and citing experiences that resonated with my own sense of modern software management.

Jim's conference biography only hinted at his experiences thus far—a programmer from the early days of computers, who went deep into structured methods, but also saw their weaknesses. The decade before this talk, he'd been actively exploring a new route, one that had a lot of similarities with my very different community.

After that time in New Zealand, our paths crossed more often, although it became a running joke that we rarely met in the country in which we both lived. Jim's book *Adaptive Software Development* was an influence on many people in my circles. A few years later, we were together in Snowbird writing the Manifesto for Agile Software Development. Since then, it's been a mixed couple of decades. The approach we advocate has come further than we thought it would, but it continues to run into obstacles, often caused by the common human inclination to favor surface impressions over deeper understanding. Jim has helped tackle this challenge head on, rethinking the dreaded project management triangle, teaching managers to live with ambiguity, and mentoring new generations of software developers to work in this new style.

Jim's history in the software industry has put him at the forefront of a wave of changes, and I've enjoyed learning about his full story as I've read drafts of this book. It's a personal book, one that can only be written by

someone who values adventure but also understands that you need the right training and equipment to get down from the mountain safely. Reading this memoir of someone who worked in the heart of Monumental Methodologies but recognized their limitations and cut a path out of them, I'm learning about many things that influence our current world. I've always felt that understanding history is important, because it's hard to understand where we are unless you understand the path that we took to get here. Jim's memoir is an entertaining and astute odyssey through this history.

—**Martin Fowler**, Chief Scientist, Thoughtworks

Foreword

I'VE SPENT MOST OF MY CAREER—which included five different C-level roles encompassing six different business models—leading and advising businesses on designing new operating and engagement models to drive digital transformation and achieve enterprise agility through the adoption of fundamentally different ways of working, thinking, and being. Like Jim, what I've learned—sometimes quickly, sometimes slowly—is that "Waiting for something to happen and relying on your ability to adapt is one thing, but building a sustainable enterprise having a greater *capability* to adapt is even better." I can assure you, Jim never waited for something to happen: He was a true adventurer and pioneer throughout his career.

This book is an extraordinary trip down memory lane! Jim meticulously describes the evolution of software methods, methodologies, and mindsets through the eras of software development that occurred during his extraordinary six-decades-long career. A prolific storyteller, Jim guides us through this journey from the perspectives of both his personal experiences and the experiences of the adventurous pioneers he encountered along the way, and through the lenses of technical innovation and management trends during these periods.

Jim's last book, *EDGE: Value-Driven Digital Transformation*, which he co-authored with David Robinson and Linda Luu, helped leaders unleash the promise of agile development. It also helped leaders build the capabilities to transform and demanded that one develop the capacity to embrace and lead change. This book, *Wild West to Agile: Adventures in Software Development Evolution and Revolution*, will help all of us who are interested in learning about how agile practices evolved to the pillars of agility: to consistently deliver customer value, to foster enterprise benefits, and to build a

sustainable enterprise. Agile methods and methodologies will continue to adapt and evolve, but the need for enterprise agility will not lessen.

What strikes me most about this book is Jim's commitment to preparing for the future by learning from the past. Not only does he honor the pioneers of software development, but he does so in a way that gives both my generation and younger generations insights into the events that we lived through, events that we may have missed, so that the seeds of agility planted and germinated decades ago will continue to flower in the future.

Finally, I would be remiss if I did not note the underlying thread that is interwoven through Jim's braided narratives. This entire journey—beginning with the Wild West era of software development through the Agile era to today's Digital Transformation era—*is entirely empowered by people*. Thank you, Jim, for sharing these beautiful stories and honoring the people who were a part of this amazing journey.

—Heidi J. Musser, Board Member, Board Advisor,
Executive Consultant, Vice President and CIO,
USAA, *retired*

Preface

WHY *WILD WEST TO AGILE*? As I retired and began writing a family-centered memoir for my grandkids, I realized I was taking a trip back through time in all-embracing ways I'd not done before. These reminiscence trips through my outdoor and career adventures were both revealing and thought provoking. Every now and then, I would stop and ask younger colleagues if they knew of Tom DeMarco or Jerry Weinberg or Ken Orr. They didn't. They knew about Azure and Ruby, and agile practices, but little about software development history. Technology blazes forward, leaving little time to contemplate the past.

I wanted to write about the history of software development, embellish it with my personal experiences, and introduce the people, the pioneers, who strived to make the world a better place, by building better software. Pioneers—whether 1800s fur trapper Jim Bridger, Apollo astronauts, structured software developer Ken Orr, or agile methodologist Kent Beck—displayed adventurousness, adaptability, and nonconformity. I wanted to resurrect experiences shared with colleagues of earlier generations and offer a sense of perspective to colleagues of a more recent generation.

COVID-19. Lockdown. Retirement. Building project completed. Languishing. What next? These were the thoughts running through my mind as 2022 got under way. As I began to remember, research, and find old emails and documents, the idea of turning a family memoir into a book began to take shape. I had the idea of organizing the book around different eras of software development and writing about my work, stories, experiences, and observations during each era. So, little by little this book evolved from a few fuzzy narrative chunks. I wanted to explore how and why the software industry evolved from the ad hoc code scribbling of the 1960s to the blizzard of methods, methodologies, and tools available in 2022.

Both my career and software development in general were hugely impacted by changes in information technology (IT). One simple illustration: An iPhone with 64 gigabytes of memory has 250,000 times more bytes of memory than the IBM 360 mainframe computer that I worked on in the early 1970s. In 2021, a gigabyte of memory cost about $10. In the Wild West era, although a gigabyte of memory was technologically[1] impossible, the cost for that gigabyte would be nearly $734 million![2] We need to remember that methods, methodologies, and mindsets all evolved to solve the problems of each era, and were both enabled and constrained by the technology of that time.

As I explored this history, *Wild West to Agile* evolved into a work of braided, creative nonfiction. Nonfiction, as the name suggests, is the opposite of fiction. I've always wondered why this genre was named the "non" of something else. Books about technology and science are usually nonfiction and, regrettably, sometimes tedious to a non-researcher. Enter "creative" nonfiction, whose writers use literary craft elements of character, story, structure, tension, and plot to make nonfiction readable and enjoyable. Simply put, they are "true stories, well told."

Braided narrative is the name bestowed on a nonfiction (or fiction) subtype. One braid tells the author's personal story, while another explores an environmental or social justice issue, or a historical event. These two story lines weave together over time, each enhancing the other to create a cohesive whole.

Wild West to Agile weaves together several braids. The first includes the overall evolution and various revolutions occurring in software development over four distinct eras. The second describes my personal and client experiences during each era. The third pays tribute to the adventurous, innovative pioneers. The fourth and fifth braids are technology innovations and management trends.[3]

Writing in this braided narrative genre provided two benefits: scope and stories. A book that purported to be about "the" history of software development would be far beyond my interest or capability. Limiting the scope to events I participated in narrowed the coverage substantially. My career was exclusively involved with business systems (except in my early years

1. The maximum memory available for the popular IBM 360/30 was 64K.
2. https://ourworldindata.org/grapher/historical-cost-of-computer-memory-and-storage?country=~OWID_WRL
3. These braids are further explained in Chapter 1.

as an electrical engineer). I wasn't involved in scientific or engineering computing, never wrote compilers or operating systems, never wrote complex algorithms, never worked on Unix systems. What I did work on were business systems, such as those used for accounting, finance, order processing, inventory management, and transportation. What I did work on were methods and methodologies that improved software development. What I did work on were technical, project management, organizational, and leadership issues.

Terminology was a challenge. Today's popular terminology was probably not yesterday's. Should I use the term *software development*, *software delivery*, or *software engineering*? Controversy over terms was, and remains, rampant. Is software engineering really engineering? Is software development a subset of software engineering or a superset? And on and on. My first inclination was to jump into the definitional fray, but then I rethought this decision: "That's the road to lunacy!" So, considering my own personal preference, I used *software development* as a broad term and threw in a few software engineering labels when it felt appropriate. In *Wild West to Agile*, my definition of software development encompasses a complete range of activities—from product and project management, to requirements, design, programming, testing, and deployment.

Another conundrum was topic timing. For example, the term *object-oriented programming* first appeared in the mid-1960s, but had limited use until the 1990s, when the market expanded rapidly. Technical debt followed a similar path. My guideline was to delve into topics during their market expansion period.

I was lucky, fortunate, and humbled to collaborate with two generations of software development pioneers. In the early eras, I was colleagues with people like Ken Orr, Tom DeMarco, Tim Lister, Ed Yourdon, Larry Constantine, and Jerry Weinberg. As 1999 turned the corner into this century, I added agilists Alistair Cockburn, Pat Reed, Kent Beck, Mike Cohn, Ken Schwaber, Jeff Sutherland, and Martin Fowler to this list.[4]

My goals for this book are as follows:

- Document the evolution and revolutions of software methods, methodologies, and mindsets.
- Remember and honor the pioneers of software development.

4. More about each of these people in later chapters.

- Prepare for the future, by learning from the past.
- Give my generation a vehicle to reminisce about events we lived through.
- Give younger generations a peek into events they may have missed.

Additionally, I wanted my grandkids to know more about me, to understand my career and explore its purpose.

Finally, a word about my lens into software development history. "Perspective is the point of view that a person sees a historical event from. . . . Every source has a perspective."[5] I've approached this history from my perspective, which of course includes my age, education, work experience, and geography, but also race, gender, sexual preference, and religion. A software history written by a New Yorker would look different from one written by a Silicon Valley alum. The lens of a marginalized person—female, BIPOC, LBGTQ+, or a person with a disability—would be different from mine, very different. I can only write from my perspective, my lens, but I can also acknowledge and support the blossoming goals around diversity, equity, and inclusion.[6]

The braids of this book weave together to tell a story. For mountaineers, a tightly braided climbing rope binds them together into a collaborative, self-organizing team. There are many kinds of braids that bring people together.

5. historyskills.com, https://www.historyskills.com/2019/03/22/what-s-the-difference-between-perspective-and-bias/
6. More about diversity in the Afterword.

Acknowledgments

HOW DOES ONE WRITE an acknowledgments section when the time span is six decades? My solution was to hack away at it and hope I didn't leave anyone out. I owe much to many.

A special thanks to my consulting clients who, over many years, have courageously endeavored to try new methods, methodologies, and mindsets.

The best products come from collaborative efforts. The content, structure, and flow of this book were greatly enhanced by the invaluable contributions of Heidi Musser, Amy Irvine, Martin Fowler, Barton Friedland, Freddy Jandeleit, Pat Reed, Ken Collier, and Mike Cohn.

Thanks to my industry colleagues, some of whom I have worked with over a span of years: Sam Bayer, Donna Fitzgerald, Jurgen Appelo, Josh Kerievsky, Ken Schwaber, Kent Beck, Anne Mullaney, Sanjiv Augustine, Scott Ambler, Linda Luu, Kevin Tate, Jerry Gordon, Morris Nelson, Lynne Nix, Steve Smith, Gary Walker, Jeff Sutherland, Chris Guzikowski, Mac Lund, Ken Delcol, Larry Constantine, Israel Gat, Tom DeMarco, Tim Lister, Ken Orr, Martyn Jones, Michael Mah, Ricard Vilà, Todd Little, Dave Higgins, John Fahlberg, Karen Coburn, Gil Broza, Alistair Cockburn, Ed Yourdon, Jerry Weinberg, and Wendy Eakin.

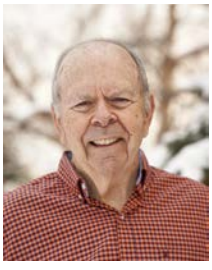
Although I retired from Thoughtworks in 2021, a number of Thoughtworks colleagues contributed to this book: Chad Wathington, Rebecca Parsons, Angela Ferguson, Mike Mason, Neal Ford, Roy Singham, David Robinson, Marcelo De Santis, and Xiao Guo.

Thanks to graphics designer Mustafa Hacalaki, who provided just the style and tone I wanted for the book's graphics.

Thanks also to the wonderful staff at Pearson who shepherded me through the editing and production process—executive editor Haze Humbert, developmental editors Adriana Cloud and Sheri Replin, copy editor Jill Hobbs, and the rest of the production team.

This page intentionally left blank

About the Author



Jim Highsmith retired as Executive Consultant at Thoughtworks, Inc., in 2021. Prior to his tenure at Thoughtworks, he was director of Cutter Consortium's Agile Project Management practice. He has nearly 60 years' experience as an IT manager, product manager, project manager, consultant, software developer, and storyteller. Jim has been a leader in the agile software development community for the past three decades.

Jim is the author of *EDGE: Value-Driven Digital Transformation* (2020; with Linda Luu and David Robinson); *Adaptive Leadership: Accelerating Enterprise Agility* (2013); *Agile Project Management: Creating Innovative Products* (2009); *Agile Software Development Ecosystems* (2002); and *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems* (2000), winner of the prestigious Jolt Award. Jim is the recipient of the 2005 international Stevens Award for outstanding contributions to software engineering.

Jim is a co-author of the Agile Manifesto, a founding member of the Agile Alliance, co-founder and first president of the Agile Leadership Network, and co-author of the Declaration of Interdependence for project leaders. Jim has consulted with information technology organizations and software companies worldwide.

This page intentionally left blank

2

Wild West

(1966–1979)

"A SMALL STEP for man, one giant leap for mankind," said Neil Armstrong, venturing onto the lunar landscape for the first time. Upon college graduation in 1966, I had several electrical engineering job offers—Westinghouse in Pittsburgh, Pennsylvania; IBM in Poughkeepsie, New York; and Pan American World Airways Aerospace in Coco Beach, Florida, working on the Apollo moon landing program. My selection process was not difficult. Pan Am had a contract with the Air Force to manage the missile flight operations at Cape Canaveral (not at the Kennedy Space Center, a National Aeronautics and Space Administration [NASA] facility), so I jumped at the chance to be a tiny part of Apollo.

Apollo

My first assignment was looking over engineering drawings and calculating component and system mean-time-to-failure (MTTF) and mean-time-to-repair (MTTR). One time, I flew to the primary downrange missile tracking station on Ascension Island in the middle of the Atlantic Ocean to oversee a computer memory upgrade and test. What I remember most about the trip was the discomfort of flying in an Air Force C-130, losing an engine in-flight, overnighting in Antigua, and the happy hour at the officer's club on Ascension when drinks were 10 cents.

Figure 2.1 *Apollo downrange tracking ship.*[1]

In the early 1960s when Apollo was being designed, global communications was iffy, so five ships were retrofitted with radars, telemetry, inertial navigation systems, and a fully functioning, but smaller, mission control center mirroring the one in Houston. These ships were to be deployed to the Pacific Ocean to acquire and track the command module as it returned to Earth. I worked on two of these ships, like the one shown in Figure 2.1.

"Work" was somewhat of a misnomer for my job. In reality, I reviewed and approved others' work. The number of entities involved in the Apollo mission was startling to a newbie like me. As you would expect, there were multiple contractors for the ships themselves, for everything from anchor chains to computers. What I did not expect was the contract management flotilla—NASA, NASA prime contractors building things, other contractors to monitor the primary contractors, and the Air Force, with the same menagerie

1. Photo courtesy of CPC Collection/Alamy Stock Photo

of which Pan Am was a part. For a computer test, for example, there might be a contractor running the test and several observers, each of whom supplied their own hierarchy with reports about the success or failure of the test.

Patrick Air Force Base (AFB), near Cocoa Beach, was the primary launch site for the Mercury and Gemini missions (and military vehicles like Titan), while the launch complex for Apollo was being constructed on Merritt Island, north of Patrick AFB. My office was located at Patrick. One morning, as a friend and I walked into work, we saw tourists standing around the life-sized missile display in front of the building (Figure 2.2). We were dressed for the times in white shirts, thin dark ties, and—the sure sign of an engineer—pocket protectors jammed with pens. We walked over to one of the missiles, looking seriously up and down and around, and started yelling, "Ten, nine, eight, …." as we ran in the other direction. Wow, the tourists took off! We laughed all the way into the office.

By the time I arrived, the Mercury program was completed, Gemini was under way, and Apollo was getting ready for early test launches of the Saturn rocket. My next-door neighbor was an Air Force camera operator, and

Figure 2.2 *Missiles outside Patrick Airforce Base.*[2]
(Courtesy of Air Force Space and Missile Museum.)

2. https://afspacemuseum.org/sites/patrick-air-force-base-florida/

he invited me to accompany him to watch launches from extremely close up—almost hair-scorchingly close. We were about a mile in front of the news media cameras. I made it to the top of a Saturn rocket on the launch pad and got to look inside the Apollo command module (but unfortunately did not go in). During the first Saturn test flight (at the Patrick site), the ground shook so violently that three separate power feeds to the primary computer facility shut down—they were down for the first 2 to 3 minutes of the flight.

The ships had C-band and S-band radar, telemetry for data downloads, and inertial navigation (secret at the time, used on nuclear subs, and predating the Global Positioning System [GPS]). The computer was a reduced-size, hardened military computer used on Navy ships. It had a red "battle" button so it could run past its temperature limits (and literally burn up) while a battle was under way. This 8-bit machine had a 32-bit word length and an eye-popping 36K of memory (36K was the maximum at the time). At sea, our programming interface was punched paper tape (on land, punch cards). There was a control panel with 3 or 4 columns across and about 10 to 12 rows down—each intersection was a button that signaled currently unneeded programs to roll out of memory so others could be read in.

Consider trying to program complex orbital mechanics calculations to acquire a tiny command module as it came up over the distant horizon given the memory and processing speed constraints. Besides radar and navigation data, another calculation input was ship flex data.[3] The command module acquisition calculation was so sensitive it required ship flexure data in the calculation—all programmed in 36K.

For six months, I was on temporary duty at a shipyard in New Orleans, where two ships were being built. We went out on sea trials—down the Mississippi River to the Gulf of Mexico to test the systems by trying to find airplanes. In an actual mission, the command module would enter the atmosphere at 24,000 miles per hour, then slow down by using retro-burn engines to less than 350 miles per hour. Our target planes flew right over the ship at several hundred miles per hour. The first few tests we couldn't even find the planes.

The entire Apollo program was huge, and its success was a testament to a big vision, creativity, collaboration, learning through failure, engineering expertise, and management talent. It was a fun, but busy time, and it was

3. Obtained by measurements in a narrow tunnel between the radars through which a laser was beamed to measure the ship's flex.

great for me to play even a small part in this event. Looking back, it was an exciting way to begin my career adventure. However, by the first manned Apollo flight, the ships had been repurposed due to advances in communications.

BEFORE CONTINUING WITH MY CAREER stops, a couple of leading-edge projects, the state of software methods, and establishing a management context for the entire six decades, I first need to set the technology stage for this Wild West era.

Technology and the world

Apollo was an electric piece of the wider world during the early part of the Wild West era, which included events like the Beatles, the Vietnam War, flower children, antiwar protests, President Richard Nixon and Watergate, and rising inflation. While the global geopolitical and social changes were significant, business leaders continued much as before. The economy was hit by oil crises in 1973–1974 and 1979, whose major impacts included rapid wage and price inflation, which slowed business growth. The combination of these conditions inspired the term *stagflation*. Retail sales sank, and corporate profit margins suffered.

Some big businesses suffered. In particular, manufacturing titans like General Motors and Ford lost ground to European and Japanese car makers, whose vehicles offered both higher mileage ratings and lower costs. But there was another emerging trend. New companies like Apple (1976), Starbucks (1971), Microsoft (1975), and Nike (1964) showed smaller, nimble companies might have a future.

The 1950s had laid the groundwork for 50 years of corporate myopia. The economy boomed, people had jobs, prosperity seemed inevitable, and the future appeared to stretch out into a grand undertaking (unless, of course, you were female, BIPOC [Black, Indigenous, and people of color], LBGTQ+, or a person with a disability). Corporate executives planned for the future as if the progression was predictable and linear. Some things did change, however, so businesses had to adapt, but within acceptable limits. This assumption of predictability infused everything from business planning to project management. "Plan the work and Work the plan" was the corporate mantra. This predictability and internal focus led to trends like

management by objectives (MBO) and cost and schedule as the primary objectives of project management. Even into the mid-1980s, big corporations had large planning departments.

In the late 1960s and all of the 1970s, IBM dominated the mainframe business computer market. Prior to that time, IBM offered different lines of computers depending on how much processing power a customer required. Unfortunately, these computers, which had designations like 1620 and 7064, were incompatible, so upgrading from one size to the next was difficult and expensive. The IBM 360/30, released in 1964, was the first of the 360 series of computers having, among many innovations, a common operating system. Magnetic tape drives provided external storage for these early systems.

Beginning in the 1970s, IBM began delivering random-access disk drives with its 360 computers. A small configuration of 2314 disk drives, with 146 MB of storage, sold for $175,000[4] (today that much storage would cost about ½ cent, not adjusting for inflation). Commensurate with the development of disk drives, IBM introduced an early database management system, called Information Management System (IMS). Random-access drives and IMS introduced new complexity, and new opportunities, into software development.

The rise to prominence of minicomputers began in the 1970s and extended into the 1980s, led by Digital Equipment Corporation (DEC), which released the PDP-8 in the late 1960s. DEC developed ever more powerful minis, driving Data General, another major manufacturer, to release its Eclipse superminicomputer in 1980. The intense development effort of the Eclipse was documented in a Pulitzer Prize–winning book, Tracy Kidder's *The Soul of a New Machine* (1981). Still, IBM mainframe computers dominated business computing during this entire era.

Interactions with computers during this time were primitive and impersonal (as illustrated in Figure 1.5). Large mainframe computers resided in specially constructed rooms with raised floors, overhead wire bins, and serious air conditioning.[5] Computer operators input card decks, mounted and dismounted tape and disk drives, and gathered and distributed printouts. Because disk storage was so expensive, most systems used a combination of storage forms—magnetic tape for high-volume data, disks for lower-volume data. Online, time-sharing systems were available on minicomputer systems

4. 2314 drive performance and cost in 1970, IBM Archives, www.ibm.com/ibm/history/exhibits/storage/storage_2314.html.

5. No one worried about energy costs or environmental impacts in those days.

using Unix and some mainframes, but were primarily reserved for academic and engineering applications.

Software was poorly understood by business executives. They could see the mammoth computers, but the software was hidden. In addition, most of the vendor-supplied software during this time was included in the price of the hardware—software appeared to be free!

WANTING TO DESIGN and build rather than audit, I quit Pan Am and relocated to Saint Paul, Minnesota, to work for Univac Federal Systems Division, which manufactured computers for the Navy and Apollo ships. I was involved in designing gates and registers for computers and early communication modem design. After two engineering jobs, I began to see myself as more a generalist than a specialist and decided to pursue an MBA degree, attending night school at the University of Minnesota for the prerequisite accounting and economics courses. I had anticipated the cold weather in Minnesota but driving to work one morning, after hacking ice off the car windows, with a windchill of minus 78 degrees proved too much. Having braved the cold for one winter, I decided to high-tail it out of there.

Back to the warmth of the South in Tampa, I graduated with a master of science in management degree from the University of South Florida in 1970. For my master's project, I developed a simulation application for analyzing barge traffic from Tampa to ports along the Mississippi River. While I was working as an intern for a local company, the model proved useful, and managers were pleased with the results. The simulation utilized a package called the General-Purpose Simulation System (GPSS). This software was new, so none of my professors could help. I learned how it worked from incomplete manuals and trial and error. Since this was still the punch card and printout era, with slow turnarounds, I spent many evenings in the school computer center. However, there was a flaw in my analysis, which one of the managers caught in my final presentation. One of the data tables contained bad data, throwing off the final results a bit. The bad data was given to me, but I should have been more diligent in reviewing it. In projects to follow, I made sure someone on the team was as detail oriented as I was big picture oriented—I sought to ensure the team had the diversity of skills required to do the work and optimize team performance.

Esso Business Systems

In 1970, fresh out of graduate business school, I moved to Baytown, Texas, just east of Houston, to work in the Esso[6] refinery as a business systems analyst. But the job wasn't just analysis: It also included design, programming, testing, documentation, and—at crunch times—mainframe computer operator.

One leading-edge project I worked on would in the future be called a management information system (MIS). A study team consisting of young gung-ho, mostly MBA types did the analytical work to identify management's needs. John Fahlberg,[7] a colleague from that period, was on the team and reminded me of details in a 2022 call. John's first recollection was our regular Friday night retreat to the top-floor bar in the then-new Galleria Hotel, where we commiserated about long work hours and management, *of course*. When our report was presented and approved, I was a team of one who wrote the software system, which consisted of COBOL[8] and Mark IV (a high-level reporting language) code that extracted data from various operational systems and generated the new reports. The system was a big kluge, but it worked, and the executives benefited from the data. The information included barrels of refinery products produced, staff levels, maintenance activities, costs, and other financial analyses. It was the first time, at least in the Baytown location, data was extracted from multiple operating systems and consolidated for management. Previously, managers received transaction data from operating systems, but no cross-system data in a consolidated manner.

I was still new to programming, with no formal training, and I'm sure my COBOL programs were a maintenance nightmare. We did have a few pattern-like models, such as updating a primary file with transaction files—all of which contained serial data stored on magnetic tape. This was an era in which the last record in files had to contain all 9s as an end-of-file indicator.[9] The only interaction tools at the time were punch card input and printed reports. Turnaround for running and testing programs was usually overnight. It often took several iterations to get a clean compile, at 12 hours

6. Esso became Exxon while I worked there.
7. John went on to become the CFO at Target and CEO of several Silicon Valley start-ups.
8. COBOL: Common Business-Oriented Language.
9. Without the 9s record, file read processing would abort when it tried to read past the last record.

per try. In addition to programs, linking a series of programs together with needed data files and tape drives required knowing IBM's arcane Job Control Language (JCL).

> *"JCL is a clumsy and cumbersome system that is hard to learn, full of inconsistencies, and avoided by anyone with an iota of common sense and access to an alternative."*
>
> —Mainframes.com

Testing in those days was a trip. Testing tools were nonexistent. Once you had a clean compile, test data was developed and key-punched into cards, JCL was modified, and the test was run. Of course, if the file was greater than 80 characters, first you ran a program to combine multiple cards into an extended file format. Not surprisingly, many transaction files were 80 characters in length. If you were lucky, the test results were printed and analyzed. If you were unlucky—which was most of the time in the beginning—execution terminated, resulting in a core dump. This 144 characters per line printout of the entire computer memory in hexadecimal[10] looked like "01 A9 34 5A D2 88 88" and went on for page after page. Figuring out where your program started and then tracing the execution path was loads of fun!

In those days we had a vision of what management wanted, but the technology was severely limiting.

A Wild West Story

At Esso, my colleague and early mentor, Ed, was in his 60s. His desk, shelves, and floor overflowed with punch card decks. Instead of the stacked books of a university professor, every surface in his office was covered with computer printouts. Ed was responsible for maintaining accounting systems. He had a box of "secret" one-time "card" decks that he would tweak each year to complete year-end financial books. There was no backup for Ed. If he wasn't there, the books didn't close! None of us had any backup in those days. It really was the Wild West of IT.

10. The hexadecimal (hex) numbering system used in computing has 16 symbols (base 16) rather than the standard decimal (base 10) system. The 16 symbols are 0–9 and A–F.

I managed the implementation of a new accounting system scheduled to replace both the software application and the entire account coding system (another team developed the software). Because of the new coding system, the operational systems, such as payroll, would begin using new codes during the month. As a result, when we turned the old month-end system off, and switched the new one on, there was no going back. The project involved modifying and integrating many subsystems, which pushed the implementation to a nine-month project.

My innovation as the project manager was to build a test program that compared the outputs of the existing subsystems (payroll, accounts payable, cost allocation) with the outputs of the new or modified ones. My comparison program was complex, since it involved mappings for all the data in the operational systems feeding data to the accounting application. As different subsystems (for example, the maintenance cost system) became ready and started generating new codes, the testing program would map the new codes back to the old, and compare them with the correct mapping—and we would find tons of errors. It was my first insight into how critical, and hard, testing was.

As we neared our dreaded conversion date, working nights and weekends, we often helped the operators in the computer center, mounting tape drives, making card deck corrections on the fly, and rerunning the system. In subsequent years, developers were banned from operations due to audit "separation of duties" controls.

One evening toward the end of the project, the team returned from dinner, and I parked my car in a director's reserved spot. I worked all night and completely forgot about the parking. The next morning, one of my colleagues informed me I was in big trouble and needed to go see the director post haste. He was a gruff, traditional executive, so I was nervous as I entered his office, apologized, and told him my story of working all night. He surprised me by being gracious: "Anyone who works 24 hours straight gets to park anywhere they damn well please. Just leave your car where it is." He also thanked me and the team for our hard work.

Closing the accounting books for the month, even with the new system, took three nights.[11] The morning after the first day, things weren't looking good: The dreaded, and complex, cost accounting system, named BUPS

11. Computing resources were costly, so balancing loads over a 24-hour period was necessary. Therefore, jobs for many operational systems such as accounting were run overnight.

(Burden, Utilities, and Plant Services),[12] wasn't working correctly. There were only two or three people who knew this system well enough to troubleshoot it, so we took a small conference room to hash out solutions. My manager at the time was an old-school micromanager, and he walked into the room and started trying to "fix" things. Having worked two days with almost no sleep, I had little tolerance for his interference, but I did have enough sense not to confront him directly. I went to my friend, John Fahlberg, who was on my manager's level. "Get him out of there before I lose it," I tried to say somewhat civilly. John, being the suave guy he was, guided my manager out of the room and convinced him to let the team fix the problem—which we did shortly thereafter.

The project was a huge success, and everyone was relieved after nine months of 60- to 80-hour workweeks. My friend John planned a party for the team and their significant others. Ignoring his boss's suggestion to keep the cost of the party down, he put on a first-class event, replete with surf-and-turf entrees. Considering the enormous number of overtime hours put in by the project team, this was a minor concession.

While my title on this effort was Project Manager, I knew nothing about project management, except perhaps what a Gantt chart looked like. As with programming, I'd had no formal education and there was little material on the topic available. But I knew other team members had programming experience, knew their systems, and didn't need me micromanaging. We had a deadline, lots to do; the team just needed an overall game plan. It was the Wild West of project management, but I was starting to develop an inkling of management *style*.

During this period, most systems users were clueless, and most of us computer pioneers were marginally less clueless. A project with the refinery maintenance department provided a couple of clues. Managers there wanted a rudimentary system to keep track of maintenance tickets. Like a good analyst, I talked to several maintenance people who currently performed the task manually, wrote down a few specifications, and then developed and tested a system in Mark IV in about three months. I showed the various reports to the managers, which they liked, and then showed them the input forms

12. This was a state-of-the-art cost allocation system that allocated administrative costs, based on various factors such as people count, square footage of warehouse space, and many more, to production units, which were eventually reflected in product (e.g., gasoline, heating oil) costs. However, before allocation to the production units, admin costs were allocated among the admin groups—for example, accounting costs to IT, and the reverse, and around and around it went! This single system took 3–4 hours to run on our IBM 360 computer.

they would need to fill out and get keypunched. "Cease" was the response. "You mean we have to fill out these forms?" I tried to explain that producing reports required input data. They were not happy and ended up abandoning the effort. The knowledge on both sides about computers and how to make them effective was in its infancy.

The successful accounting system implementation led to my first management job—promotion to supervisor of an accounting group responsible for payroll, accounts payable, and materials accounting, among other areas. I was 28 years old, and the next youngest person in the group was 45, and they were all unionized. In this role, I learned the difference between being in IT and being on the business side of the IT–user interface. In IT, we were always asking for business users' time to learn about what they did, so we could build systems to support their efforts. Due to the long IT project timelines, there wasn't always daily stress—until the end, of course. On the business user side, there were daily stresses of making deadlines for payroll, accounting closes, and invoice payments. I've never forgotten the different dynamics of these interactions.

Once, when a staff member complained about a rude, disrespectful vendor demanding immediate payment (it wasn't overdue), I called the vendor vice president. I said if his staff were ever rude again, they would never do business with Exxon in the future! Of course, I had no such authority, but he didn't know that, and my staff loved it. This episode added another bit to my nascent management style—you treat everyone with equal respect.

In the early 1970s, there were seven Exxon refineries: four large and three smaller ones. The business systems group in the large ones had developed systems independently and the accounting system mentioned earlier was an initial step in instituting commonality. This reconciliation of differences wasn't easy, as each location had its own way of doing business and was reluctant to change.

To assist in rationalizing IT systems throughout the refining department, the position of business systems coordinator was established in the Refining Controller's office in Houston.[13] As in many companies at that time, IT reported to the controller. Later, as IT became an integral part of businesses, IT organizations would report to a chief information officer (CIO), who then reported to the chief executive officer (CEO). Accepting the coordinator's job,

13. Because the most important applications at that time were accounting and finance oriented, most IT departments reported to the controller (now we use the term chief financial officer [CFO]).

I worked to consolidate business systems, of which the accounting system was an excellent start. Consolidation of the software systems eventually led to consolidation of the refinery's computer facilities—a major accomplishment in those days.

Since we were in the controller's organization, one of my infrequent tasks was consolidating quarterly refinery financial reports. One day, I received a call from my counterpart at the corporate level who consolidated reports from all the divisions—refining, exploration, production: "Congratulations, your numbers are off by only a billion dollars." "Well," I said, "it was only one digit."

This was an influence job, not a managerial job, as the business systems supervisors in each refinery didn't work for me. Even so, I needed their help in planning how to bring the refinery IT systems into a modicum of commonality.

Exxon to Oglethorpe

In 1976, after nearly six years with Exxon, I moved to Atlanta, where, after a couple of short-term jobs, I ended up at Oglethorpe Power Company as software development manager.

During a brief stint in an Atlanta bank, where computer operations staff deposited piles of computer printouts on the president's desk each day, I worked on their first international banking system. This project included a trip to Citibank in New York City to investigate its state-of-the-art international banking system running on a DEC superminicomputer. Combining what I learned at Citibank with the knowledge I had gleaned in my discussions with our international bankers, I wrote a proposal for moving forward with new international banking systems for processing foreign exchange, letters of credit, and other international banking transactions. I liked working with international staff, as they were laid-back as bankers go.

But others were not so laid back, as I learned when confronted about my adherence to standards and procedures in a conservative bank. It seems my travel and proposal costs had exceeded my budget by more than the 10% limit. I hadn't even known I had a budget, but I received a dunning letter from a guy in accounting. I would have ignored it were it not for the five to six pages copied from an accounting book and stapled to the letter, which suggested I bone up on my cost accounting. I invited the guy to my office for a chat.

"What is that on my wall?" I asked, pointing to a framed document.

"It looks like a CPA certificate," he replied.

"Do you have one?" was my second question.

"No."

"Well, until you do, don't send me more pages from accounting books!"

Although my proposal on developing an international banking system was well received, by that time my nonconformist side conflicted with a banking career, so I quickly moved on to a less rigid culture.

OGLETHORPE POWER was a newly formed power generation, transmission, and distribution cooperative serving power companies in rural Georgia (in my undergraduate program, I majored in power systems engineering). My software development manager job was like a start-up position, as I got to build my staff and implement methods.

My department manager had worked for Andersen Consulting[14] in the past, so we adopted that firm's Method/1 methodology, which was advanced for the times. While Method/1 was intended as a project management tool for software projects, it didn't include specific development methods. For example, it contained tasks like "Define a file format" and "Complete a file layout form," but had no methods for actually defining them.

I had taken a few courses in structured techniques from Yourdon, Inc., and wanted to incorporate them into our company's development process. In a structured techniques discussion with one of the Andersen consultants, he suggested we look at the Warnier-Orr approach in addition to the Yourdon one. Subsequently, I brought Ken Orr in to teach us about his methodology and methods. While these methods were initially created by Frenchman Jean-Dominique Warnier, Ken Orr added to the original ideas and popularized them in the United States. The diagrams showed constructs such as hierarchy, sequence, repetition, and alternation. The methodology focused on starting with the outputs and working out the flows that produced those outputs. This emphasis on outputs rather than inputs was a concept that stuck with me, eventually extending to the concept of outcomes in the Agile era.

My staff embraced the Warnier-Orr approach, and we delivered several applications using it. We also purchased and used Ken's software package called Structure(s), a precursor to computer-aided software engineering (CASE) tools. One member of my staff got excited. An experienced

14. Arthur Andersen was primarily an accounting firm. The Andersen Consulting group grew and was eventually spun off as Accenture.

programmer, he remarked, "This is the first time I've ever written a COBOL program that didn't have a compiler error the first time through." This system was our first complete life-cycle use of the Warnier-Orr techniques. It was successful and well designed. However, it sowed a tiny seed of doubt in the back of my mind: "It sure took a longer development time than I expected."

During 1978, my writing career began with a published article, when "Solving Design Problems More Effectively" appeared in *Management Information Systems Quarterly*. Interestingly, this paper was not about software development, but rather about a process for group problem solving. During the 1970s and 1980s, I published other articles in *MISQ*, *Auerbach Reports*, *Datamation* (Highsmith, 1981), and *Business Software Review* (Highsmith, 1987).

Software development

In the early Wild West era, software processes, tools, reference books, and training were scarce. I learned from the IBM COBOL manual and rushed frequently into Ed's (from Esso) office to ask questions. Most of my knowledge acquisition came through experiences—both good and bad. Owing to his quiet style Ed was difficult to communicate with, but his experience made him an invaluable early mentor to me.

In the Exxon business group, there was a programmer who transferred in from technical systems. His technical programming had been in Fortran,[15] so he wrote his first business application, a payroll system, in COBOL, but used Fortran-like data names. Fortran programmers were accustomed to using data names like "EMPRT2" because of language restrictions,[16] rather than COBOL data names like "Employee-Pay-Rate2." His COBOL programs using Fortran data names caused untold maintenance headaches when he moved on. More Wild West.

One infamous, insidious, and even dangerous COBOL statement epitomizes the nature of this era—the ALTER statement. Think of a program statement—Go To CALC-Pay-Status. Okay, so far. Now comes the fun part.

15. Fortran (FORMula TRANslator) was an early computer language used for scientific and engineering applications.
16. Variable names in Fortran were limited to six digits, a–z and 0–9. In a large system, this limitation led to bizarre variable names. In addition, COBOL was a file-oriented language, designed for business systems. Fortran was a variable-oriented language designed for scientific and engineering calculations. IBM offered PL/1 as a one-language solution to replace both Fortran and COBOL, but it was not widely used.

Three pages down on the program printout (remember, only paper output in those days), an ALTER statement, based on some variable, modifies the initial GO TO destination to something like Go To ALT-CAL-PAY-STATUS. Wow. Now think of a COBOL program of 1,000 statements containing 50 of these ALTER-GOTO constructs and the difficulty in following the logic. You needed at least 25 fingers to keep up. Maintenance of these programs was a nightmare—usually passed along to the next poor soul.

During the time of serial tape files, prior to random-access databases, we used techniques like assigning a single data field multiple data types depending on a variable such as the record type. Field 4 might be used for "color" if the record type was "commercial" and for "size" if it was "retail." Date fields were often two digits, which precipitated the Y2K problem 30 years later. Why? Why would programmers create these maintenance nightmares?

Today, cell phones have 128 MB of memory and access to inexpensive terabytes of cloud data. The first Intel chip, introduced in 1971, had a clock speed of a little less than 1 megahertz.[17] Today, chip clock speeds can exceed 5 gigahertz.[18] In the Wild West era, computer speeds were glacial and memory was exorbitantly expensive. As programmers during this period, we needed to save every byte and hertz we could. We had to know which COBOL statements were fast and which were slow. This contributed to the overuse of ALTER statements, because they were fast.

During this period, development tools included flowcharts and hierarchical input–output (HIPO) diagrams. In the mid- to late 1970s, data flow diagrams and other structured methods emerged.[19]

In 1978 I read Tom DeMarco's (1978) book, *Structured Analysis and System Specification*, and attended a Yourdon class on structured analysis taught by Steve McMenamin.[20] I was an instant convert to this systematic approach to uncovering and documenting requirements. This "engineering" approach fueled my enthusiasm. When I accepted the job as software development manager at Oglethorpe, I knew incorporating these methods was part of my mission.

17. In this context, hertz is not a car rental company, but rather a frequency measure of cycles per second. One megahertz (MHz) equals 1 million Hz; 1 gigahertz equals 1,000 MHz or 1 million Hz.
18. In 2022, Oakridge National Labs supercomputer exceeded 1 petahertz (1×10^{15} Hz).
19. These diagrams are explained in Chapter 3.
20. More on these topics in Chapter 3.

Management trends

My pursuit of a management degree arose from curiosity and a growing sense I wanted a better understanding of management and leadership as a context for software development. General and project management trends shaped software development in the early years, as well as in following eras.

Inflexible cultures were the norm during the Wild West era—hierarchical, command-control, focused on planning and execution of those plans. Great strides were being made in engineering, and its assumed predictability crept into management thinking. Businesses were universally measured in financial terms—as always, driven by Wall Street. Software project success was measured by completion and cost. Just getting software delivered was considered a success, but schedule was also essential for projects. Cost was certainly important, but was secondary to getting systems up and operating.

Businesses operated on the premise—correct or not—that the world was nominally predictable and if plans failed to materialize, the problem was execution, not planning. Good managers and executives got things done—end of story. The nascent IT world was less predictable, which put IT executives in the hot seat because little allowance was given by general management for the still experimental nature of computers and software.

Looking at how management evolution impacted software development, four factors appeared important: industry evolution, work type, management style, and worker category.

As the industrial age blossomed in the early twentieth century, researchers like Frederick Winslow Taylor introduced the term *scientific management*, extolling the virtues of precise measurements and rigorous, prescriptive job duties. The view of the organization as machine became embedded in the management culture, and optimizing those machines became a key management goal.

Later, management theory began to change based on the work of individuals like Douglas McGregor and Peter Drucker. We hear about GOATs (greatest of all time) in various categories—but who would take the prize in literature? While it might depend on whose list you use, the general consensus is *In Search of Lost Time* by Marcel Proust.[21] If there is a GOAT in management theory, it could well be Peter Drucker. In his time, Drucker wrote 39 books and coined the term *knowledge work* in 1959. Called the father of modern management, he defined *management* as follows: "Management is

21. Depends on which Google list one concurs with.

a multi-purpose organ that manages business and manages managers and manages workers and work" (Drucker, 1954). This succinct definition helps us assess changes over time as work changes, workers change, managers change, and managers of managers change. Drucker's coining of the term *knowledge work* signaled that the very nature of work was changing.

"Organization as machine"—this imagery from our industrial past continues to cast a long shadow over management. Managers assumed stability was the normal situation and change was the "unusual state," writes Rita McGrath in a 2014 *Harvard Business Review* article. McGrath identifies three ages of management—execution, expertise, and empathy. "If organizations existed in the execution era to create scale and in the expertise era to provide advanced services, today many are looking to organizations to create complete and meaningful experiences" (McGrath, 2014). These management style categories bring another dimension to our discussion of the software development eras.[22]

Unfortunately, I never found further McGrath material other than her *Harvard Business Review* article. Moreover, there is debate about the empathetic style.[23] Even so, I liked the words McGrath used to categorize management periods. While the label command-control has often been applied to traditional management, none of the recent style names has emerged as "the" term. Names such as leadership-collaboration, adaptive leadership, Agile leadership, Management 3.0, savant leadership, and others have all appeared in the last two decades. So, I will nominate McGrath's "empathy" as the best name for modern management.

KLAUS SCHWAB, CEO of the World Economic Forum, proposed a way of looking at the evolution of work. Schwab's four ages are centered on the advances of science and technology:

- First: The Age of Mechanical Production
- Second: The Age of Science and Mass Production
- Third: The Digital Revolution
- Fourth: The Imagination Age

22. McGrath's styles are revisited in Chapter 8.
23. "Easily one of the most debated topics currently, the trend of increasing empathy in leadership has two very opinionated sides" (www.business.com).

As the Age of Science and Mass Production[24] got under way, organizations got bigger and needed a way to manage multilayered organizations, from ground-level supervisors to executives. Practices such as standardized processes, quality control, and specialization of labor were widely applied. Optimization—efficiency, consistency, measurability, predictability—was the goal. This approach, dubbed command-control management, defined the Execution Age. This was the age in which industrial workers were performing physical work.

With the Digital Revolution, computer technology evolved from mainframes to minicomputers to personal computers, broadening access to computing power. Concepts from other disciplines such as psychology and sociology began to creep into management theory, but this age primarily brought expertise into play, characterized by the concepts of reengineering, Six Sigma, and MBO.

Software development would add its own terms in this period—namely, *waterfall* and *Monumental Methodologies*. As the use of technology, including software, medicine, computers, materials, and computing devices, exploded, so did the need for knowledge workers. As knowledge work expanded, employees rebelled against existing manager–subordinate relationships, which drove early agilists to focus on building person-centric workplaces. In recognition of this change, *Adaptive Software Development* (Highsmith, 2000) used the term "leadership-collaboration" management, in contrast to the earlier "command-control," to characterize practices of this age.

Schwab doesn't set the time frame for the fourth industrial age, nor does he explicitly name it the Imagination Age, although there are references to imagination and innovation in his work. He defines this age by the *velocity* of change, the *breadth and depth* of change caused by the rapid evolution and integration of technology, and the *systems impact*, referring to international sociological systems. To prosper in this era, we will need to define "work" yet again, understand the differences between knowledge workers and innovation workers, and know how to lead, organize, and manage in an empathetic way that encourages imagination and creativity.

The Imagination Age is the period beyond the Digital Revolution, where creativity and imagination become the primary creators of economic value, as technologies such as artificial intelligence, biotechnology, robotics, quantum computing, and robotics become integrated into our world.

24. I didn't describe the First Age because it wasn't relevant to software development.

> *"We stand on the brink of a technological revolution that will fundamentally alter the way we live, work, and relate to one another. In its scale, scope, and complexity, the transformation will be unlike anything humankind has experienced before."*
>
> — Klaus Schwab, January 14, 2016

Eventually, workers were classified into three types: industrial, knowledge, and innovation. As the nature of work changed, the types of workers required changed, which in turn changed the way managers and executives (managers of managers) viewed and interacted with the workforce.

REMEMBER HOW YOU MIGHT have felt about the certainty of the future before the COVID-19 pandemic. And now? The ripple effects of the pandemic are unknown and largely unknowable until they fully play out. Many of these changes were emerging before 2020, and the pandemic just accelerated them. As uncertainty has increased, people have begun to theorize ways to model uncertainty and devise tools and methods to manage it.

Stephan H. Haeckel, who worked at the IBM Advanced Business Institute, published a *Harvard Business Review* article in 1993 and went on to further explain his ideas in his book *Adaptive Enterprise* in 1999 (Haeckel, 1999). His message: Organizations needed to move from a plan-and-execute to a sense-and-respond approach to the future. Sense-and-respond enables organizations to sense the outside world, respond quickly, and use feedback to initiate the next cycle. Organizations dedicated to plan-and-execute become so plan obsessed that deviations from the plans are considered mistakes rather than opportunities.

Why didn't Kodak respond to the digital camera threat? Did digital cameras appear overnight, or did Kodak miss market cues? Why did Netflix oust Blockbuster? Didn't the latter pick up on Netflix's rising market share of movie rentals? Sensing, in our fast-moving business and technology environment, can be extremely difficult. What is noise? When does accumulated noise raise to the level of alarm? In her latest book, *Seeing Around Corners: How to Spot Inflection Points in Business Before They Happen* (2019), Rita McGrath provides insight into this difficult question. In attempting to sort through and analyze streams of data, you need context—what arena are you playing in?

Dave Snowden devised a way to think about uncertainty in a context that supports decision making. In 1999, Snowden introduced the Cynefin model derived from his study of complexity theory. Snowden's model has

been embraced and widely used by the agile community. With each category of change, Snowden proposed a practice type to use. His model identifies five categories, or types, of change:

- Obvious, for which best practices suffice
- Complicated, for which good practices are used
- Complex, for which emergent practices are best
- Chaotic, which requires novel practices
- Disorder, for which practices might be unknown

As economies, businesses, and technologies evolved from somewhat complicated in the 1980s to complex, and then to chaotic in the 2000s, Snowden's framework helps us understand the role that combating uncertainty played in the transition from structured to agile development. In this book, I will use the Cynefin model as an indicator of strategic, high-level changes in the business and technology worlds. At the tactical, project, and product levels, I will introduce the exploration factor (EF) in Chapter 6. These two "methods"—Cynefin and EF—provide tools for managing uncertainty.

Table 2.1 summarizes the changes in these factors over the four software development eras and helps us understand why methods and methodologies evolved as they did. During my evolution from structured to agile methods, these frameworks helped me put useful context around my work.

Table 2.1 *Management and Work Evolution*

Key Factors and Thinkers				
Software Era	**Management Style (McGrath)**	**Work Type (Schwab)**	**Worker Category (Drucker)**	**Type of Change (Snowden)**
Wild West	Execution	Science and Mass Production	Industrial	Obvious/ complicated
Structured	Execution/ expertise	Digital Revolution	Knowledge	Complicated
Roots of Agile	Expertise	Digital Revolution	Knowledge	Complex
Agile	Empathy	Imagination	Innovation	Chaotic/disorder

TOWARD THE LATTER HALF of the Wild West era, I began to delve into project management practices. While project management had a long history, practices relevant to software development emerged only in the 1950s and 1960s. Gantt charts (task and schedule) were used successfully on projects such as the Hoover Dam in the early 1930s. Other large projects in these early years included the Manhattan Project to develop nuclear bombs in the 1940s. Bernard Shriever, while in the U.S. Air Force, was credited with originating the term *project management* in 1954.

The cornerstone of modern project management techniques was the Program Evaluation Technique (PERT), popularized by the Navy's successful use building Polaris submarines. PERT and Critical Path Method (CPM), invented in 1958 at Du Pont, began to be used in the U.S. aerospace, construction, and defense industries. The use of work breakdown structures (WBS) began in the early 1960s. The Project Management Institute (PMI) was founded in 1969 to do research into and promote project management practices. The most famous project undertaken in the 1960s was the Apollo Project (1963–1972), in which NASA successfully led six missions to explore the moon. Even though I had an exceedingly small part in the Apollo mission, this experience provided me with a happy quip: "My first project was a success."

Era observations

The 1960s and 1970s set the stage for subsequent eras in software development. Computer performance began to realize exponential improvements. Random-access storage devices multiplied. Core memory evolved from workers manually feeding wires through sets of tiny toroid "doughnuts." Person–computer interactions began their steady evolution.

Through the early years of this era, we might label software development as "ad hoc," but pioneers worked on early methods they envisioned turning into an engineering discipline. By the end of the era, structured methods and project management methodologies began to bring better organization and control to bear on the process of delivering working software; we might label them "advanced ad hoc." The next era would build on this base.

In the Wild West era, optimizing computer resources took precedence over optimizing people resources.[25] The costs of computer processing cycles, core memory, and external memory were enormous compared to those today.

25. Thanks to David Robinson, my *EDGE* book co-author, for this concept.

Hardware began its Moore's law[26] performance improvement march. In early years, computing power was expensive compared to personnel costs, which led to compromises, some of which caused problems for years (such as the Y2K issue). Today, in a world mired in the digital revolution, the situation has reversed: People costs are high compared to computer resources.

Although software development was in its infancy in the Wild West era, valuable solutions were delivered. Some of these systems, modified repeatedly, still exist today. Systems were primitive by today's standards, but they worked.

26. In 1965, Gordon Moore, the co-founder of Intel, observed what became known as Moore's law: "The number of transistors in an integrated circuit doubles about every two years."

This page intentionally left blank

Index