29782 00 ppi-xxii r4jm.ps 6/2/03 4:23 PM Page xv

Introduction

It looks obvious until you try it.

IEEE SOFTWARE[1]

My flight was waiting on the runway when the captain made an announcement. "We've had some trouble with the plane's air conditioning system. In a plane, the air conditioner controls the oxygen levels so we need to make sure it's working before we can take off. Restarting the air conditioning unit hasn't worked, so we're going to power down the aircraft and power it back on. *These modern airplanes are all computer controlled, you know, so they're not very reliable.*"

The pilot powered down the airplane, powered it back up—essentially, rebooted the airplane—and our flight continued without incident. Needless to say, I was especially glad to deplane at the end of that particular trip.

The Best of Times, the Worst of Times

The best software organizations control their projects to meet defined quality targets. They accurately predict software delivery dates months or years in advance. They deliver their software projects within budget, and their productivity is constantly improving. Their staff morale is high, and their customers are highly satisfied.

- A telecom company needed to change about 3,000 lines of code in a code base of about 1 million lines of code. They made their changes so carefully that a year later no errors had been found in operation.

29782 00 ppi-xxii r4jm.ps 6/2/03 4:23 PM Page xvi

Their total time to make the changes—including requirements analysis, design, construction, and testing—was *9 hours.*[2]

- A team developing software for the United States Air Force committed to a one-year schedule and a $2 million budget even though other credible bids for the project had run as high as two years and $10 million. When the team delivered the project one month early, the project manager said the team's success arose from using techniques that have been known for years but that are rarely used in practice.[3]
- An aerospace company develops software for companies on a fixed-price basis. Three percent of its projects overrun their budgets; ninety-seven out of a hundred meet their targets.[4]
- An organization that committed to achieving outstanding quality attained an average of 39 percent reduction in its post-release defect rate every year for a period of 9 years—a cumulative reduction of 99 percent.[5]

In addition to these notable successes, software pumps billions of dollars into the economy every year, both directly through sales of software itself and indirectly through improved efficiency and through creation of products and services that are made possible only with software's support.

The practices needed to create good software have been well established and readily available for 10 to 20 years or more. Despite some amazing triumphs, however, the software industry is not living up to its full potential. There is a wide gulf between the average practice and the best, and many of the practices in widespread use are seriously outdated and underpowered. Performance of the average software project leaves much to be desired, as many well-known disasters will attest.

- The IRS bumbled an $8 billion software modernization program that cost the United States taxpayers $50 billion per year in lost revenue.[6]
- The FAA's Advanced Automation System overran its planned budget by about $3 billion.[7]
- Problems with the baggage handling system caused a delay of more than a year in opening Denver International Airport. Estimates of the delay's cost ranged as high as $1.1 million per day.[8]

29782 00 ppi-xxii r4jm.ps 6/2/03 4:23 PM Page xvii

- The Ariane 5 rocket blew up on its maiden launch because of a software error.[9]
- The B-2 bomber wouldn't fly on its maiden flight because of a software problem.[10]
- Computer-controlled ferries in Seattle caused more than a dozen dock crashes, resulting in damage worth more than $7 million. The state of Washington recommended spending more than $3 million to change the ferries back to manual controls.[11]

Many projects that are lower profile than these are equally troubled. Roughly 25 percent of all projects fail outright,[12] and the typical project is 100 percent over budget at the point it's cancelled. Fifty percent of projects are delivered late, over budget, or with less functionality than desired.[13]

At the company level, these cancelled projects represent tremendous lost opportunity. If projects that are ultimately cancelled could be shut down at 10 percent of their intended budgets rather than 200 percent, imagine what a company could do by redirecting those resources at projects that were not ultimately cancelled.

At the national level, cancelled projects represent prodigious economic waste. A rough calculation suggests that cancelled software projects currently impose about a $40 billion drain on the United States economy.[14]

When projects succeed, they can still present risks to the public safety or welfare. A project lead at Lotus received a call from a surgeon who was using a spreadsheet to analyze patient data during open-heart surgery.[15] *Newsweek* magazine printed pictures of soldiers using Microsoft Excel on laptop computers to plan operations, and the Excel technical support team has received calls from the battlefield during active military operations.

The Purpose of This Book

Software development can be predictable, controllable, economical, and manageable. Software isn't usually developed that way, but it can be developed that way. This book is about the emerging profession of software

29782 00 ppi-xxii r4jm.ps 6/2/03 4:23 PM Page xviii

engineering—and professional software practices that support economical creation of high-quality software.

The essays in this book address questions like these:

- What is software engineering?
- How does software engineering relate to computer science?
- Why isn't regular computer programming good enough?
- Why do we need a *profession* of software engineering?
- Why is *engineering* the best model for a software development profession?
- In what ways do effective practices vary from project to project (or company to company), and in what ways are they usually the same?
- What can organizations do to support a professional approach to software development?
- What can individual software developers do to become full-fledged professionals?
- What can the software industry as a whole do to create a true profession of software engineering?

How This Book Is Organized

The parts in this book progress from looking at the trade of computer programming as it exists today to exploring the profession of software engineering as it might exist in the future.

Part 1, The Software Tar Pit, explains how the software field got to be the way it is. There are many valid reasons why the software field came to its current state. Understanding those reasons should be used to accelerate, not delay, the changes needed to make successful projects an everyday habit.

Part 2, Individual Professionalism, looks at the steps individuals can take on their own to achieve higher levels of software professionalism.

29782 00 ppi-xxii r4jm.ps 6/2/03 4:23 PM Page xix

Software projects are so complex that numerous key factors cannot be addressed effectively at the individual level. Part 3, Organizational Professionalism, digs into the organizational practices needed to support more professional software projects.

Part 4, Industry Professionalism, examines steps that must be taken by the software industry at large to support professionalism at the individual and organizational levels.

What I've Learned Since 1999

Professional Software Development is an updated and significantly expanded edition of my 1999 book, *After the Gold Rush.*[16] Since 1999, I've learned several lessons that are reflected in this new edition:

- Licensing of software developers is more controversial than I expected. I still think that licensing a small percentage of software engineers is an important step toward protecting the general public's safety and welfare. I have tried to clarify that licensing is only one of many initiatives needed to improve the software development profession, and not the most important one.
- Education of software engineers does not have to be tightly linked to licensing. Undergraduate and graduate educational programs can seek to instill an engineering mindset in software developers without necessarily preparing them to become licensed professional engineers. Indeed, if fewer than five percent of software developers are eventually licensed—which seems likely—targeting the majority of educational programs at licensing seems misguided.
- The world didn't fall apart on January 1, 2000. Although I didn't think Y2K would be catastrophic, I did believe that Y2K-related problems would be more significant than they were. The software industry's repair efforts turned out to be far more effective than I expected. Beyond that, the Y2K problem itself was in some sense a result of *successful* software development practices. Y2K would not

29782 00 ppi-xxii r4jm.ps 6/2/03 4:23 PM Page xx

have been an issue in the first place if so many software systems had not survived longer than their originally expected lifespans.

- Modern software development is truly impressive in many respects, and any comments about professionalizing the field of software development should account for software's numerous successes. We must be careful not to throw out the field's better practices as we try to strengthen the weaker ones.

Who Should Read This Book

If you develop software for a living, this book will explore what you need to do to become a truly professional software developer.

If you manage software projects, this book will summarize the differences between poorly run and well run software projects and overview what you can do to make your projects more successful.

If you manage a software organization, this book will outline the benefits available from systematic approaches to software development and sketch what you need to do to realize those benefits.

If you are a student who wants to work in the software field, this book will introduce you to the body of knowledge that makes up the field of software engineering and show you what a career in software engineering will look like.

Toward Professional Software Development

Industry researchers have long observed 10 to 1 differences in productivity between different organizations competing in the same industries.[17] More recently, researchers have observed differences as high as 600 to 1.[18] The most effective organizations are doing very well indeed.

The benefits of creating a true profession of software engineering are compelling. Traditional thinking would have it that change presents the greatest risk. In software's case, the greatest risk lies with not changing—staying mired in unhealthy, extravagant development practices instead

29782 00 ppi-xxii r4jm.ps 6/2/03 4:23 PM Page xxi

of switching to practices that were proven to be more effective many years ago.

How to change? That is the central topic of the rest of this book.

—Bellevue, Washington
Memorial Day, 2003

Notes

1. This is from a book review in *IEEE Software* about the book *Software Engineering Principles and Practice* by Hans van Vliet, West Sussex, England: John Wiley & Sons Ltd, 1993.
2. Pitterman, Bill, "Telcordia Technologies: The Journey to High Maturity," *IEEE Software,* July 2000.
3. Gibbs, W. Wayt, "Command and Control: Inside a Hollowed-Out Mountain, Software Fiascoes—and a Signal Success," *Scientific American,* August 1997, pp. 33–34. Tackett, Buford D., III, and Buddy Van Doren, "Process Control for Error Free Software: A Software Success Story," *IEEE Software,* May 1999.
4. Private communication with the author.
5. Herbsleb, James, et al., *Benefits of CMM Based Software Process Improvement: Initial Results,* Pittsburgh: Software Engineering Institute, Document CMU/SEI-94-TR-13, August 1994.
6. Anthes, Gary H., "IRS Project Failures Cost Taxpayers $50B Annually," *Computerworld,* October 14, 1996.
7. Britcher, Robert N., "Why (Some) Large Computer Projects Fail," in Glass, Robert L., *Software Runaways,* Englewood Cliffs, NJ: Prentice Hall, 1998. Gibbs, W. Wayt, "Software's Chronic Crisis," *Scientific American,* September 1994, pp. 86–95.
8. Glass, Robert L., *Software Runaways,* Englewood Cliffs, NJ: Prentice Hall, 1998. Gibbs, W. Wayt, "Software's Chronic Crisis," *Scientific American,* September 1994, pp. 86–95.
9. Nuseibeh, Bashar, "Ariane 5: Who Dunnit?" *IEEE Software,* May/June 1997, pp. 15–16.
10. Fishman, Charles, "They Write the Right Stuff," *Fast Company,* December 1996.
11. Neumann, Peter G., *Computer Related Risks,* Reading, MA: Addison-Wesley, 1995.
12. The Standish Group, "Charting the Seas of Information Technology," Dennis, MA: The Standish Group, 1994. Jones, Capers, *Patterns of Software Systems Failure and Success,* Boston, MA: International Thomson Computer Press, 1996.

29782 00 ppi-xxii r4jm.ps 6/2/03 4:23 PM Page xxii

13. The Standish Group, "Charting the Seas of Information Technology," Dennis, MA: The Standish Group, 1994.

14. This rough calculation is based on the employment figures presented in Table 7-2, "Job breakdown for software workers," on the job titles of computer and information scientists, research; computer programmers; computer software engineers, applications; computer software engineers, systems software; and computer systems analysts. Other job titles were not considered in this analysis. Total U.S. economic expenditure on software development was computed by multiplying an average fully burdened labor cost of $95,000 times 1,741,000 personnel in the job titles listed. Of the total amount of roughly $160 billion, 25% is the amount that is spent on cancelled projects. This analysis may understate the impact of cancelled projects due to the fact that the risk of cancellation increases with the size of the project, so that cancelled projects may be more costly than average projects.

15. Wiener, Lauren Ruth, *Digital Woes: Why We Should Not Depend on Software,* Reading, MA: Addison-Wesley, 1993.

16. McConnell, Steve, *After the Gold Rush,* Redmond, WA: Microsoft Press, 1999.

17. Mills, Harlan D., *Software Productivity,* Boston, MA: Little, Brown, 1983.

18. Yourdon, Edward, *Rise and Resurrection of the American Programmer,* Englewood Cliffs, NJ: Prentice Hall, 1996.