Extracted from:

Beyond Legacy Code

Nine Practices to Extend the Life (and Value) of Your Software

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Beyond Legacy Code

Nine Practices to Extend the Life (and Value) of Your Software

David Scott Bernstein

Foreword by Ward Cunningham

edited by Jacquelyn Carter

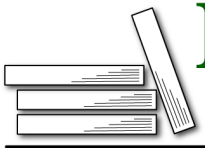
Beyond Legacy Code

Nine Practices to Extend the Life (and Value) of Your Software

David Scott Bernstein

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *https://pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-079-0
Encoded using the finest acid-free high-entropy binary digits.
Book version: B1.0—June 17, 2015

Introduction

This book will help you drop the cost of building and maintaining software.

If you're a software developer, you'll learn a set of practices to help you build changeable code, because when code is used it often needs to be changed. For managers working with software developers, this book will show you how investing in nine essential practices will help your team work more efficiently to deliver software that doesn't devolve into legacy code. And to do that, you need more than just a technical to do list—you need a firm understanding of the principles that add the *why* to the *how*.

Every day, we lose time, money, and opportunities because of legacy code.

Different people have different definitions for "legacy code," but put most simply, legacy code is code that, for a few different reasons, is particularly difficult to fix, enhance, and work with.

And there's a lot of it out there. Virtually all software that I've seen in production is legacy code.

The software industry as a whole hasn't put enough value on maintainability, so businesses wind up spending a great deal more to maintain code than they initially spent to write it. As we'll see in Chapter 2, *Out of CHAOS*, on page ?, inefficiencies in how software is built costs businesses at least tens of billions of dollars every year in the United States alone—and this is hardly just some abstract figure on a ledger sheet somewhere. We feel the effects of legacy code every day. Software is expensive, buggy, and hard to enhance.

People from inside and outside the industry have taken sides and argued for or against certain project management methodologies—a lot of which contain some truly brilliant ideas—but in order to affect lasting change for the better, we first have to come to a mutual understanding of the fundamental goals of software development.

This book isn't just about creating better software, it's about creating a better software industry. It includes the best of what I've learned in the last thirty

years as a professional developer. The first two decades of my career were spent doing traditional Waterfall software development where systems were planned, built, and tested in distinct phases. The problem was, the way we planned to build software turned out to be fraught with many unforeseen issues that forced us to make serious compromises in both quality and budget.

But over the last decade things have been changing for me, and for other software developers I know, who have been practicing an Agile software development methodology called Extreme Programming (XP). Instead of trying to figure everything out up front, we figure things out as we go, designing, building, and testing little bits of software at a time.

XP practices such as test-driven development and refactoring have taught me valuable lessons for decreasing both the risks and the costs of building and extending software. Using these practices has shown me a range of different approaches for solving software problems. Can applying these practices reveal ways of building higher quality, more maintainable software?

I say the answer is a resounding *Yes!*

Early in my career as a programmer, I got an assignment to reconcile stock data from the Standard and Poor's feed into my client's proprietary database. Up until then the process was done manually, which was error-prone and took, on average, fourteen hours each day to complete. I was asked to automate this process, but the best approach to accomplish this was not clear to me at first.

After a few weeks, and writing over forty pages of code, I had a flash of insight that involved reorganizing how I processed the data. Within a few hours I had finished the project and slashed out all but five pages of code. What I thought would take me several more months when I came into work that morning turned out to be finished before I left that evening. Since then I have had many flashes of insight that have revealed underlying patterns in problems that, once recognized, showed me how to rapidly build highly maintainable solutions.

This is but one example of dramatic differences in productivity between alternate ways of approaching the same problem. I've heard many similar stories from other developers. Perhaps you have your own stories of when you had a flash of insight and suddenly a difficult problem became simple.

In my professional experience, the difference between highly productive developers and average developers can be profound. I've spent most of my career studying those rare individuals who are many times more productive

than average software developers. What I've learned is that these people weren't born that way. They've simply formed some different distinctions than the rest of us and perhaps follow some unusual practices. All of these things are learnable skills.

As a young industry we're still figuring things out and learning to distinguish what's important from what's unimportant. Building software is very different from building physical things. Perhaps some of the challenges facing the software industry are rooted in a misconception of what software development actually is. In an effort to understand software development, to make it predictable, it has been compared to manufacturing and civil engineering. While there are similarities between software engineering and other fields of engineering, there are some fundamental differences that may not be obvious to someone who isn't actually writing software on a daily basis.

The fact that software engineering is not like other forms of engineering should really come as no surprise. Medicine is not like the law. Carpentry is not like baking. Software development is like one thing, and one thing only: software development. We need practices that make what we do more efficient, more verifiable and easier to change. If we can do this, we can slash the short-term cost of building software, and all but eliminate the crippling long-term cost of maintaining it.

To that end, I offer nine practices that come from the Agile methodologies of Extreme Programming, Scrum, and Lean. When not just adopted but fully understood, these nine practices can help prevent the code we write in the future from becoming legacy code.

And though there is an awful lot of code out there that's either impossible to fix or already slipping into obsolescence, we can use these same practices to slowly dig our way out of the mountain of legacy code we've already accumulated.

These nine practices will help development teams build better software, and help the industry as a whole stop leaking money, time, and resources.

I've seen these nine practices work for my clients, who build some of the biggest and most complex software ever created. I know it's possible to achieve extraordinary results using these practices, but just "using" them by no means guarantees success. We must understand the principles behind the practices in order to use them correctly.

These are interesting times, and we get to be part of them. And while we are pioneers venturing into uncharted territory, there are guiding lights. The nine

practices in this book have been guiding lights for me in my career as a software developer and beyond. I hope they become guiding lights for you as well.

How to Use this Book

Beyond Legacy Code: Nine Practices to Extend the Life (and Value) of Your Software is about developing software, but you don't have to be a software developer to understand it.

How software is written may be an alien concept to most people, yet it affects us all. Because it has become such a complex activity, developers often find themselves trying to explain concepts for which their customers, and even their managers, may have no point of reference. This book helps bridge that communications gap and explains technical concepts in common sense language to help us forge a common understanding about what good software development actually is.

Getting different kinds of readers on the same page with technical practices is no easy task, but to that end this book is designed to help five different groups of people share a common understanding of software development:

- software developers
- software development and IT managers
- software customers
- product and project managers from any industry
- and literally anyone interested in this vital technology

I've tried to make the world of software development accessible to everyone by borrowing elements of story structure, writing the book in the first person, and drawing on a wide range of stories, analogies, and metaphors to illuminate technical concepts. It can be hard to generalize about software development and easy to find exceptions to a lot of what I say but there's usually a deeper insight to be found.

To make this book accessible to non-developers and focus on the importance of these practices, it's not written as a *how-to* book. There are already several good books on everything from story writing to refactoring (see the bibliography). While this book does offer lots of practical advice, what makes it different and most valuable are the discussions of *why* the technical practices are useful. This approach helps non-developers, like our managers and stakeholders, understand some of the issues and challenges developers face when building software.

Part 1: The Legacy Code Crisis

In Part I, *The Legacy Code Crisis*, on page ?, I confront the significant issues facing the software industry head on and find that billions of dollars are lost every year due to poor software development processes.

Much of the software that runs our lives is buggy, brittle, and nearly impossible to extend, which is what we mean when we say "legacy code." How did we get here and what does that mean, not just in terms of the software industry but also in terms of all the other people and industries it touches?

If you're already familiar or perhaps even frustrated with the software industry you'll find more than just "preaching to the choir" in these pages. You'll also find a deeper insight into why things often don't work out as planned when building software, and lots of good reasons why better approaches are needed.

Even for software industry insiders, *Part 1* can put these significant challenges in their proper context. Managers and developers alike may discover a fresh perspective on the problems we, as an industry, are facing every day. As one manager said to me, "It added arguments to my arsenal." It may help you spread the message that at least we have to recognize that there is a problem before we can solve it.

For people coming to this book from outside the software industry in particular, *Part 1* may surprise you. In fact, I all but guarantee it will surprise, even shock you.

Part 2: Nine Practices for Building Maintainable Software

In Part II, *Nine Practices to Extend the Life (and Value) of Your Software*, on page ?, with the problem clearly stated, the remaining three quarters of the book moves past the doom and gloom and into a set of practices that provides a real, workable solution, beginning with practices that are most useful for managers.

In Chapter 5, *Practice 1: Say What, Why, and for Whom Before How*, on page ? and Chapter 6, *Practice 2: Build in Small Batches*, on page ?, I offer some hands-on recommendations for not just how better to begin implementing a complex software development process but to manage that process all the way through to completion. These two practices will be of particular interest to those of you coming from outside the software industry, with advice that can easily translate to any project management environment. By adopting these practices you can…

- operate more efficiently

- save money in both the short and long term
- create higher quality software
- increase customer satisfaction and repeat business

The next seven practices are much more software developer specific and present technical practices I found most helpful in my career.

I've seen these practices both succeed and fail. Software development teams have applied best practices but with poor technique so they didn't get the value they were hoping for. The difference between teams that are successful with these practices and teams that aren't comes down to understanding why these practices are important in the first place. That's what this book stresses.

And though these are ultimately technical practices, I urge managers—in any industry—to open your minds to the basic concepts. Know the challenges your developers face, and share with them the fact that there are practical, start-right-now practices that can move that team from foundering through broken processes to a new level of efficiency and effectiveness.

As you read through the descriptions of these practices, I urge you to think about why these practices are valuable before studying how to implement them on your project. This will help you learn how to use the practices more effectively.

Though I do recommend reading—and adopting—all nine practices, feel free to adopt the practices in any order you like. I certainly recognize that everyone coming to this book will have specific issues and specific needs, which is why I organized Part II as nine individual practices. Concentrate on what will help you most and help you fastest, but please don't stop there.

I'm Not Planting a Flag, I'm Opening a Door

How you read this book, and what you take from it, is up to you. I've tried to avoid circling the wagons around terms like Agile, Scrum, or XP. I want *Beyond Legacy Code: Nine Practices to Extend the Life (and Value) of Your Software* to change the way people think about the still-new profession of software development and help bring it into the mainstream. I want to open up discussion throughout the software community, where we too often take sides, argue details while missing the bigger picture, and otherwise dig ourselves into trenches when we should be sharing ideas based on a single common goal: to build the best possible software.

Online Resources

The code examples in this book can be found online at the Pragmatic Programmers web page for this book.[1] You'll also find a discussion forum where you can ask questions and receive feedback, as well as an errata submission form where issues with the text can be reported, plus a lot more.

1. http://pragprog.com/book/dblegacy