Working Effectively With Legacy Code Robert C Mart

Refactoring
Learning Test-Driven Development
Speed-Learning Agile Software Development
Making Software
Nine Practices to Extend the Life (and Value) of Your Software
The Mikado Method
The Pragmatic Programmer
your journey to mastery, 20th Anniversary Edition
Quality Code
Your Code as a Crime Scene
More Working Effectively with Legacy Code
Software Design X-Rays
Software Testing Principles, Practices, and Patterns
Improving the Design of Existing Code
WORK EFFECT LEG CODE _p1
Code Quality
Code Complete
Working Effectively with Legacy Code
Fix Technical Debt with Behavioral Code Analysis
A Code of Conduct for Professional Programmers
The Clean Coder
Rails 5 Test Prescriptions

Professionalism, Pragmatism, Pride
The Open Source Perspective
Patterns, Principles, and Practices of Domain-Driven Design
Effective Objective-C 2.0
The Art of Agile Development
What every programmer needs to know about cognition
Use Forensic Techniques to Arrest Defects, Bottlenecks, and Bad Design in Your Programs
Perl Medic
Java Concurrency in Practice
Release It!
The Software Craftsman
52 Specific Ways to Improve Your iOS and OS X Programs
Design and Deploy Production-Ready Software
The Programmer's Brain
The Legacy Code Programmer's Toolbox
with examples in C#
Re-Engineering Legacy Software
Unit Testing Principles, Practices, and Patterns

WELLS AVERY

Refactoring
Pearson Education

Test-Driven Development (TDD) is now an established technique for delivering better software faster. TDD is based on a simple idea: Write tests for your code before you write the code itself. However, this

"simple" idea takes skill and judgment to do well. Now there's a practical guide to TDD that takes you beyond the basic concepts. Drawing on a decade of experience building real-world systems, two TDD pioneers show how to let tests guide your development and “grow” software that is coherent, reliable, and maintainable. Steve Freeman and Nat Pryce describe the processes they use, the design principles they strive to achieve, and some of the tools that help them get the job done. Through an extended worked example, you’ll learn how TDD works at multiple levels, using tests to drive the features and the object-oriented structure of the code, and using Mock Objects to discover and then describe relationships between objects. Along the way, the book systematically addresses challenges that development teams encounter with TDD—from integrating TDD into your processes to testing your most difficult features. Coverage includes Implementing TDD effectively: getting started, and maintaining your momentum throughout the project Creating cleaner, more expressive,

more sustainable code Using tests to stay relentlessly focused on sustaining quality Understanding how TDD, Mock Objects, and Object-Oriented Design come together in the context of a real software development project Using Mock Objects to guide object-oriented designs Succeeding where TDD is difficult: managing complex test data, and testing persistence and concurrency **Learning Test-Driven Development** Pearson Education This deck of index cards is arranged in four sections: concepts, planning, teamwork and coding. The front of the card lists the things you need to know and the back provides further detail. **Speed-Learning Agile Software Development** Simon and Schuster Working Effectively with Legacy CodeWORK EFFECT LEG CODE _p1Prentice Hall Professional *Making Software* Addison-Wesley Summary The Mikado Method is a book written by the creators of this process. It describes a pragmatic, straightforward, and empirical method to plan and perform non-trivial technical improvements on an existing software system. The

method has simple rules, but the applicability is vast. As you read, you'll practice a step-by-step system for identifying the scope and nature of your technical debt, mapping the key dependencies, and determining the safest way to approach the "Mikado"—your goal. About the Technology The game "pick-up sticks" is a good metaphor for the Mikado Method. You eliminate "technical debt" —the legacy problems embedded in nearly every software system— by following a set of easy-to-implement rules. You carefully extract each intertwined dependency until you expose the central issue, without collapsing the project. About the Book The Mikado Method presents a pragmatic process to plan and perform nontrivial technical improvements on an existing software system. The book helps you practice a step-by-step system for identifying the scope and nature of your technical debt, mapping the key dependencies, and determining a safe way to approach the "Mikado"—your goal. A natural by-product of this process is the Mikado Graph, a roadmap that reflects deep understanding of how your system works.

This book builds on agile processes such as refactoring, TDD, and rapid feedback. It requires no special hardware or software and can be practiced by both small and large teams. Purchase of the print book includes a free eBook in PDF, Kindle, and ePub formats from Manning Publications. What's Inside Understand your technical debt Surface the dependencies in legacy systems Isolate and resolve core concerns while creating minimal disruption Create a roadmap for your changes About the Authors Ola Ellnestam and Daniel Brolund are developers, coaches, and team leaders. They developed the Mikado Method in response to years of experience resolving technical debt in complex legacy systems. Table of Contents PART 1 THE BASICS OF THE MIKADO METHOD Meet the Mikado Method Hello, Mikado Method! Goals, graphs, and guidelines Organizing your work PART 2 PRINCIPLES AND PATTERNS FOR IMPROVING SOFTWARE Breaking up a monolith Emergent design Common restructuring patterns Nine Practices to Extend the Life (and Value) of Your Software Pearson

Education Many claims are made about how certain tools, technologies, and practices improve software development. But which claims are verifiable, and which are merely wishful thinking? In this book, leading thinkers such as Steve McConnell, Barry Boehm, and Barbara Kitchenham offer essays that uncover the truth and unmask myths commonly held among the software development community. Their insights may surprise you. Are some programmers really ten times more productive than others? Does writing tests first help you develop better code faster? Can code metrics predict the number of bugs in a piece of software? Do design patterns actually make better software? What effect does personality have on pair programming? What matters more: how far apart people are geographically , or how far apart they are in the org chart? Contributors include: Jorge Aranda Tom Ball Victor R. Basili Andrew Begel Christian Bird Barry Boehm Marcelo Cataldo Steven Clarke Jason Cohen Robert DeLine Madeline Diep Hakan Erdogmus Michael Godfrey Mark Guzdial Jo E. Hannay Ahmed E. Hassan Israel Herraiz Kim Sebastian Herzig Cory

Kapser Barbara Kitchenham Andrew Ko Lucas Layman Steve McConnell Tim Menzies Gail Murphy Nachi Nagappan Thomas J. Ostrand Dewayne Perry Marian Petre Lutz Prechelt Rahul Premraj Forrest Shull Beth Simon Diomidis Spinellis Neil Thomas Walter Tichy Burak Turhan Elaine J. Weyuker Michele A. Whitecraft Laurie Williams Wendy M. Williams Andreas Zeller Thomas Zimmermann

The Mikado Method

Pearson Education

Users can dramatically improve the design, performance, and manageability of object-oriented code without altering its interfaces or behavior. "Refactoring" shows users exactly how to spot the best opportunities for refactoring and exactly how to do it, step by step.

The Pragmatic Programmer

Pearson Education

Summary The Art of Unit Testing, Second Edition guides you step by step from writing your first simple tests to developing robust test sets that are maintainable, readable, and trustworthy. You'll master the foundational ideas and quickly move to high-value subjects like mocks, stubs, and isolation, including frameworks such as Moq, FakeItEasy, and Typemock Isolator. You'll

explore test patterns and organization, working with legacy code, and even "untestable" code. Along the way, you'll learn about integration testing and techniques and tools for testing databases and other technologies. About this Book You know you should be unit testing, so why aren't you doing it? If you're new to unit testing, if you find unit testing tedious, or if you're just not getting enough payoff for the effort you put into it, keep reading. The Art of Unit Testing, Second Edition guides you step by step from writing your first simple unit tests to building complete test sets that are maintainable, readable, and trustworthy. You'll move quickly to more complicated subjects like mocks and stubs, while learning to use isolation (mocking) frameworks like Moq, FakeItEasy, and Typemock Isolator. You'll explore test patterns and organization, refactor code applications, and learn how to test "untestable" code. Along the way, you'll learn about integration testing and techniques for testing with databases. The examples in the book use C#, but will benefit anyone using a statically typed language such as Java or C++. Purchase of the print book includes a free eBook in PDF,

Kindle, and ePub formats from Manning Publications. What's Inside Create readable, maintainable, trustworthy tests Fakes, stubs, mock objects, and isolation (mocking) frameworks Simple dependency injection techniques Refactoring legacy code About the Author Roy Osherove has been coding for over 15 years, and he consults and trains teams worldwide on the gentle art of unit testing and test-driven development. His blog is at ArtOfUnitTesting.com. Table of Contents PART 1 GETTING STARTED The basics of unit testing A first unit test PART 2 CORE TECHNIQUES Using stubs to break dependencies Interaction testing using mock objects Isolation (mocking) frameworks Digging deeper into isolation frameworks PART 3 THE TEST CODE Test hierarchies and organization The pillars of good unit tests PART 4 DESIGN AND PROCESS Integrating unit testing into the organization Working with legacy code Design and testability your journey to mastery, 20th Anniversary Edition Pragmatic Bookshelf "After many decades - and even more methodologies - software projects are still failing. Why? Managers see software

development as a production line. Companies don't know how to manage software projects and hire good developers. Many developers still behave like factory workers, providing terrible service to their employers and clients. Agile was a big step forward, but not enough. What's missing? The right mindset - for both developers and their employers. As developers worldwide are recognizing, the right mindset is craftsmanship ... Mancuso explains what craftsmanship means to the developer and his or her organization, and shows how to live it every day in your real-world development environment. Mancuso shows how software craftsmanship fits with and helps you improve upon best-practice technical disciplines such as agile and lean, taking all your development projects to the next level. You'll learn how to change the disastrous perception that software developers are the same as factory workers, and that software projects can be run like factories. By placing greater professionalism, technical excellence, and customer satisfaction at the heart of what you do, you won't just deliver more value to everyone

involved: you'll be happier and more fulfilled doing it"--Publisher's description.
Quality Code Addison-Wesley
Does your Rails code suffer from bloat, brittleness, or inaccuracy? Cure these problems with the regular application of test-driven development. You'll use Rails 5.1, Minitest 5, and RSpec 3.6, as well as popular testing libraries such as factory_girl and Cucumber. Updates include Rails 5.1 system tests and Webpack integration. Do what the doctor ordered to make your applications feel all better. Side effects may include better code, fewer bugs, and happier developers.
Your Ruby on Rails application is sick. Deadlines are looming, but every time you make the slightest change to the code, something else breaks. Nobody remembers what that tricky piece of code was supposed to do, and nobody can tell what it actually does. Plus, it has bugs. You need test-driven development: a process for improving the design, maintainability, and long-term viability of software. With both practical code examples and discussion of why testing works, this book starts with the most basic features delivered as

part of core Ruby on Rails. Once you've integrated those features into your coding practice, work with popular third-party testing tools such as RSpec, Jasmine, Cucumber, and factory_girl. Test the component parts of a Rails application, including the back-end model logic and the front-end display logic. With Rails examples, use testing to enable your code to respond better to future change. Plus, see how to handle real-world testing situations. This new edition has been updated to Rails 5.1 and RSpec 3.6 and contains full coverage of new Rails features, including system tests and the Webpack-based JavaScript setup. What You Need: Ruby 2.4, Rails 5.1

Your Code as a Crime Scene

Prentice Hall Professional

Automated testing is a cornerstone of agile development. An effective testing strategy will deliver new functionality more aggressively, accelerate user feedback, and improve quality. However, for many developers, creating effective automated tests is a unique and unfamiliar challenge. xUnit Test Patterns is the definitive guide to writing automated

tests using xUnit, the most popular unit testing framework in use today. Agile coach and test automation expert Gerard Meszaros describes 68 proven patterns for making tests easier to write, understand, and maintain. He then shows you how to make them more robust and repeatable--and far more cost-effective. Loaded with information, this book feels like three books in one. The first part is a detailed tutorial on test automation that covers everything from test strategy to in-depth test coding. The second part, a catalog of 18 frequently encountered "test smells," provides trouble-shooting guidelines to help you determine the root cause of problems and the most applicable patterns. The third part contains detailed descriptions of each pattern, including refactoring instructions illustrated by extensive code samples in multiple programming languages.

More Working Effectively with Legacy Code

Pragmatic Bookshelf

As iOS apps become increasingly complex and business-critical, iOS developers must ensure consistently superior code quality. This means adopting best practices for creating and testing iOS apps. Test-

Driven Development (TDD) is one of the most powerful of these best practices. Test-Driven iOS Development is the first book 100% focused on helping you successfully implement TDD and unit testing in an iOS environment. Long-time iOS/Mac developer Graham Lee helps you rapidly integrate TDD into your existing processes using Apple's Xcode 4 and the OCUnit unit testing framework. He guides you through constructing an entire Objective-C iOS app in a test-driven manner, from initial specification to functional product. Lee also introduces powerful patterns for applying TDD in iOS development, and previews powerful automated testing capabilities that will soon arrive on the iOS platform. Coverage includes Understanding the purpose, benefits, and costs of unit testing in iOS environments Mastering the principles of TDD, and applying them in areas from app design to refactoring Writing usable, readable, and repeatable iOS unit tests Using OCUnit to set up your Xcode project for TDD Using domain analysis to identify the classes and interactions your app needs, and designing it accordingly Considering

third-party tools for iOS unit testing Building networking code in a test-driven manner Automating testing of view controller code that interacts with users Designing to interfaces, not implementations Testing concurrent code that typically runs in the background Applying TDD to existing apps Preparing for Behavior Driven Development (BDD) The only iOS-specific guide to TDD and unit testing, Test-Driven iOS Development covers both essential concepts and practical implementation.

Software Design X-Rays Pearson Education

We're losing tens of billions of dollars a year on broken software, and great new ideas such as agile development and Scrum don't always pay off. But there's hope. The nine software development practices in Beyond Legacy Code are designed to solve the problems facing our industry. Discover why these practices work, not just how they work, and dramatically increase the quality and maintainability of any software project. These nine practices could save the software industry. Beyond Legacy Code is filled with practical, hands-on advice and a common-

sense exploration of why technical practices such as refactoring and test-first development are critical to building maintainable software. Discover how to avoid the pitfalls teams encounter when adopting these practices, and how to dramatically reduce the risk associated with building software--realizing significant savings in both the short and long term. With a deeper understanding of the principles behind the practices, you'll build software that's easier and less costly to maintain and extend. By adopting these nine key technical practices, you'll learn to say what, why, and for whom before how; build in small batches; integrate continuously; collaborate; create CLEAN code; write the test first; specify behaviors with tests; implement the design last; and refactor legacy code. Software developers will find hands-on, pragmatic advice for writing higher quality, more maintainable, and bug-free code. Managers, customers, and product owners will gain deeper insight into vital processes. By moving beyond the old-fashioned procedural thinking of the Industrial Revolution, and working together to embrace standards and

practices that will advance software development, we can turn the legacy code crisis into a true Information Revolution.

Software Testing Principles, Practices, and Patterns

Simon and Schuster

A single dramatic software failure can cost a company millions of dollars - but can be avoided with simple changes to design and architecture. This new edition of the best-selling industry standard shows you how to create systems that run longer, with fewer failures, and recover better when bad things happen. New coverage includes DevOps, microservices, and cloud-native architecture. Stability antipatterns have grown to include systemic problems in large-scale systems. This is a must-have pragmatic guide to engineering for production systems. If you're a software developer, and you don't want to get alerts every night for the rest of your life, help is here. With a combination of case studies about huge losses - lost revenue, lost reputation, lost time, lost opportunity - and practical, down-to-earth advice that was all gained through painful experience, this book helps you avoid the

pitfalls that cost companies millions of dollars in downtime and reputation. Eighty percent of project life-cycle cost is in production, yet few books address this topic. This updated edition deals with the production of today's systems - larger, more complex, and heavily virtualized - and includes information on chaos engineering, the discipline of applying randomness and deliberate stress to reveal systematic problems. Build systems that survive the real world, avoid downtime, implement zero-downtime upgrades and continuous delivery, and make cloud-native applications resilient. Examine ways to architect, design, and build software - particularly distributed systems - that stands up to the typhoon winds of a flash mob, a Slashdotting, or a link on Reddit. Take a hard look at software that failed the test and find ways to make sure your software survives. To skip the pain and get the experience...get this book.

Improving the Design of Existing Code

"O'Reilly Media, Inc." Bring new power, performance, and scalability to your existing Perl code! Cure whatever ails your Perl code! Maintain, optimize, and scale any Perl software... whether you

wrote it or not Perl software engineering best practices for enterprise environments Includes case studies and code in a fun-to-read format Today's Perl developers spend 60-80% of their time working with existing Perl code. Now, there's a start-to-finish guide to understanding that code, maintaining it, updating it, and refactoring it for maximum performance and reliability. Peter J. Scott, lead author of Perl Debugged, has written the first systematic guide to Perl software engineering. Through extensive examples, he shows how to bring powerful discipline, consistency, and structure to any Perl program-new or old. You'll discover how to: Scale existing Perl code to serve larger network, Web, enterprise, or e-commerce applications Rewrite, restructure, and upgrade any Perl program for improved performance Bring standards and best practices to your entire library of Perl software Organize Perl code into modules and components that are easier to reuse Upgrade code written for earlier versions of Perl Write and execute better tests for your software...or anyone else's Use Perl in team-based, methodology-driven environments Document your Perl code more effectively and

efficiently If you've ever inherited Perl code that's hard to maintain, if you write Perl code others will read, if you want to write code that'll be easier for you to maintain, the book that comes to your rescue is Perl Medic. If you code in Perl, you need to read this book.–Adam Turoff, Technical Editor, The Perl Review. Perl Medic is more than a book. It is a well-crafted strategy for approaching, updating, and furthering the cause of inherited Perl programs.–Allen Wyke, co-author of several computer books including JavaScript Unleashed and Pure JavaScript. Scott's explanations of complex material are smooth and deceptively simple. He knows his subject matter and his craft-he makes it look easy. Scott remains relentless practical-even the 'Analysis' chapter is filled with code and tests to run.–Dan Livingston, author of several computer books including Advanced Flash 5: Actionscript in Action **WORK EFFECT LEG CODE _p1** "O'Reilly Media, Inc." Threads are a fundamental part of the Java platform. As multicore processors become the norm, using concurrency effectively becomes essential for building high-performance

applications. Java SE 5 and 6 are a huge step forward for the development of concurrent applications, with improvements to the Java Virtual Machine to support high-performance, highly scalable concurrent classes and a rich set of new concurrency building blocks. In Java Concurrency in Practice , the creators of these new facilities explain not only how they work and how to use them, but also the motivation and design patterns behind them. However, developing, testing, and debugging multithreaded programs can still be very difficult; it is all too easy to create concurrent programs that appear to work, but fail when it matters most: in production, under heavy load. Java Concurrency in Practice arms readers with both the theoretical underpinnings and concrete techniques for building reliable, scalable, maintainable concurrent applications. Rather than simply offering an inventory of concurrency APIs and mechanisms, it provides design rules, patterns, and mental models that make it easier to build concurrent programs that are both correct and performant. This book covers: Basic concepts of concurrency and thread safety Techniques for

building and composing thread-safe classes Using the concurrency building blocks in java.util.concurrent Performance optimization dos and don'ts Testing concurrent programs Advanced topics such as atomic variables, nonblocking algorithms, and the Java Memory Model *Code Quality* Pearson Education India

Are you working on a codebase where cost overruns, death marches, and heroic fights with legacy code monsters are the norm? Battle these adversaries with novel ways to identify and prioritize technical debt, based on behavioral data from how developers work with code. And that's just for starters. Because good code involves social design, as well as technical design, you can find surprising dependencies between people and code to resolve coordination bottlenecks among teams. Best of all, the techniques build on behavioral data that you already have: your version-control system. Join the fight for better code! Use statistics and data science to uncover both problematic code and the behavioral patterns of the developers who build your software. This combination gives you insights you

can't get from the code alone. Use these insights to prioritize refactoring needs, measure their effect, find implicit dependencies between different modules, and automatically create knowledge maps of your system based on actual code contributions. In a radical, much-needed change from common practice, guide organizational decisions with objective data by measuring how well your development teams align with the software architecture. Discover a comprehensive set of practical analysis techniques based on version-control data, where each point is illustrated with a case study from a real-world codebase. Because the techniques are language neutral, you can apply them to your own code no matter what programming language you use. Guide organizational decisions with objective data by measuring how well your development teams align with the software architecture. Apply research findings from social psychology to software development, ensuring you get the tools you need to coach your organization towards better code. If you're an experienced programmer, software architect, or technical manager, you'll get a

new perspective that will change how you work with code. What You Need: You don't have to install anything to follow along in the book. TThe case studies in the book use well-known open source projects hosted on GitHub. You'll use CodeScene, a free software analysis tool for open source projects, for the case studies. We also discuss alternative tooling options where they exist.

Code Complete Springer Science & Business Media

For those considering Extreme Programming, this book provides no-nonsense advice on agile planning, development, delivery, and management taken from the authors' many years of experience. While plenty of books address the what and why of agile development, very few offer the information users can apply directly.

Working Effectively with Legacy Code Pearson Education

Get more out of your legacy systems: more performance, functionality, reliability, and manageability Is your code easy to change? Can you get nearly instantaneous feedback when you do change it? Do you understand it? If the answer to any of these questions is no, you have legacy code, and it is

draining time and money away from your development efforts. In this book, Michael Feathers offers start-to-finish strategies for working more effectively with large, untested legacy code bases. This book draws on material Michael created for his renowned Object Mentor seminars: techniques Michael has used in mentoring to help hundreds of developers, technical managers, and testers bring their legacy systems under control. The topics covered include Understanding the mechanics of software change: adding features, fixing bugs, improving design, optimizing performance Getting legacy code into a test harness Writing tests that protect you against introducing new problems Techniques that can be used with any language or platform—with examples in Java, C++, C, and C# Accurately identifying where code changes need to be made Coping with legacy systems that aren't object-oriented Handling applications that don't seem to have any structure This book also includes a catalog of twenty-four dependency-breaking techniques that help you work with program elements in isolation and make safer changes.

Fix Technical

Debt with Behavioral Code Analysis Createspace Independent Publishing Platform Jack the Ripper and legacy codebases have more in common than you'd think. Inspired by forensic psychology methods, you'll learn strategies to predict the future of your codebase, assess refactoring direction, and understand how your team influences the design. With its unique blend of forensic psychology and code analysis, this book arms you with the strategies you need, no matter what programming language you use. Software is a living entity that's constantly changing. To understand software systems, we need to know where they came from and how they evolved. By mining commit data and analyzing the history of your code, you can start fixes ahead of time to eliminate broken designs, maintenance issues, and team productivity bottlenecks. In this book, you'll learn forensic psychology techniques to successfully maintain your software. You'll create a geographic profile from your commit data to find hotspots, and apply temporal coupling concepts to uncover hidden relationships between unrelated

areas in your code. You'll also measure the effectiveness of your code improvements. You'll learn how to apply these techniques on projects both large and small. For small projects, you'll get new insights into your design and how well the code fits your ideas. For large projects, you'll identify the good and the fragile parts. Large-scale development is also a social activity, and the team's dynamics influence code quality. That's why this book shows you how to uncover social biases when analyzing the evolution of your system. You'll use commit messages as eyewitness accounts to what is really happening in your code. Finally, you'll put it all together by tracking organizational problems in the code and finding out how to fix them. Come join the hunt for better code! What You Need: You need Java 6 and Python 2.7 to run the accompanying analysis tools. You also need Git to follow along with the examples.

A Code of Conduct for Professional Programmers Simon and Schuster Explains the importance of the test-driven environment in assuring quality while developing software, introducing patterns, principles, and techniques for testing any software system.

Best Sellers - Books :
• The Collector: A Novel By Daniel Silva
• The Seven Husbands Of Evelyn Hugo: A Novel By Taylor Jenkins Reid
• It Starts With Us: A Novel (2) (it Ends With Us)
• Never Never: A Romantic Suspense Novel Of Love And Fate By Colleen Hoover
• The Woman In Me By Britney Spears
• The Summer I Turned Pretty (summer I Turned Pretty, The)
• Demon Copperhead: A Pulitzer Prize Winner By Barbara Kingsolver
• Love You Forever By Robert Munsch
• The Silent Patient
• Our Class Is A Family (our Class Is A Family & Our School Is A Family)