Refactoring Improving The Design Of Existing Code Object Technology Series

Learn the principles of good software design,

and how to turn those principles into great code. This book introduces you to software engineering — from the application of engineering principles to the development of software. You'll see how to run a software development project, examine the different phases of a project, and learn how to design and

implement programs that solve specific problems. It's also about code construction — how to write great programs and make them work. Whether you're new to programming or have written hundreds of applications, in this book you'll re-examine what you already do, and you'll investigate ways to improve. Using

the Java language, you'll look deeply into coding standards, debugging, unit testing, modularity, and other characteristics of good programs. With Software Development, Design and Coding, author and professor John Dooley distills his years of teaching and development experience to demonstrate practical techniques for great

coding. What You'll Learn Review modern agile methodologies including Scrum and Lean programming Leverage the capabilities of modern computer systems with parallel programming Work with design patterns to exploit application development best practices Use modern tools for

development, collaboration, and source code controls

Who This Book Is For

Early career software developers, or upper-level students in software engineering courses

Awareness of design smells – indicators of common design problems – helps developers or software

engineers understand mistakes made while designing, what design principles were overlooked or misapplied, and what principles need to be applied properly to address those smells through refactoring. Developers and software engineers may "know" principles and patterns, but are not

aware of the "smells" that exist in their design because of wrong or mis-application of principles or patterns. These smells tend to contribute heavily to technical debt – further time owed to fix projects thought to be complete – and need to be addressed via proper refactoring. Refactoring for Software Design Smells

presents 25 structural design smells, their role in identifying design issues, and potential refactoring solutions. Organized across common areas of software design, each smell is presented with diagrams and examples illustrating the poor design practices and the problems that result, creating a catalog of

nuggets of readily usable information that developers or engineers can apply in their projects. The authors distill their research and experience as consultants and trainers, providing insights that have been used to improve refactoring and reduce the time and costs of managing software projects. Along

the way they recount anecdotes from actual projects on which the relevant smell helped address a design issue. Contains a comprehensive catalog of 25 structural design smells (organized around four fundamental design principles) that contribute to technical debt in software projects

Presents a unique naming scheme for smells that helps understand the cause of a smell as well as points toward its potential refactoring Includes illustrative examples that showcase the poor design practices underlying a smell and the problems that result Covers pragmatic techniques for

refactoring design smells to manage technical debt and to create and maintain high-quality software in practice Presents insightful anecdotes and case studies drawn from the trenches of real-world projects .NET 2.0 IL (Intermediate Language) is the foundation language at the root of

all the .NET languages. It is this code which is compiled and executed by the .NET 2.0 Framework. As a result of this absolutely anything that can be expressed in IL can be carried out by the .NET 2.0 Framework. This book gives readers inside information on the language's architecture straight

from the most reliable possible source – Serge Lidin, the language's designer.
Radically improve your testing practice and software quality with new testing styles, good patterns, and reliable automation. Key Features A practical and results-driven approach to unit testing Refine your existing unit tests

by implementing modern best practices Learn the four pillars of a good unit test Safely automate your testing process to save time and money Spot which tests need refactoring, and which need to be deleted entirely Purchase of the print book includes a free eBook in PDF, Kindle, and ePub formats from

Manning Publications. About The Book Great testing practices maximize your project quality and delivery speed by identifying bad code early in the development process. Wrong tests will break your code, multiply bugs, and increase time and costs. You owe it to yourself—and your projects—to learn how to

do excellent unit testing. Unit Testing Principles, Patterns and Practices teaches you to design and write tests that target key areas of your code including the domain model. In this clearly written guide, you learn to develop professional-quality tests and test suites and integrate testing throughout the

application life cycle. As you adopt a testing mindset, you'll be amazed at how better tests cause you to write better code. What You Will Learn Universal guidelines to assess any unit test Testing to identify and avoid anti-patterns Refactoring tests along with the production code Using integration tests to

verify the whole system This Book Is Written For For readers who know the basics of unit testing. Examples are written in C# and can easily be applied to any language. About the Author Vladimir Khorikov is an author, blogger, and Microsoft MVP. He has mentored numerous teams on the ins and outs of unit

testing. Table of Contents: PART 1 THE BIGGER PICTURE 1 ¦ The goal of unit testing 2 ¦ What is a unit test? 3 ¦ The anatomy of a unit test PART 2 MAKING YOUR TESTS WORK FOR YOU 4 ¦ The four pillars of a good unit test 5 ¦ Mocks and test fragility 6 ¦ Styles of unit testing 7 ¦ Refactoring toward

valuable unit tests
PART 3
INTEGRATION
TESTING 8 ¦ Why
integration testing? 9 ¦
Mocking best practices
10 ¦ Testing the database
PART 4 UNIT
TESTING ANTI-
PATTERNS 11 ¦ Unit
testing anti-patterns
Refactoring Workbook
UML Distilled
Evolutionary Database

Design (paperback)
Refactoring to Patterns
JUnit Pocket Guide
The Rust Programming Language (Covers Rust 2018)
Systems programming provides the foundation for the world's computation. Writing performance-sensitive

code requires a programming language that puts programmers in control of how memory, processor time, and other system resources are used. The Rust systems programming language combines that

control with a modern type system that catches broad classes of common mistakes, from memory management errors to data races between threads. With this practical guide, experienced

systems programmers will learn how to successfully bridge the gap between performance and safety using Rust. Jim Blandy, Jason Orendorff, and Leonora Tindall demonstrate how Rust's features

put programmers in control over memory consumption and processor use by combining predictable performance with memory safety and trustworthy concurrency. You'll learn: Rust's fundamental data

types and the core concepts of ownership and borrowing How to write flexible, efficient code with traits and generics How to write fast, multithreaded code without data races Rust's key power tools: closures,

iterators, and asynchronous programming Collections, strings and text, input and output, macros, unsafe code, and foreign function interfaces This revised, updated edition covers the Rust 2021 Edition.

Brimming with over 100 "recipes" for getting down to business and actually doing XP, the Java Extreme Programming Cookbook doesn't try to "sell" you on XP; it succinctly documents the

most important features of popular open source tools for XP in Java--including Ant, Junit, Http'nit, Cactus, Tomcat, XDoclet--and then digs right in, providing recipes for implementing the

tools in real-world environments. 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 depe ndency-breaking

techniques that help you work with program elements in isolation and make safer changes. Presents a process called "prefactoring," the premise of which states that you're better off

considering the best possible design patterns before you even begin your projects. This book presents prefactoring guidelines in design, code, and testing, derived from lessons learned by many

developers over the years.
Extreme Abstraction -
Extreme Separation -
Extreme Reliability
Programming Rust
Managing Technical Debt
Prefactoring
When Life Gives You Demons

WORK EFFECT LEG CODE _p1

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.
You know good software when you see it, but how do you explain what good software is? Experienced software developers have pet practices and techniques that

make their software easier to test, maintain and understand. But when you ask them how to make your software like theirs, they give you a seemingly endless list of rules. How can they remember all those rules? The

secret is that they don't! Instead, experienced software developers understand a handful of basic principles. The rules are merely manifestations of these basic principles. But, principles are hard

to explain; so experienced developers resort to explaining rules instead.In Principle-Based Refactoring, Halladay explains a set of software refactoring rules and links the refactoring rules back to the basic

principles that drive robust software design. The book identifies eight fundamental design principles and also includes a set of approximately fifty refactoring rules that illustrate the

principles. Each rule has a summary description, a discussion, including references back to the driving principles, and examples of the rules' applications. In addition, this book discusses

refactoring mechanics including test strategies that guide software developers in verifying the quality of refactored code. When carefully selected and used, Domain-Specific Languages

(DSLs) may simplify complex code, promote effective communication with customers, improve productivity, and unclog development bottlenecks. In Domain-Specific Languages , noted

software development expert Martin Fowler first provides the information software professionals need to decide if and when to utilize DSLs. Then, where DSLs prove

suitable, Fowler presents effective techniques for building them, and guides software engineers in choosing the right approaches for their applications. This book ' s techniques may be utilized with most modern object-

oriented languages; the author provides numerous examples in Java and C#, as well as selected examples in Ruby. Wherever possible, chapters are organized to be self-standing, and most

reference topics are presented in a familiar patterns format. Armed with this wide-ranging book, developers will have the knowledge they need to make important decisions about DSLs—and, where

appropriate, gain the significant technical and business benefits they offer. The topics covered include: How DSLs compare to frameworks and libraries, and when those alternatives are sufficient Using

parsers and parser generators, and parsing external DSLs Understanding, comparing, and choosing DSL language constructs Determining whether to use code generation, and comparing

code generation strategies Previewing new language workbench tools for creating DSLs “ One of the most significant books in my life. ” – Obie Fernandez, Author, The Rails Way “ Twenty years ago, the

first edition of The Pragmatic Programmer completely changed the trajectory of my career. This new edition could do the same for yours.” – Mike Cohn, Author of Succeeding with Agile, Agile

Estimating and Planning, and User Stories Applied “ . . . filled with practical advice, both technical and professional, that will serve you and your projects well for years to come. ” – Andrea Goulet, CEO, Corgibytes,

Founder, LegacyCode.Rocks “. . . lightning does strike twice, and this book is proof.” – VM (Vicky) Brasseur, Director of Open Source Strategy, Juniper Networks The Pragmatic Programmer is one of those rare

tech books you ' ll read, re-read, and read again over the years. Whether you ' re new to the field or an experienced practitioner, you ' ll come away with fresh insights each and every time. Dave Thomas and Andy

Hunt wrote the first edition of this influential book in 1999 to help their clients create better software and rediscover the joy of coding. These lessons have helped a generation of programmers examine the very

essence of software development, independent of any particular language, framework, or methodology, and the Pragmatic philosophy has spawned hundreds of books, screencasts, and

audio books, as well as thousands of careers and success stories. Now, twenty years later, this new edition re-examines what it means to be a modern programmer. Topics range from personal

responsibility and career development to architectural techniques for keeping your code flexible and easy to adapt and reuse. Read this book, and you ' ll learn how to: Fight software rot Learn

continuously Avoid the trap of duplicating knowledge Write flexible, dynamic, and adaptable code Harness the power of basic tools Avoid programming by coincidence Learn real requirements Solve the

underlying problems of concurrent code Guard against security vulnerabilities Build teams of Pragmatic Programmers Take responsibility for your work and career Test

ruthlessly and effectively, including property-based testing Implement the Pragmatic Starter Kit Delight your users Written as a series of self-contained sections and filled with classic and fresh anecdotes,

thoughtful examples, and interesting analogies, The Pragmatic Programmer illustrates the best approaches and major pitfalls of many different aspects of software development.

Whether you ’ re a new coder, an experienced programmer, or a manager responsible for software projects, use these lessons daily, and you ’ ll quickly see improvements in personal productivity,

accuracy, and job satisfaction. You ’ ll learn skills and develop habits and attitudes that form the foundation for long-term success in your career. You ’ ll become a Pragmatic Programmer. Register your book for

convenient access to downloads, updates, and/or corrections as they become available. See inside book for details.
Quick Look-up and Advice
JAVA CONCURRENCY PRACT _p1

Practical Object-Oriented Design
Java Extreme Programming Cookbook
Domain-Specific Languages
A Brief Guide to the Standard Object Modeling Language

As the application of object technology--p

articularly the Java programming language--has become commonplace, a new problem has emerged to confront the software development community. Significant numbers of poorly designed programs have

been created by less-experienced developers, resulting in applications that are inefficient and hard to maintain and extend. Increasingly, software system professionals are discovering just how difficult it is to work

with these inherited, "non-optimal" applications. For several years, expert-level object programmers have employed a growing collection of techniques to improve the structural integrity and performance of such existing

software programs. Referred to as "refactoring," these practices have remained in the domain of experts because no attempt has been made to transcribe the lore into a form that all developers could use. . .until now. In Refactoring:

Improving the Design of Existing Code, renowned object technology mentor Martin Fowler breaks new ground, demystifying these master practices and demonstrating how software practitioners can realize the

significant benefits of this new process. With proper training a skilled system designer can take a bad design and rework it into well-designed, robust code. In this book, Martin Fowler shows you where opportunities for refactoring typically

can be found, and how to go about reworking a bad design into a good one. Each refactoring step is simple--seemingly too simple to be worth doing. Refactoring may involve moving a field from one class to another, or pulling

some code out of a method to turn it into its own method, or even pushing some code up or down a hierarchy. While these individual steps may seem elementary, the cumulative effect of such small changes can radically improve the design.

Refactoring is a proven way to prevent software decay. In addition to discussing the various techniques of refactoring, the author provides a detailed catalog of more than seventy proven refactorings with helpful pointers that teach you when

to apply them; step-by-step instructions for applying each refactoring; and an example illustrating how the refactoring works. The illustrative examples are written in Java, but the ideas are applicable to any object-oriented programming

language.
Like any other software system, Web sites gradually accumulate “cruft” over time. They slow down. Links break. Security and compatibility problems mysteriously appear. New features don’t

integrate seamlessly. Things just don’t work as well. In an ideal world, you’d rebuild from scratch. But you can’t: there’s no time or money for that. Fortunately, there’s a solution: You can refactor your Web code using easy, proven

techniques, tools, and recipes adapted from the world of software development. In Refactoring HTML, Elliotte Rusty Harold explains how to use refactoring to improve virtually any Web site or application. Writing for programmers

and non-programmers alike, Harold shows how to refactor for better reliability, performance, usability, security, accessibility, compatibility, and even search engine placement. Step by step, he shows how to migrate obsolete

code to today's stable Web standards, including XHTML, CSS, and REST—and eliminate chronic problems like presentation-based markup, stateful applications, and "tag soup." The book's extensive catalog of detailed refactorings and

practical “recipes for success” are organized to help you find specific solutions fast, and get maximum benefit for minimum effort. Using this book, you can quickly improve site performance now—and make your site far easier to

enhance, maintain, and scale for years to come. Topics covered include • Recognizing the “smells” of Web code that should be refactored • Transforming old HTML into well-formed, valid XHTML, one step at a time •

Modernizing existing layouts with CSS • Updating old Web applications: replacing POST with GET, replacing old contact forms, and refactoring JavaScript • Systematically refactoring content and links • Restructuring sites

without changing the URLs your users rely upon This book will be an indispensable resource for Web designers, developers, project managers, and anyone who maintains or updates existing sites. It will be

especially helpful to Web professionals who learned HTML years ago, and want to refresh their knowledge with today's standards-compliant best practices. This book will be an indispensable resource for Web designers,

developers, project managers, and anyone who maintains or updates existing sites. It will be especially helpful to Web professionals who learned HTML years ago, and want to refresh their knowledge with today's standards-

compliant best practices.
The practice of enterprise application development has benefited from the emergence of many new enabling technologies. Multi-tiered object-oriented platforms, such as Java and

.NET, have become commonplace. These new tools and technologies are capable of building powerful applications, but they are not easily implemented. Common failures in enterprise applications often occur because their

developers do not understand the architectural lessons that experienced object developers have learned. Patterns of Enterprise Application Architecture is written in direct response to the stiff challenges that face

enterprise application developers. The author, noted object-oriented designer Martin Fowler, noticed that despite changes in technology--from Smalltalk to CORBA to Java to .NET--the same basic design ideas can be

adapted and applied to solve common problems. With the help of an expert group of contributors, Martin distills over forty recurring solutions into patterns. The result is an indispensable handbook of solutions that are

applicable to any enterprise application platform. This book is actually two books in one. The first section is a short tutorial on developing enterprise applications, which you can read from start to finish to understand the

scope of the book's lessons. The next section, the bulk of the book, is a detailed reference to the patterns themselves. Each pattern provides usage and implementation information, as well as detailed code examples in Java or

C#. The entire book is also richly illustrated with UML diagrams to further explain the concepts. Armed with this book, you will have the knowledge necessary to make important architectural decisions about

building an enterprise application and the proven patterns for use when building them. The topics covered include · Dividing an enterprise application into layers · The major approaches to organizing business

logic · An in-depth treatment of mapping between objects and relational databases · Using Model-View-Controller to organize a Web presentation · Handling concurrency for data that spans multiple transactions ·

Designing distributed object interfaces
The official book on the Rust programming language, written by the Rust development team at the Mozilla Foundation, fully updated for Rust 2018. The Rust

Programming Language is the official book on Rust: an open source systems programming language that helps you write faster, more reliable software. Rust offers control over low-level details (such as memory

usage) in combination with high-level ergonomics, eliminating the hassle traditionally associated with low-level languages. The authors of The Rust Programming Language, members of the Rust Core Team,

share their knowledge and experience to show you how to take full advantage of Rust's features--from installation to creating robust and scalable programs. You'll begin with basics like creating functions, choosing data types, and

binding variables and then move on to more advanced concepts, such as: • Ownership and borrowing, lifetimes, and traits • Using Rust's memory safety guarantees to build fast, safe programs • Testing, error handling, and effective refactoring

• Generics, smart pointers, multithreading, trait objects, and advanced pattern matching • Using Cargo, Rust's built-in package manager, to build, test, and document your code and manage dependencies •

How best to use Rust's advanced compiler with compiler-led programming techniques You'll find plenty of code examples throughout the book, as well as three chapters dedicated to building complete

projects to test your learning: a number guessing game, a Rust implementation of a command line tool, and a multithreaded server. New to this edition: An extended section on Rust macros, an expanded chapter on modules, and

appendixes on Rust development tools and editions.
Expert .NET 2.0 IL Assembler
Refactoring
Pro PHP Refactoring
Principle-Based Refactoring
Pattern Enterpr Applica Arch
An Agile Primer

Using Ruby

The expert guide to building Ruby on Rails applications Ruby on Rails strips complexity from the development process, enabling professional developers to focus on what matters most: delivering business value. Now, for the first time, there ' s a

comprehensive, authoritative guide to building production-quality software with Rails. Pioneering Rails developer Obie Fernandez and a team of experts illuminate the entire Rails API, along with the Ruby idioms, design approaches, libraries, and plug-ins that make

Rails so valuable. Drawing on their unsurpassed experience, they address the real challenges development teams face, showing how to use Rails ' tools and best practices to maximize productivity and build polished applications users will

enjoy. Using detailed code examples, Obie systematically covers Rails ' key capabilities and subsystems. He presents advanced programming techniques, introduces open source libraries that facilitate easy Rails adoption, and offers important insights into testing and production

deployment. Dive deep into the Rails codebase together, discovering why Rails behaves as it does— and how to make it behave the way you want it to. This book will help you Increase your productivity as a web developer Realize the overall joy of programming with

Ruby on Rails Learn what ’ s new in Rails 2.0 Drive design and protect long-term maintainability with TestUnit and RSpec Understand and manage complex program flow in Rails controllers Leverage Rails ’ support for designing REST-compliant APIs Master

sophisticated Rails routing concepts and techniques Examine and troubleshoot Rails routing Make the most of ActiveRecord object-relational mapping Utilize Ajax within your Rails applications Incorporate logins and authentication into your application Extend Rails with the

best third-party plug-ins and write your own
Integrate email services into your applications with ActionMailer
Choose the right Rails production configurations
Streamline deployment with Capistrano
Learn algorithms for solving classic computer science

problems with this concise guide covering everything from fundamental algorithms, such as sorting and searching, to modern algorithms used in machine learning and cryptography Key FeaturesLearn the techniques you need to know to design

algorithms for solving complex problemsBecome familiar with neural networks and deep learning techniquesExplore different types of algorithms and choose the right data structures for their optimal implementationBook Description

Algorithms have always played an important role in both the science and practice of computing. Beyond traditional computing, the ability to use algorithms to solve real-world problems is an important skill that any developer or programmer must have. This book will

help you not only to develop the skills to select and use an algorithm to solve real-world problems but also to understand how it works. You ’ ll start with an introduction to algorithms and discover various algorithm design techniques, before exploring how to

implement different types of algorithms, such as searching and sorting, with the help of practical examples. As you advance to a more complex set of algorithms, you'll learn about linear programming, page ranking, and graphs, and even work with machine learning

algorithms, understanding the math and logic behind them. Further on, case studies such as weather prediction, tweet clustering, and movie recommendation engines will show you how to apply these algorithms optimally. Finally, you ’ ll become well versed in

techniques that enable parallel processing, giving you the ability to use these algorithms for compute-intensive tasks. By the end of this book, you'll have become adept at solving real-world computational problems by using a wide range of algorithms. What you

will learnExplore existing data structures and algorithms found in Python librariesImplement graph algorithms for fraud detection using network analysisWork with machine learning algorithms to cluster similar tweets and process Twitter data in real timePredict the

weather using supervised learning algorithmsUse neural networks for object detectionCreate a recommendation engine that suggests relevant movies to subscribersImplement foolproof security using symmetric and asymmetric encryption on Google Cloud

Platform (GCP)Who this book is for This book is for programmers or developers who want to understand the use of algorithms for problem-solving and writing efficient code. Whether you are a beginner looking to learn the most commonly used

algorithms in a clear and concise way or an experienced programmer looking to explore cutting-edge algorithms in data science, machine learning, and cryptography, you'll find this book useful. Although Python programming experience is a must,

knowledge of data science will be helpful but not necessary. Refactoring is gaining momentum amongst the object oriented programming community. It can transform the internal dynamics of applications and has the capacity to transform bad code

into good code. This book offers an introduction to refactoring.
JUnit, created by Kent Beck and Erich Gamma, is an open source framework for test-driven development in any Java-based code. JUnit automates unit testing and reduces the effort

required to frequently test code while developing it. While there are lots of bits of documentation all over the place, there isn't a go-to-manual that serves as a quick reference for JUnit. This Pocket Guide meets the need, bringing together all the bits of hard to

remember information, syntax, and rules for working with JUnit, as well as delivering the insight and sage advice that can only come from a technology's creator. Any programmer who has written, or is writing, Java Code will find this book valuable. Specifically it will

appeal to programmers and developers of any level that use JUnit to do their unit testing in test-driven development under agile methodologies such as Extreme Programming (XP) [another Beck creation].

The Art of Agile Development

Improving the Design of Existing Web Applications
40 Algorithms Every Programmer Should Know
xUnit Test Patterns
Ruby Edition: Ruby Edition
Working Effectively with Legacy Code
A smart and funny YA novel from Jennifer

Honeybourn, When Life Gives You Demons Some people have school spirit. Shelby Black has real ones. Shelby Black has spent the past six months training to be an exorcist. Her great-uncle Roy—a Catholic priest—has put her through exorcist boot camp hell, hoping to

develop her talent, but ohmygod, he still doesn’t trust her to do an exorcism on her own. High school is hard enough without having to explain that you fight demons for a living, so Shelby keeps her extracurricular activity to herself. The last thing she wants is for her crush, Spencer,

to find out what she does in her off time. But Shelby knows how to keep a secret—even a big one. Like the fact that her mom left under mysterious circumstances and it ’ s all her fault. Shelby is hellbent on finding her mom, no matter what it costs her—even if what it

ends up costing her is her soul AND a relationship with Spencer. Praise for Wesley James Ruined My Life: "Everything readers expect and want from a lighthearted summer teen romance....Pitch-perfect." —School Library Journal "Light, cute, and a quick read."

—The Eater of Books

“ Immensely readable, utterly charming and absolutely un-put-downable. ”

—Jennifer McKenzie

Refactoring has proven its value in a wide range of development projects – helping software professionals improve system designs,

maintainability, extensibility, and performance. Now, for the first time, leading agile methodologist Scott Ambler and renowned consultant Pramodkumar Sadalage introduce powerful refactoring techniques specifically designed for database systems. Ambler and

Sadalage demonstrate how small changes to table structures, data, stored procedures, and triggers can significantly enhance virtually any database design – without changing semantics. You ' ll learn how to evolve database schemas in step with source code – and

become far more effective in projects relying on iterative, agile methodologies. This comprehensive guide and reference helps you overcome the practical obstacles to refactoring real-world databases by covering every fundamental concept underlying database

refactoring. Using start-to-finish examples, the authors walk you through refactoring simple standalone database applications as well as sophisticated multi-application scenarios. You ’ ll master every task involved in refactoring database schemas, and discover best practices

for deploying refactorings in even the most complex production environments. The second half of this book systematically covers five major categories of database refactorings. You ’ ll learn how to use refactoring to enhance database structure, data

quality, and referential integrity; and how to refactor both architectures and methods. This book provides an extensive set of examples built with Oracle and Java and easily adaptable for other languages, such as C#, C++, or VB.NET, and other databases, such as DB2,

SQL Server, MySQL, and Sybase. Using this book ' s techniques and examples, you can reduce waste, rework, risk, and cost – and build database systems capable of evolving smoothly, far into the future.

& Most software practitioners deal with inherited code; this

book teaches them how to optimize it & & Workbook approach facilitates the learning process & & Helps you identify where problems in a software application exist or are likely to exist 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
Refactoring JavaScript
Hone your problem-solving skills by learning different algorithms and their implementation in Python
The Rails Way
Learning Software Design Principles by Applying Refactoring Rules

Fowler
Refactoring: Improving the Design of Existing Code
In 1994, Design Patterns changed the landscape of object-oriented development by introducing classic solutions to recurring design problems. In 1999, Refactoring revolutionized design

by introducing an effective process for improving code. With the highly anticipated Refactoring to Patterns , Joshua Kerievsky has changed our approach to design by forever uniting patterns with the evolutionary process of refactoring. This book introduces the theory and practice

of pattern-directed refactorings: sequences of low-level refactorings that allow designers to safely move designs to, towards, or away from pattern implementations. Using code from real-world projects, Kerievsky documents the thinking and steps

underlying over two dozen pattern-based design transformations. Along the way he offers insights into pattern differences and how to implement patterns in the simplest possible ways. Coverage includes: A catalog of twenty-seven pattern-directed refactorings, featuring real-world

code examples Descriptions of twelve design smells that indicate the need for this book ' s refactorings General information and new insights about patterns and refactoring Detailed implementation mechanics: how low-level refactorings are

combined to implement high-level patterns Multiple ways to implement the same pattern – and when to use each Practical ways to get started even if you have little experience with patterns or refactoring Refactoring to Patterns reflects three years of refinement and the

insights of more than sixty software engineering thought leaders in the global patterns, refactoring, and agile development communities. Whether you ’ re focused on legacy or “ greenfield ” development, this book will make you a better software designer

by helping you learn how to make important design changes safely and effectively. Making significant changes to large, complex codebases is a daunting task--one that's nearly impossible to do successfully unless you have the right team, tools, and mindset. If your

application is in need of a substantial overhaul and you're unsure how to go about implementing those changes in a sustainable way, then this book is for you. Software engineer Maude Lemaire walks you through the entire refactoring process from start to finish.

You'll learn from her experience driving performance and refactoring efforts at Slack during a period of critical growth, including two case studies illustrating the impact these techniques can have in the real world. This book will help you achieve a newfound

ability to productively introduce important changes in your codebase. Understand how code degrades and why some degradation is inevitable Quantify and qualify the state of your codebase before refactoring Draft a well-scoped execution plan with strategic milestones Win

support from engineering leadership Build and coordinate a team best suited for the project Communicate effectively inside and outside your team Adopt best practices for successfully executing the refactor Introduction -- China's Sputnik moment -- Copycats in

the Coliseum -- China's alternate Internet universe -- A tale of two countries -- The four waves of AI -- Utopia, dystopia, and the real AI crisis -- The wisdom of cancer -- A blueprint for human co-existence with AI -- Our global AI story

How often do you hear people say things like

this? "Our JavaScript is a mess, but we ’ re thinking about using [framework of the month]." Like it or not, JavaScript is not going away. No matter what framework or ” compiles-to-js ” language or library you use, bugs and performance concerns will always be an issue if

the underlying quality of your JavaScript is poor. Rewrites, including porting to the framework of the month, are terribly expensive and unpredictable. The bugs won't magically go away, and can happily reproduce themselves in a new context. To complicate

things further, features will get dropped, at least temporarily. The other popular method of fixing your JS is playing “JavaScript Jenga,” where each developer slowly and carefully takes their best guess at how the out-of-control system can be altered to allow for new features,

hoping that this doesn’t bring the whole stack of blocks down. This book provides clear guidance on how best to avoid these pathological approaches to writing JavaScript: Recognize you have a problem with your JavaScript quality. Forgive the code you have now,

and the developers who made it. Learn repeatable, memorable, and time-saving refactoring techniques. Apply these techniques as you work, fixing things along the way. Internalize these techniques, and avoid writing as much problematic code to begin with. Bad code

doesn ’ t have to stay that way. And making it better doesn ’ t have to be intimidating or unreasonably expensive.

The Object-Oriented Thought Process

Turning Bad Code Into Good Code

An Agile Primer

Refactoring at Scale

Software Development,

Design and Coding Improving the Design of Existing Code Object-oriented programming (OOP) is the foundation of modern programming languages, including C++, Java, C#, Visual Basic .NET, Ruby, Objective-C, and Swift. Objects also form the basis for many web

technologies such as JavaScript, Python, and PHP. It is of vital importance to learn the fundamental concepts of object orientation before starting to use object-oriented development environments. OOP promotes good design practices, code portability, and reuse – but it requires a shift in thinking to be

fully understood. Programmers new to OOP should resist the temptation to jump directly into a particular programming language or a modeling language, and instead first take the time to learn what author Matt Weisfeld calls “ the object-oriented thought process. ” Written by a developer for developers who want to improve

their understanding of object-oriented technologies, The Object-Oriented Thought Process provides a solutions-oriented approach to object-oriented programming. Readers will learn to understand the proper uses of inheritance and composition, the difference between aggregation and

association, and the important distinction between interfaces and implementations. While programming technologies have been changing and evolving over the years, object-oriented concepts remain a constant – no matter what the platform. This revised edition focuses on the OOP technologies that have survived the

past 20 years and remain at its core, with new and expanded coverage of design patterns, avoiding dependencies, and the SOLID principles to help make software designs understandable, flexible, and maintainable.
This book focuses on the methodological treatment of UML/P and addresses three core topics of model-based

software development: code generation, the systematic testing of programs using a model-based definition of test cases, and the evolutionary refactoring and transformation of models. For each of these topics, it first details the foundational concepts and techniques, and then presents their application with UML/P. This

separation between basic principles and applications makes the content more accessible and allows the reader to transfer this knowledge directly to other model-based approaches and languages. After an introduction to the book and its primary goals in Chapter 1, Chapter 2 outlines an agile UML-based approach using

UML/P as the primary development language for creating executable models, generating code from the models, designing test cases, and planning iterative evolution through refactoring. In the interest of completeness, Chapter 3 provides a brief summary of UML/P, which is used throughout the book.

Next, Chapters 4 and 5 discuss core techniques for code generation, addressing the architecture of a code generator and methods for controlling it, as well as the suitability of UML/P notations for test or product code. Chapters 6 and 7 then discuss general concepts for testing software as well as the special features

which arise due to the use of UML/P. Chapter 8 details test patterns to show how to use UML/P diagrams to define test cases and emphasizes in particular the use of functional tests for distributed and concurrent software systems. In closing, Chapters 9 and 10 examine techniques for transforming models and

code and thus provide a solid foundation for refactoring as a type of transformation that preserves semantics. Overall, this book will be of great benefit for practical software development, for academic training in the field of Software Engineering, and for research in the area of model-based software

development. Practitioners will learn how to use modern model-based techniques to improve the production of code and thus significantly increase quality. Students will find both important scientific basics as well as direct applications of the techniques presented. And last but not least, the book will offer scientists a

comprehensive overview of the current state of development in the three core topics it covers. Fully Revised and Updated – Includes New Refactorings and Code Examples “ Any fool can write code that a computer can understand. Good programmers write code that humans can understand. ” —M.

Fowler (1999)For more than twenty years, experienced programmers worldwide have relied on Martin Fowler ’ s Refactoring to improve the design of existing code and to enhance software maintainability, as well as to make existing code easier to understand. This eagerly awaited new edition has been fully

updated to reflect crucial changes in the programming landscape. Refactoring, Second Edition, features an updated catalog of refactorings and includes JavaScript code examples, as well as new functional examples that demonstrate refactoring without classes. Like the original, this edition explains what refactoring

is; why you should refactor; how to recognize code that needs refactoring; and how to actually do it successfully, no matter what language you use. Understand the process and general principles of refactoring Quickly apply useful refactorings to make a program easier to comprehend and change Recognize “ bad

smells ” in code that signal opportunities to refactor Explore the refactorings, each with explanations, motivation, mechanics, and simple examples Build solid tests for your refactorings Recognize tradeoffs and obstacles to refactoring Includes free access to the canonical web edition, with even more refactoring resources.

(See inside the book for details about how to access the web edition.) The Complete Guide to Writing More Maintainable, Manageable, Pleasing, and Powerful Ruby Applications Ruby's widely admired ease of use has a downside: Too many Ruby and Rails applications have been created without concern

for their long-term maintenance or evolution. The Web is awash in Ruby code that is now virtually impossible to change or extend. This text helps you solve that problem by using powerful real-world object-oriented design techniques, which it thoroughly explains using simple and practical Ruby examples.

This book focuses squarely on object-oriented Ruby application design. Practical Object-Oriented Design in Ruby will guide you to superior outcomes, whatever your previous Ruby experience. Novice Ruby programmers will find specific rules to live by; intermediate Ruby programmers will find

valuable principles they can flexibly interpret and apply; and advanced Ruby programmers will find a common language they can use to lead development and guide their colleagues. This guide will help you Understand how object-oriented programming can help you craft Ruby code that is easier to maintain and upgrade

Decide what belongs in a single Ruby class Avoid entangling objects that should be kept separate Define flexible interfaces among objects Reduce programming overhead costs with duck typing Successfully apply inheritance Build objects via composition Design cost-effective tests Solve common problems associated with poorly

designed Ruby code
Refactoring for Software Design Smells
Refactoring HTML
Code Generation, Testing, Refactoring
Refactoring: Improving The Design Of Existing Code
Practical Object-oriented Design in Ruby
Refactoring Test Code
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. The Complete Guide to Writing Maintainable,

Manageable, Pleasing, and Powerful Object-Oriented Applications Object-oriented programming languages exist to help you create beautiful, straightforward applications that are easy to change and simple to extend. Unfortunately, the world is awash with object-oriented (OO) applications that are difficult to understand

and expensive to change. Practical Object-Oriented Design, Second Edition, immerses you in an OO mindset and teaches you powerful, real-world, object-oriented design techniques with simple and practical examples. Sandi Metz demonstrates how to build new applications that can “ survive success ” and

repair existing applications that have become impossible to change. Each technique is illustrated with extended examples in the easy-to-understand Ruby programming language, all downloadable from the companion website, poodr.com. Fully updated for Ruby 2.5, this guide shows how to Decide what belongs in a

single class Avoid entangling objects that should be kept separate Define flexible interfaces among objects Reduce programming overhead costs with duck typing Successfully apply inheritance Build objects via composition Whatever your previous object-oriented experience, this concise guide will help you

achieve the superior outcomes you ’ re looking for. Register your book for convenient access to downloads, updates, and/or corrections as they become available. See inside book for details. More than 300,000 developers have benefited from past editions of UML Distilled . This third edition is the

best resource for quick, no-nonsense insights into understanding and using UML 2.0 and prior versions of the UML. Some readers will want to quickly get up to speed with the UML 2.0 and learn the essentials of the UML. Others will use this book as a handy, quick reference to the most common parts of the UML. The author

delivers on both of these promises in a short, concise, and focused presentation. This book describes all the major UML diagram types, what they're used for, and the basic notation involved in creating and deciphering them. These diagrams include class, sequence, object, package, deployment, use case, state machine,

activity, communication, composite structure, component, interaction overview, and timing diagrams. The examples are clear and the explanations cut to the fundamental design logic. Includes a quick reference to the most useful parts of the UML notation and a useful summary of diagram types that were added to

the UML 2.0. If you are like most developers, you don't have time to keep up with all the new innovations in software engineering. This new edition of Fowler's classic work gets you acquainted with some of the best thinking about efficient object-oriented software design using the UML--in a convenient format that will be

essential to anyone who designs software professionally.
The Definitive Refactoring Guide, Fully Revamped for Ruby
With refactoring, programmers can transform even the most chaotic software into well-designed systems that are far easier to evolve and maintain. What's more, they can do it one step at

a time, through a series of simple, proven steps. Now, there's an authoritative and extensively updated version of Martin Fowler's classic refactoring book that utilizes Ruby examples and idioms throughout-not code adapted from Java or any other environment. The authors introduce a

detailed catalog of more than 70 proven Ruby refactorings, with specific guidance on when to apply each of them, step-by-step instructions for using them, and example code illustrating how they work. Many of the authors' refactorings use powerful Ruby-specific features, and all code samples are available for download. Leveraging

Fowler's original concepts, the authors show how to perform refactoring in a controlled, efficient, incremental manner, so you methodically improve your code's structure without introducing new bugs. Whatever your role in writing or maintaining Ruby code, this book will be an indispensable

resource. This book will help you * Understand the core principles of refactoring and the reasons for doing it * Recognize "bad smells" in your Ruby code * Rework bad designs into well-designed code, one step at a time * Build tests to make sure your refactorings work properly * Understand the challenges of

refactoring and how they can be overcome * Compose methods to package code properly * Move features between objects to place responsibilities where they fit best * Organize data to make it easier to work with * Simplify conditional expressions and make more effective use of polymorphism * Create interfaces that are

easier to understand and use * Generalize more effectively * Perform larger refactorings that transform entire software systems and may take months or years * Successfully refactor Ruby on Rails code

The Pragmatic Programmer

Refactoring Databases

Unit Testing Principles, Practices, and Patterns

Java Concurrency in Practice
AI Superpowers China, Silicon Valley, and the New World Order
Many businesses and organizations depend on older high-value PHP software that risks abandonment because it is impossible to maintain. The reasons for this may be that the software is not

well designed; there is only one developer (the one who created the system) who can develop it because he didn ' t use common design patterns and documentation; or the code is procedural, not object-oriented. With this book, you ' ll learn to identify problem code and refactor it to create more effective applications using test-

driven design.
Agile Modeling with UML
your journey to mastery, 20th Anniversary Edition
Refactoring Object-Oriented Frameworks
With Patterns, Debugging, Unit Testing, and Refactoring