

Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology | ed4374a70e09aa34147d16636651e6fc

Release It! Debugging Refactoring Object-Oriented
Frameworks Refactoring Working Effectively with Legacy
Code Refactoring Refactoring: Improving The Design Of Existing Code The
Software Craftsman Refactoring Refactoring JavaScript Pair Programming
Illuminated Refactoring Agile Modeling with UML The Pragmatic
Programmer The Rust Programming Language (Covers Rust
2018) Prefactoring Implementation Patterns Refactoring: Improving the
Design of Existing Code Service Design Patterns Refactoring to
Patterns Refactoring in Ruby Beyond Software Architecture Pro PHP
Refactoring The Clean Coder Domain-Specific Languages Refactoring
HTML Refactoring Databases Refactoring HTML UML Distilled Python
Workout Pragmatic Thinking and Learning Refactoring xUnit Test
Patterns Refactoring Workbook Test-Driven iOS
Development Fowler Refactoring for Software Design Smells Analysis
Patterns The Object-Oriented Thought Process The Robert C. Martin Clean
Code Collection (Collection)

This innovative book recognizes the need within the object-oriented community for a book that goes beyond the tools and techniques of the typical methodology book. In *Analysis Patterns: Reusable Object Models*, Martin Fowler focuses on the end result of object-oriented analysis and design—the models themselves. He shares with you his wealth of object modeling experience and his keen eye for identifying repeating problems and transforming them into reusable models. *Analysis Patterns* provides a catalogue of patterns that have emerged in a wide range of domains including trading, measurement, accounting and organizational relationships. Recognizing that conceptual patterns cannot exist in isolation, the author also presents a series of "support patterns" that discuss how to turn conceptual models into software that in turn fits into an architecture for a large information system. Included in each pattern is the reasoning behind their design, rules for when they should and should not be used, and tips for implementation. The examples presented in this book comprise a cookbook

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

of useful models and insight into the skill of reuse that will improve analysis, modeling and implementation.

& 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

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

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

learned HTML years ago, and want to refresh their knowledge with today's standards-compliant best practices.

The Robert C. Martin Clean Code Collection consists of two bestselling eBooks: Clean Code: A Handbook of Agile Software Craftmanship The Clean Coder: A Code of Conduct for Professional Programmers In Clean Code, legendary software expert Robert C. Martin has teamed up with his colleagues from Object Mentor to distill their best agile practice of cleaning code "on the fly" into a book that will instill within you the values of a software craftsman and make you a better programmer--but only if you work at it. You will be challenged to think about what's right about that code and what's wrong with it. More important, you will be challenged to reassess your professional values and your commitment to your craft. In The Clean Coder, Martin introduces the disciplines, techniques, tools, and practices of true software craftsmanship. This book is packed with practical advice--about everything from estimating and coding to refactoring and testing. It covers much more than technique: It is about attitude. Martin shows how to approach software development with honor, self-respect, and pride; work well and work clean; communicate and estimate faithfully; face difficult decisions with clarity and honesty; and understand that deep knowledge comes with a responsibility to act. Readers of this collection will come away understanding How to tell the difference between good and bad code How to write good code and how to transform bad code into good code How to create good names, good functions, good objects, and good classes How to format code for maximum readability How to implement complete error handling without obscuring code logic How to unit test and practice test-driven development What it means to behave as a true software craftsman How to deal with conflict, tight schedules, and unreasonable managers How to get into the flow of coding and get past writer's block How to handle unrelenting pressure and avoid burnout How to combine enduring attitudes with new development paradigms How to manage your time and avoid blind alleys, marshes, bogs, and swamps How to foster environments where programmers and teams can thrive When to say "No"--and how to say it When to say "Yes"--and what yes really means

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

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

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.

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

A complete practitioner's catalog of proven domain services design solutions that can help any organization leverage SOA's full benefits *
*Provides a vocabulary of proven SOA design solutions, with concrete examples and code that is easy for architects to adapt and implement. *By Rob Daigneau, one of the industry's leading experts in complex systems

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

integration. *Helps architects and IT leaders accurately set stakeholder expectations for major SOA initiatives. Service-oriented architectures are typically called upon to deliver two general categories of services: enterprise services and domain services. Enterprise services are essentially composite services that typically leverage technologies such as message-oriented middleware. Domain services are the building blocks these composites depend upon. Each service category is best served by a distinct set of design solutions. This is the first book to systematically identify and explain best practice patterns for domain services. Rob Daigneau expands upon the Service Layer concept (covered expertly by Fowler in Patterns of Enterprise Application Architecture) domain services can be used with Enterprise Integration Patterns (made famous by Hohpe and Woolf). Daigneau begins by reviewing SOA concepts, illuminating the distinctions between enterprise and domain services, and identifying key relationships between domain services and other pattern groups. Next, he introduces each essential pattern for creating and delivering domain services, providing a vocabulary of design solutions that architects and other IT professionals can implement by referencing and adapting the concrete examples he supplies.

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.

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

"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.

This text aims to help all members of the development team make the correct nuts-and-bolts architecture decisions that ensure project success.

"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

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

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.

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

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

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.)

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.

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-

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

time iOS/Mac developer Graham Lee helps you rapidly integrate TDD into your existing processes using Apple's Xcode 4 and the OUnit 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 OUnit 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.

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

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

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.

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.

Written as instruction for pair programming newbies, with practical improvement tips for those experienced with the concept, this guide explores the operational aspects and unique fundamentals of pair programming; information such as furniture set-up, pair rotation, and weeding out bad pairs.

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

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

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.

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.

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

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.

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

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

dependency-breaking techniques that help you work with program elements in isolation and make safer changes.

As the application of object technology--particularly 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.

Software Expert Kent Beck Presents a Catalog of Patterns Infinitely Useful for Everyday Programming Great code doesn't just function: it clearly and consistently communicates your intentions, allowing other programmers to understand your code, rely on it, and modify it with confidence. But great

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

code doesn't just happen. It is the outcome of hundreds of small but critical decisions programmers make every single day. Now, legendary software innovator Kent Beck—known worldwide for creating Extreme Programming and pioneering software patterns and test-driven development—focuses on these critical decisions, unearthing powerful “implementation patterns” for writing programs that are simpler, clearer, better organized, and more cost effective. Beck collects 77 patterns for handling everyday programming tasks and writing more readable code. This new collection of patterns addresses many aspects of development, including class, state, behavior, method, collections, frameworks, and more. He uses diagrams, stories, examples, and essays to engage the reader as he illuminates the patterns. You'll find proven solutions for handling everything from naming variables to checking exceptions.

Presents practical advice on the disciplines, techniques, tools, and practices of computer programming and how to approach software development with a sense of pride, honor, and self-respect.

The First Hands-On, Practical, All-Ruby Refactoring Workbook!
Refactoring—the art of improving the design of existing code—has taken the world by storm. So has Ruby. Now, for the first time, there's a refactoring workbook designed from the ground up for the dynamic Ruby language. Refactoring in Ruby gives you all the realistic, hands-on practice you need to refactor Ruby code quickly and effectively. You'll discover how to recognize “code smells,” which signal opportunities for improvement, and then perfect your program's design one small, safe step at a time. The book shows you when and how to refactor with both legacy code and during new test-driven development, and walks you through real-world refactoring in detail. The workbook concludes with several applications designed to help practice refactoring in realistic domains, plus a handy code review checklist you'll refer to again and again. Along the way, you'll learn powerful lessons about designing higher quality Ruby software—lessons that will enable you to experience the joy of writing consistently great code. Refactoring in Ruby will help you Recognize why poor code design occurs, so you can prevent it from occurring in your own code Master better design techniques that lead to more efficient, reliable, and maintainable software Fix code that's too long, large, or difficult to follow Ferret out duplication, and express each idea “once and only once” Recognize missing or inadequately formed classes

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

Simplify overly complex relationships between classes and their subclasses
Achieve the right balance of responsibilities among objects
Make your code easier to test and change
Cope with incomplete library modules, and fix runaway dependencies
Learn the next steps to take after you refactor

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.

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

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

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 only way to master a skill is to practice. In Python Workout, author Reuven M. Lerner guides you through 50 carefully selected exercises that invite you to flex your programming muscles. As you take on each new challenge, you'll build programming skill and confidence. Summary The only way to master a skill is to practice. In Python Workout, author Reuven M. Lerner guides you through 50 carefully selected exercises that invite you to flex your programming muscles. As you take on each new challenge, you'll build programming skill and confidence. The thorough explanations help you lock in what you've learned and apply it to your own projects. Along the way, Python Workout provides over four hours of video instruction walking you through the solutions to each exercise and dozens of additional exercises for you to try on your own. Purchase of the print book includes a free eBook in PDF, Kindle, and ePub formats from Manning Publications. About the technology To become a champion Python programmer you need to work out, building mental muscle with your hands on the keyboard. Each carefully selected exercise in this unique book adds to your Python prowess—one important skill at a time. About the book Python Workout presents 50 exercises that focus on key Python 3 features. In it, expert Python coach Reuven Lerner guides you through a series of small projects, practicing the skills you need to tackle everyday tasks. You'll appreciate the clear

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

explanations of each technique, and you can watch Reuven solve each exercise in the accompanying videos. What's inside 50 hands-on exercises and solutions Coverage of all Python data types Dozens more bonus exercises for extra practice About the reader For readers with basic Python knowledge. About the author Reuven M. Lerner teaches Python and data science to companies around the world. Table of Contents 1 Numeric types 2 Strings 3 Lists and tuples 4 Dictionaries and sets 5 Files 6 Functions 7 Functional programming with comprehensions 8 Modules and packages 9 Objects 10 Iterators and generators

Writing for programmers and non-programmers alike, Harold explains how to use refactoring to improve virtually any Web site or application, and shows them how to refactor for better reliability, performance, usability, security, accessibility, compatibility, and even search engine placement.

Printed in full color. Software development happens in your head. Not in an editor, IDE, or designtool. You're well educated on how to work with software and hardware, but what about wetware--our own brains? Learning new skills and new technology is critical to your career, and it's all in your head. In this book by Andy Hunt, you'll learn how our brains are wired, and how to take advantage of your brain's architecture. You'll learn new tricks and tipsto learn more, faster, and retain more of what you learn. You need a pragmatic approach to thinking and learning. You need to Refactor Your Wetware. Programmers have to learn constantly; not just the stereotypical new technologies, but also the problem domain of the application, the whims of the user community, the quirks of your teammates, the shifting sands of the industry, and the evolving characteristics of the project itself as it is built. We'll journey together through bits of cognitive and neuroscience, learning and behavioral theory. You'll see some surprising aspects of how our brains work, and how you can take advantage of the system to improve your own learning and thinking skills. In this book you'll learn how to: Use the Dreyfus Model of Skill Acquisition to become more expert Leverage the architecture of the brain to strengthen different thinking modes Avoid common "known bugs" in your mind Learn more deliberately and more effectively Manage knowledge more efficiently

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

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

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

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

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

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.

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.

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 rules of battle for tracking down -- and eliminating -- hardware and software bugs. When the pressure is on to root out an elusive software or hardware glitch, what's needed is a cool head courtesy of a set of rules guaranteed to work on any system, in any circumstance. Written in a frank but engaging style, Debugging provides simple, foolproof principles guaranteed to help find any bug quickly. This book makes those shelves of application-specific debugging books (on C++, Perl, Java, etc.) obsolete. It changes the way readers think about debugging, making those pesky problems suddenly much easier to find and fix. Illustrating the rules with real-life bug-detection war stories, the book shows readers how to:

- * Understand the system: how perceiving the ""roadmap"" can hasten your journey
- * Quit thinking and look: when hands-on investigation can't be avoided
- * Isolate critical factors: why changing one element at a time can be an essential tool
- * Keep an audit trail: how keeping a record of the debugging process can

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

win the day The rules of battle for tracking down -- and eliminating -- hardware and software bugs. When the pressure is on to root out an elusive software or hardware glitch, what's needed is a cool head courtesy of a set of rules guaranteed to work on any system, in any circumstance. Written in a frank but engaging style, Debugging provides simple, foolproof principles guaranteed to help find any bug quickly. This book makes those shelves of application-specific debugging books (on C++, Perl, Java, etc.) obsolete. It changes the way readers think about debugging, making those pesky problems suddenly much easier to find and fix. Illustrating the rules with real-life bug-detection war stories, the book shows readers how to:

- * Understand the system: how perceiving the "roadmap" can hasten your journey
- * Quit thinking and look: when hands-on investigation can't be avoided
- * Isolate critical factors: why changing one element at a time can be an essential tool
- * Keep an audit trail: how keeping a record of the debugging process can win the day

The rules of battle for tracking down -- and eliminating -- hardware and software bugs. When the pressure is on to root out an elusive software or hardware glitch, what's needed is a cool head courtesy of a set of rules guaranteed to work on any system, in any circumstance. Written in a frank but engaging style, Debugging provides simple, foolproof principles guaranteed to help find any bug quickly. This book makes those shelves of application-specific debugging books (on C++, Perl, Java, etc.) obsolete. It changes the way readers think about debugging, making those pesky problems suddenly much easier to find and fix. Illustrating the rules with real-life bug-detection war stories, the book shows readers how to:

- * Understand the system: how perceiving the "roadmap" can hasten your journey
- * Quit thinking and look: when hands-on investigation can't be avoided
- * Isolate critical factors: why changing one element at a time can be an essential tool
- * Keep an audit trail: how keeping a record of the debugging process can win the day

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

Online Library Refactoring Improving The Design Of Existing Code Addison Wesley Object Technology

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.)

Copyright code : [ed4374a70e09aa34147d16636651e6fc](#)