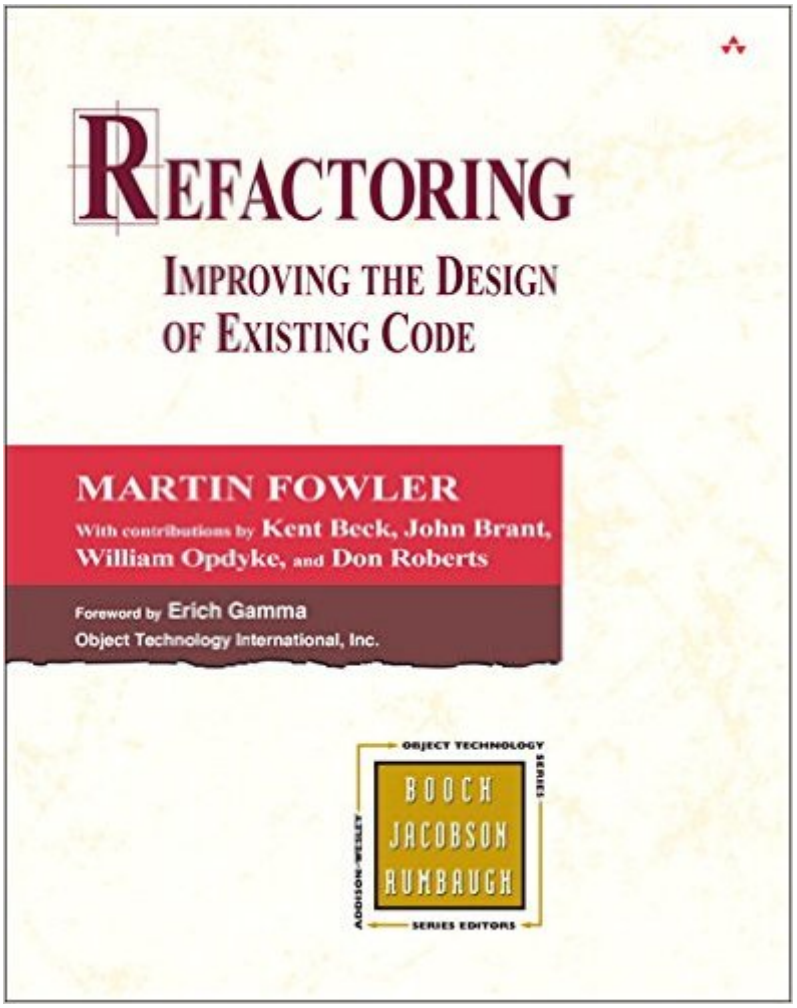
Refactoring: Improving The Design Of Existing Code



Synopsis

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 Software, 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 designe

Book Information

Hardcover: 464 pages
Publisher: Addison-Wesley Professional; 1 edition (July 8, 1999)
Language: English
ISBN-10: 0201485672
ISBN-13: 978-0201485677
Product Dimensions: 7.7 x 1.2 x 9.3 inches
Shipping Weight: 2.3 pounds (View shipping rates and policies)
Average Customer Review: 4.5 out of 5 starsÂ Â See all reviewsÂ (203 customer reviews)
Best Sellers Rank: #22,972 in Books (See Top 100 in Books) #10 inÂ Books > Textbooks > Computer Science > Object-Oriented Software Design #22 inÂ Books > Computers & Technology > Programming > Software Design, Testing & Engineering > Object-Oriented Design #89 inÂ Books > Textbooks > Computer Science > Programming Languages

Customer Reviews

Like the Gang of Four's landmark book _Design Patterns_, Fowler and his cohorts have created another catalog-style book, this time on refactoring.Refactoring refers to taking existing, working software, and changing it about to improve its design, so that future modifications and enhancements are easier to add. _Refactoring_ is primarily a catalog of 70 or so different kinds of

improvements you can make to object-oriented software.Each entry in the catalog describes an implementation problem, the solution, motivation for applying the solution, the mechanics of the refactoring, and examples. The book's examples are all in Java, but C++ programmers should be able to approach the refactorings with ease. Often, Fowler diagrams the refactorings in UML, so a little Unified Modeling Language experience will help, too.While the catalog is nice, the kinds of refactorings are obvious is most cases. Even moderately experienced programmers won't need the step-by-step mechanics described. The real benefit, though, is that the mechanics of each refactoring help guarantee that you can pull off the refactoring without introducing new bugs or side effects. They encourage you to take smaller, verifiable steps, than the more gross refactorings that most developers would naturally take. You actually save time doing so.How do you know your refactorings are safe? Unit testing is the answer that Fowler et al. provide. Java developers will find the introduction to the Junit Testing Framework the most valuable part of the book, more so than the catalog of refactorings itself.There's more to the book than the catalog and Junit, of course.

A little while back I was introduced to a word I had never heard before, Refactoring. I was told toget Martin Fowler's book and read it so I could gain a better understanding of what Refactoringwas. Well folks, I would classify this book as a 'Hidden Treasure'.Although it is not a flashy or well known title, I believe its impact can be much deeper and longlasting than many of the mainstream, more popular technology books. The underlying theoriesthat it teaches can be applied for years, even when languages change.There are only a couple of things I would change about this book, which I will mention below.PrefaceThe Preface it brief enough, and gives the definition for the word Refactoring. This is a good thingbecause right form the start you get the true definition of Refactoring. In short, refactoring is theprocess of changing code to improve the internal structure, but not changing the externalbehavior.Chapter 1: Refactoring, a First ExampleIn this chapter Mr. Fowler tries to start by showing a simple Refactoring example. The problem isthat the chapter then goes on for 50+ pages. Mr. Fowler explains his reasons for doing this, but Ithink that a simple example should have been much simpler. Especially when it is in the firstchapter of the book. It's not that this isn't a good chapter. I feel it's just too soon in the book. Iwould have put it at the end.Chapter 2: Principles of RefactoringThis is an excellent chapter. The definition of Refactoring is discussed as well as the followingquestions: Why should you refactor? When should you refactor? What do I tell my manager?

The basic thesis of this book is that, for various reasons, real programs are poorly designed. They

get that way for a variety of reasons. Initially well designed, extending the program may lead to software decay. Huge methods may result from unanticipated complexity. Refactoring, according to Fowler, is a function preserving transformation of a program. The transformations are reversible, so the intention is to improve the program in some way.Fowler suggests refactoring a program to simplify the addition of new functionality. The program should also be refactored to make it easier for human readers to understand at the same time.He also insists that each step is small and preserves functionality, and on frequent unit testing with a comprehensive test suite.Half of the book consists of a catalogue of refactorings. He gives each refactoring a memorable name, such as "Replace Type Code with Subclasses". He illustrates the design transformation with a pair of UML class diagrams, and has a standard set of sections: Motivation, Mechanics and Example.The Motivation is a prose section that describes and justifies the refactoring, showing the relationship to other refactorings.The Mechanics is a sequence of steps needed to carry out the refactoring, shown as a list of bullet points He expands on some points.The Example is where the value of this book lies. Fowler takes a fragment of Java code, and takes us step by step through the refactoring. The code is small enough that he can show it all each step of the way without overwhelming us, but is large enough to be realistic.The code is clear enough for non-Java programmers to follow.

flexibility and protecting against worst-case scenarios with XHTML and CSS (2nd Edition) Bulletproof Web Design: Improving flexibility and protecting against worst-case scenarios with HTML5 and CSS3 (3rd Edition) (Voices That Matter) The ITSM Process Design Guide: Developing, Reengineering, and Improving IT Service Management Bulletproof Web Design: Improving flexibility and protecting against worst-case scenarios with XHTML and CSS Discussing Design: Improving Communication and Collaboration through Critique