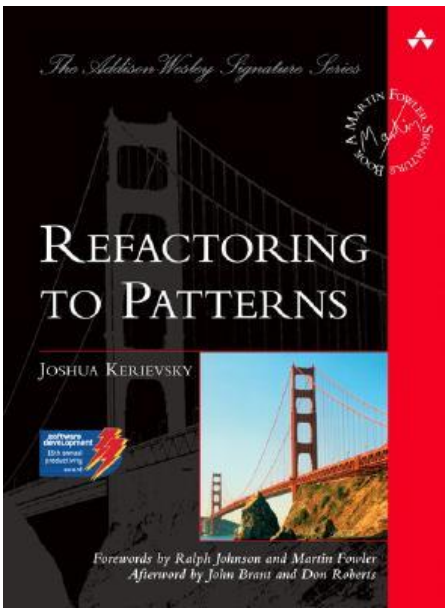

Refactoring to Patterns

Joshua Kerievsky , Martin Fowler (Foreword by) , Ralph Johnson (Foreword by)

Refactoring to Patterns

Joshua Kerievsky , Martin Fowler (Foreword by) , Ralph Johnson (Foreword by)

Refactoring to Patterns Joshua Kerievsky , Martin Fowler (Foreword by) , Ralph Johnson (Foreword by) 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 refactoringDetailed 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.
"

Refactoring to Patterns Details

Date	:	Published August 15th 2004 by Addison-Wesley Professional (first published 2004)
ISBN	:	0785342213355
Author	:	Joshua Kerievsky , Martin Fowler (Foreword by) , Ralph Johnson (Foreword by)
Format	:	Hardcover 400 pages
Genre	:	Computer Science, Programming, Science, Technology, Software, Technical

From Reader Review Refactoring to Patterns for online ebook

Ash Mishra says

The subject material in this book is what separates those who think they understand the purpose and utilization of patterns, from those who realize that patterns are essential not to just the design of an application, but more importantly to its extensibility and forward maintenance. Too often as software engineers, we have seen two camps of developers: those who are new to the field and unaware of good design, and the latter are those armed and dangerous with knowledge of patterns, but use them to overengineer solutions. This book provides insights into a balance - a systematical method of "refactoring" to a pattern. Filled with a large catalog of patterns (27), and with real-world examples, Joshua Kerievsky has done a fantastic job of illustrating and explaining a pattern and it's use, compared to many previous books on the subjects of patterns, which are to say very dry-guaranteed-to-kill-your-passion at the least. Well worth a read.

Justin says

Kerievsky provides a succinct set of patterns with non-trivial examples for each. All developers should have this for reference.

Tim says

Really useful reference material. You need to be comfortable with design patterns in order to fully appreciate the message of this book. The mechanics for each type of refactoring is refreshing as we're often introduced to design patterns or refactoring from a singular example. This book bridges the gap between an existing solution to one that uses design patterns.

Marko Kunic says

This should for sure be your first book about patterns. I really enjoyed the approach in this book, it is very well explained. Joshua Kerievsky first shows the problem and then refactors the code step by step into a pattern. Why did I enjoy the approach? Because giving you a pattern catalog and just going over patterns isn’t enough. You need to understand when to use patterns and not to use it prematurely, maybe you won’t even need it. When you see the “bad” code and you take steps to refactor it into a pattern you start appreciating that pattern more.

Jordi Espasa Cusachs says

The book drives you into the world of patterns in a very didactic way. Easy to read, engages you to use the patterns and also, very important, when not to use it. Full of stories and real examples, it shows you the decision process to when to use a pettern or another, or not use it at all. Not using patterns is an enemy, overengineering is an enemy as well.

Frederik Sørensen says

Without context advises from pattern books like Design Patterns: Elements of Reusable Object-Oriented Software can be difficult to apply. This is where Joshua's book shine. He walks through many different patterns, shows how messy code could look and explains how to refactor the code into patterns.

Most importantly he also gives many advises to when not to apply the refactorings. I particularly likes the boxes with pros and cons for each refactoring.

The book heavily references Design Patterns: Elements of Reusable Object-Oriented Software and Refactoring: Improving the Design of Existing Code. Although I have read both books I find it difficult to figure out which order it would be best to read the books in. But in the end all three books needs to be read multiple times to absorb all the knowledge.

The main concern I have with the book is the mechanics section that step by step explain how to apply the refactoring, right after this there is an example. I think the mechanics are almost redundant. I say almost because the section might prove usefull when using the book as a reference.

Rod Hilton says

Refactoring to Patterns essentially attempts to merge together two types of books: books on code design (such as the GoF Patterns book, Head First Design Patterns, Patterns of Enterprise Application Architecture, etc) and books on engineering principles (such as the eXtreme Programming series and, most of all, Fowler's Refactoring).

In this, it succeeds. Refactoring to Patterns really makes explicit the relationship between agile programming practices and OO design. It draws a connection precisely where two concepts are connected but where few books tread. Kerievsky isn't a Patternista either, he makes it clear that Patterns are often overkill for some types of problems, and he always mentions both the benefits and costs of a pattern being discussed.

The main problem with the book is that it exists to create a connection that I think most people can create on their own. If you've read Refactoring and a Design Patterns book, I don't think there's a lot to be gained here. Often I found myself reading the writeup of a pattern, looking at its diagram, and skimming the "How To" section. I'd often skip the example entirely, as it was clear by that point that I knew exactly how to refactor to a pattern.

The book claims that its goal is to make it clear not just HOW to implement a pattern (which GoF does a great job of), but WHY. What does the code look like when it's in a state that it should be refactored to a design pattern? Kerievsky does a decent job with this goal, but often I wondered if it would have been better for him to go into more examples and detail on that subject, and spend less time taking the reader step-by-step through the actual refactoring itself. Those pages often felt like padding in a book struggling to reach 300 pages.

If you feel comfortable implementing Design Patterns but unsure of when it's appropriate to use them, this would be a good book, though I have a hard time believing there isn't a better one out there (though to be fair, I can't think of one). Otherwise, I don't think there's much to glean from the book - often pages were

spent spelling out refactoring steps that anyone with some experience with Patterns probably already understands.

Johnny Graber says

If you read Refactoring, then this book will be the next step. While Martin Fowler explains in Refactoring the mechanics of the trade, Joshua Kerievsky explains how you can use the small building blocks to make significant refactorings towards patterns. Especially helpful when you need to solve problems that fall within a useful pattern and you don’t want to reinvent the wheel. The book is even better when you are at a pattern that someone forced upon your application but doesn’t solve the problem. You will profit from the parts of the book that explain refactorings away from a pattern. That is an often-overlooked aspect of the book that can bring you the most. Therefore, please read those parts as well.

Josh Hamacher says

Like several other reviewers, this book left me scratching my head slightly and wondering what its aim really was. I was hoping the focus would be more on analysis of legacy code, with advice on discovering and teasing out potential patterns. Instead, this is almost entirely a "how-to" book.

The vast majority of its 350 pages are taken up with 27 refactorings. Each refactoring includes a "how to" section and then an (often lengthy) step-by-step example. Yet, if you're familiar with design patterns and refactorings, both sections could be significantly shorter.

Only about 50 pages are devoted to the when-and-why of refactoring, and I found the advice there to be fairly generic.

It's not a bad book; as programming books go it's pretty well written and mostly manages to avoid coming off as dry and academic despite its subject matter. But I just don't think it really adds much to the literature on either refactorings or design patterns.

Apple84 Wylie says

In regard to design patterns, lines can be strongly drawn between developers. Some argue it is the only way to code while others believe the practice is sterile and inhibits creativity. I think patterns are useful in some situations and a hinderance in others; for me, their utility factors on a large number of variables, including project type, resources, language, and number of developers involved in the project. It helps to understand and research the technology if only to eschew or discount the position. So--I recommend this book to any developer, if only as an overview of re-factoring and design pattern usage. I recommend reading the "gang-of-four" book (Design Patterns: Elements of Reusable Object-Oriented Software) prior to reading this one, however.

Blair Conrad says

A very good book, balancing the need to present useful refactorings against the risk of alienating readers with too-complicated refactorings. The constant references to Martin Fowler’s work were justified, and if you really want to get the most out of this book, you should have Refactoring and Design Patterns with you. I didn’t, though, and still found it very interesting. By the end, the “mechanism” section of each refactoring was a little tough slogging, but the examples were very followable. A must-have for work, and I was considering shelling out my own money for a copy, until my wife bought me a copy for my birthday, because she loves me even though I'm a geek.
Oh, and there are **two** integrated bookmarks!

Marshall says

This book is an excellent combination of Design Patterns and Refactoring. It's perfect if you're looking to improve your understanding of design patterns and/or refactoring, but it really shines if your goal is to understand how they both work together. Rather than thinking of design patterns as things you cook into your program, which is what usually leads to "design pattern abuse," this book recommends you start with a simple design first, and evolve to design patterns if you start noticing "code smells" that are ideally solved with them, unless you know without a doubt that you will need them.

This book is organized exactly like Refactoring, and looks very simiilar in its layout. Unlike Refactoring, this book isn't quite so useful as a cookbook of common refactorings. Its refactorings are very uncommon, but the situations that need them REALLY need them. So, as the Afterword recommends, don't try so hard to get good at these refactorings. Instead, use it to understand the thought processes that lead to those refactorings. Don't memorize this book--"grok" it.

The code samples in this book are perfect, short enough to be straightforward and concise, but real enough to not resort to "toy code." This is real-world code, and they are perfect candidates for the refactorings they are used to demonstrate. However, I wasn't so impressed with the "Mechanics" section of each refactoring. They were very hard to follow, though I'm not sure how they can be improved, so it may just be a symptom of the complex nature of many of these refactorings, rather than a reflection on the author's explanatory abilities.

Ronald Rajagukguk says

Personally I expect more the book, nevertheless it gave me quite an impression. The book gave a lot of code example but unfortunately some of them is unclear, which need me to stare at the code several minutes till I understand the author intention. Good book a software engineer who want to know design pattern deeper, but i don't recommend this book for beginner.

Stijn says

Refactoring is my favorite topic in Software Quality. This book has only made me an even a bigger Merciless Refactorer. I like the way Joshua put the focus on learning the problem and not the solution.

Madhur Ahuja says

Another book to be read again and again, after "Refactoring: Improving the Design of Existing Code"