Prefactoring approaches software development of new systems using lessons learned from many developers over the years. It is a compendium of ideas gained from retrospectives on what went right and what went wrong in development. Some of these ideas came from experience in refactoring. Refactoring is improving the design of existing code to make it simpler and easier to maintain.

This practical, thought-provoking guide details prefactoring guidelines in design, code, and testing. These guidelines can help you create more readable and maintainable code in your next project.

To help communicate the many facets of this approach, Prefactoring follows the development of a software system for a fictitious client, named Sam, from vision through implementation. Some of the guidelines you'll encounter along the way include:

- When You're Abstract, Be Abstract All the Way
- Splitters Can Be Lumped Easier Than Lumpers Can Be Split
- Do a Little Job Well and You May Be Called Upon Often
- Plan Globally, Develop Locally
- Communicate with Your Code
- The Easiest Code to Debug Is That Which Is Not Written
- Use the Clients Language
- Don't Let the Cold Air In
- Never Be Silent
- Don't Speed Until You Know Where You Are Going

Features:
* ISBN13: 9780596008741
* Condition: New
My Personal Review:
Prefactoring is an exposition of principles for software design, laid out in the context of the development of a fictional application. I've never been into that particular style of writing about software design; in fact, it was the only thing I didn't like about Martin Fowler's Refactoring, which is the inspiration for both Prefactorings own fictional case study and its name as well.

When you've got one programming book named after another one, one reasonable idea is to compare the two. Prefactorings premise is that, instead of fixing your design afterwards -- an extremely terse summary of Refactoring -- you apply what you've learned from that in the past to build it with the right design principles from the get-go. That sounds like good common sense, and it is. Unfortunately, it really only makes sense in the context of a misunderstanding of Refactoring. Refactoring and debugging are different things. Its very common in software for people to use buzzwords and catchphrases as an alternative to thinking, and consequently, in certain organizations, you'll hear refactoring used as a synonym for debugging.

In fact, refactoring is supposed to happen during debugging -- but its also supposed to happen during the course of development, and in fact this is the preferred time to do it. Refactoring comes from agile development, specifically Extreme Programming, where the basic cycle is to write unit tests, write the simplest code that can possibly satisfy those tests, refactor that code, and then begin again with new unit tests. Refactoring can mean improving things during debugging, but much more importantly, what it really means is streamlining existing code as you refine it. To say that the best thing to learn from refactoring is to get your code right first time is to use the vocabulary of agile development to advocate waterfall development, and this, in fact, is what Prefactoring often seems to do.

Worse still, many of the code examples are in Java, and they don't use Josh Bloch's guidelines from Effective Java or Java Puzzlers. This might be a quibble, but I'd certainly hire or fire based on this quibble, as I think its very important (and therefore not a quibble at all). Pretty much everything I've done for months has been in Ruby on Rails, so I'm frequently reminded that Java is not popular in every sector of the tech industry -- however, if you are going to write Java, I personally feel that writing Java without observing Bloch's guidelines is careless at best, and borders on outright negligence.

On the other hand, I seem to be kind of eviscerating this book here, and thats not quite fair. I disagree with some of the design principles laid out in this book, but most of them are pretty strong in the common sense
department. Also, software development is one of those things where you can be better off after reading a book even if you disagree with it. For instance, just in criticizing this books attitude towards refactoring, Ive had to question my own understanding of it. If you read this book with the right frame of mind, youll challenge your own ideas and come to new conclusions, and probably become a better developer in the process.

For More 5 Star Customer Reviews and Lowest Price:
Prefactoring: Extreme Abstraction, Extreme Separation, Extreme Readability by Ken Pugh - 5 Star Customer Reviews and Lowest Price!