Foreword

Design Patterns described several ways to use patterns. Some people plan for patterns long before they write any code, while other people add a pattern after a lot of code has been written. The second way of using patterns is refactoring, because you are changing the design of the system without adding features or changing the external behavior. Some people put a pattern into a program because they think the pattern will make the program easier to change, but other people do it as a way to simplify the current design. If code has been written, then both of these are refactorings, because the first is refactoring to make a change easier, while the second is refactoring to clean up after a change.

Although a pattern is something you can see in a program, a pattern is also a program transformation. Each pattern can be explained by showing a program before the pattern was used and then afterwards. This is another way that patterns can be thought of as refactorings.

Unfortunately, many readers missed the connection between design patterns and refactoring. They thought of patterns as entirely related to design, not to code. I suppose that the name might have misled them, but the fact that the book was mostly C++ code should have indicated that patterns are about code as well as design, and adding a pattern usually requires changing code.

Joshua Kerievsky got the connection right away. I met him soon after he started the Design Patterns Study Group of New York City. He introduced the idea of a "before and after" study of a pattern, an example that shows the effect of a pattern on a system. Before he left town, his infectious enthusiasm had led to a group of over sixty people meeting several times a month. He started teaching patterns courses to companies and has taught them on-site, at his own location, and on the Internet. He has even taught other people how to teach them.

Joshua has gone on to become an XP practitioner and teacher as well. So, it is perfectly fitting that he has written a book that shows the connection between design patterns and refactoring, one of the core XP practices. Refactoring is not orthogonal to patterns—it is intimately related. Not all of the patterns that he talks about are from *Design Patterns*, but they all are in the style of *Design Patterns*. Joshua's book shows how patterns can help you design without causing you to create an up-front design.

If you practice what this book teaches, you will improve both your ability to create good designs and your ability to think about them.

—Ralph Johnson

Foreword

For several years now, I've been involved with advocating agile methods in general and extreme programming in particular. When I do, people often question how this fits in with my long-running interest in design patterns. Indeed, I've heard people claim that by encouraging refactoring and evolutionary design, I'm recanting what I've previously written about analysis and design patterns.

Yet all it takes is a quick look at people to realize that this view is flawed. Look at the leading members of the patterns community and at the leading members of the agile and XP communities, and you see a huge intersection. The truth is that patterns and evolutionary design have had a close relationship since their very beginnings.

Josh Kerievsky has been at the heart of this overlap. I first met him when he organized the successful patterns study groups in New York City. These groups did a collaborative study of the growing literature on design patterns. I quickly learned that Josh's understanding of design patterns was second to none, and I gained a lot of insight into those patterns by listening to him. Josh adopted refactoring early and was an extremely helpful reviewer on my book. As such it was no surprise to me that he also was a pioneer of extreme programming. His paper on patterns and extreme programming at the first XP conference is one of my favorites.

So if anyone is perfectly suited to write about the interplay of patterns and refactoring, Josh is. It's territory I explored a little bit in *Refactoring*, but I didn't take it too far because I wanted to concentrate on the basic refactorings. This book greatly expands that area, discussing in good detail how to evolve most of the popular patterns used in *Design Patterns* [DP], showing that they need not be designed into a system up front but can be evolved to as a system grows.

As well as the specific knowledge about these refactorings that you can gain from studying them, this book also tells you more about patterns and refactoring in general. Many people have said they find a refactoring approach to be a better way of learning about patterns because they see in gradual stages the interplay of problem and solution. These refactorings also reinforce the critical fact that refactoring is all about making large changes in tiny steps.

So I'm pleased to be able to present this book to you. I've spent a long time cajoling Josh to write a book and then working with him on this one. I'm delighted with the result, and I think you will be too.

—Martin Fowler