Foreword

Design Patterns: Elements of Reusable Object-Oriented Software, affectionately known by many as the "Gang of Four book" (GoF) is the first reference work on the topic to be published in a mainstream book. It has sold over half a million copies since 1995 and undoubtedly influenced the thoughts and code of millions of programmers worldwide. I still vividly remember buying my first copy of the book in the late nineties. Due in part to the enthusiasm with which it was recommended to me by my peers, I treated it as part of my coming-of-age as a programmer. I tore through the book in a few days, eagerly thinking up practical applications for each pattern.

It's commonly agreed that the most useful thing about patterns is the way in which they form a vocabulary for articulating design decisions during the normal course of development conversations among programmers. This is especially true during pair-programming, a cornerstone of Extreme Programming and other Agile processes, where design is an ongoing and shared activity. It's fantastically convenient to be able to say to your pair, "I think we need a strategy here" or "Let's add this functionality as an observer."

Knowledge of design patterns has even become an easy way to screen programming job candidates in some shops, where it's common to hear:

"What's your favorite pattern?"

"Um . . . factory?"

"Thanks for coming, there's the door."

Then again, the whole notion of having a *favorite* pattern is kind of strange isn't it? Our favorite pattern should be the one that applies to a given circumstance. One of the

classic mistakes made by inexperienced programmers just beginning to learn about patterns is to choose to implement a pattern as an end of its own, rather than as a means. Why do people get wrapped up in implementing patterns "just for fun" anyway?

At least in the statically typed world, there are a fair amount of technical challenges to tackle when you implement design patterns. At best, you use some ninja techniques that really show your coding prowess. Worst case scenario you end up with a bunch of boilerplate gunk. It makes the topic of design patterns a fun one, at least for programming geeks like me.

Are the GoF design patterns difficult to implement in Ruby? Not really. For starters, the absence of static typing lowers the code overhead involved in our programs overall. The Ruby standard library also makes some of the most common patterns available as one-line includes, and others are essentially built into the Ruby language itself. For instance, a Command object in the GoF sense is essentially a wrapper around some code that knows how to do one specific thing, to run a particular bit of code at some time. Of course, that is also a fairly accurate description of a Ruby code block object or a Proc.

Russ has been working with Ruby since 2002 and he knows that most experienced Rubyists already have a good grasp of design patterns and how to apply them. Thus his main challenge, as far as I can tell, was to write this book in such a way that it would be relevant and essential for professional Ruby programmers, yet still benefit newcomers to our beloved language. I think he has succeeded, and you will, too. Take the Command object example again: In its simple form it may be implemented with simply a block, but add state and a bit of behavior to it and now the implementation is not so simple anymore. Russ gives us proven advice that is specific to Ruby and instantly useful.

This book also has the added benefit of including new design patterns specific to Ruby that Russ has identified and explained in detail, including one of my favorite ones: Internal Domain Specific Languages. I believe that his treatment of the subject, as an evolution of the Interpreter pattern, is the first significant reference work in publication on the topic.

Finally, I think this book will hugely benefit those that are just beginning their professional careers in Ruby or migrating from languages such as PHP, where there isn't as much of a cultural emphasis on OO design and patterns. In the process of describing design patterns, Russ has captured the essence of solving many of the common programming hurdles that we face in day-to-day programming of significant Ruby programs—priceless information for newbies. So much so that I'm sure that this book will be a staple of my gift-list for new programmer colleagues and friends.

—Obie Fernandez, Professional Ruby Series Editor