Foreword

Jeff Langr has written a very interesting Java book in *Agile Java: Crafting Code with Test-Driven Development*. Its purpose is to teach new programmers the Java language, and to do so in the context of the best development approach that he and I know—namely Test-Driven Development (TDD). It's an undertaking of great potential value, and Jeff has done it nicely. It's my pleasure to provide this foreword and to recommend this book.

Agile Java isn't just for rank beginners. It's also a good book for bringing experienced programmers new to the Java language up to speed. It won't replace a certification manual or one of those huge books of "Everything Java." That's not its point. The point of *Agile Java* is to get you up to speed. Better yet, you come up to speed using TDD, which will serve you well in future learning and in your day-to-day work.

The book starts right off with object-oriented concepts and ideas. It helps if you know a bit about objects when you step in, but if you don't, hang on and you should pick up the basic ideas as you go along. Furthermore, every step of the way you'll be using the Test-Driven Development technique. If you haven't tried TDD, it may seem a bit odd at the beginning, but if you're like most of us who have given it a fair try, it will become a frequently-used tool in your kit.

If you already have Java and JUnit set up on your machine, dig right in. If not, be sure to use the "Setting Up" chapter to get your system correctly configured before moving on to the real examples. Once you can compile and run a simple Java program on your machine, you're ready to go.

Jeff asks you to type in the tests and example code, and I would echo that request. The TDD discipline is one that is learned by doing and practicing, not just by reading. You need to develop your own sense of the rhythm of development. Besides, typing in the examples from any programming book is the best way to learn what it has to offer.

In *Agile Java*, Jeff helps you build pieces of two applications. One of these relates to a student information system, the other focuses on playing chess. By the time you have worked through all the chapters, Jeff has introduced you to the basics of Java. Perhaps more importantly, you have met some of the most important deep capabilities, including interfaces, polymorphism, mock objects, reflection, multi-threading, and generics.

I found that Lesson 10, regarding the mathematical features of Java, particularly brought out the way I like to use tests as part of my learning of a new fea-

ture of a language or library. It's easy to read about something like BigDecimal and think, "I get it." For a while, I might even really get it. But when I encode my learning as tests, two things happen: First, I learn things about the topic that I would have missed if just reading about it. Writing the code pounds ideas into my thick head a bit better than just reading. Second, the tests record what I've learned as well as my thought process as I learned it. Because I've developed the habit of saving all my test cases, I can refer back to them and quickly refresh my memory. Often I'll even put a book and page reference into the tests as a comment, in case I want to go back later and dig out more.

Lesson 11, regarding I/O, includes a nice example of something I'm not very familiar with. Since I don't work in Java much, and most of the languages I commonly use don't have an equivalent, I'm not familiar with nested classes. Jeff gives a good example of when we would be well-advised to use nested classes, and shows how to use and test them.

As I write this foreword, I'm really getting into the book, because Jeff is taking me where I've not gone before. I like that. Lesson 12 is about Mock Objects, and the first example is one that we agile software developers encounter often: How can we deal in our incremental development with an external API that is fairly well defined but isn't available yet? Jeff shows us how to do this by defining the interface—from the documentation if necessary—and then by building a Mock Object that represents our understanding of what the API will do when we finally get it. Writing tests against our Mock Object gives us tests that we can use to ensure that the API does what we expect when we finally get the real code. An excellent addition to your bag of tricks!

Jeff is an educator, and a darn good one. Jeff wants us to think, and to work! He knows that if you and I are ready to learn, we have to practice: we have to do the work. His chapters have exercises. We are well-advised to think about all of them, and to do the ones that cover topics we aren't familiar with. That's how we'll really hammer these ideas into our heads. Read and study his examples, type them in to drill them into your mind, and then follow his lead as you work the interesting examples. You'll be glad you did.

Agile Java: Crafting Code with Test-Driven Development offers you at least three benefits: You'll learn things about Java that you probably didn't know, even if you're not an absolute beginner. You'll learn how to use test-driven development in a wide range of cases, including some in which you'd probably find difficult to invent on your own. And, through building up your skill, you'll add this valuable technique to your professional bag of tricks.

I enjoyed the book and found it valuable. I think you will, too. Enjoy!

Ron Jeffries
www.XProgramming.com
Pinckney, Michigan
November 2, 2004