foreword to the first edition

When Roy Osherove told me that he was working on a book about unit testing, I was very happy to hear it. The testing meme has been rising in the industry for years, but there has been a relative dearth of material available about unit testing. When I look at my bookshelf, I see books that are about test-driven development specifically and books about testing in general, but until now there has been no comprehensive reference for unit testing—no book that introduces the topic and guides the reader from first steps to widely accepted best practices. The fact that this is true is stunning. Unit testing isn't a new practice. How did we get to this point?

It's almost a cliché to say that we work in a very young industry, but it's true. Mathematicians laid the foundations of our work less than 100 years ago, but we've only had hardware fast enough to exploit their insights for the last 60 years. There was an initial gap between theory and practice in our industry, and we're only now discovering how it has impacted our field.

In the early days, machine cycles were expensive. We ran programs in batches. Programmers had a scheduled time slot, and they had to punch their programs into decks of cards and walk them to the machine room. If your program wasn't right, you lost your time, so you desk-checked your program with pencil and paper, mentally working out all of the scenarios, all of the edge cases. I doubt the notion of automated unit testing was even imaginable. Why use the machine for testing when you could use it to solve the problems it was meant to solve? Scarcity kept us in the dark.

Later, machines became faster and we became intoxicated with interactive computing. We could just type in code and change it on a whim. The idea of desk-checking

code faded away, and we lost some of the discipline of the early years. We knew programming was hard, but that just meant that we had to spend more time at the computer, changing lines and symbols until we found the magical incantation that worked.

We went from scarcity to surplus and missed the middle ground, but now we're regaining it. Automated unit testing marries the discipline of desk-checking with a newfound appreciation for the computer as a development resource. We can write automated tests, in the language we develop in, to check our work—not just once, but as often as we're able to run them. I don't think there is any other practice that's quite as powerful in software development.

As I write this, in 2009, I'm happy to see Roy's book come into print. It's a practical guide that will help you get started and also serve as a great reference as you go about your testing tasks. *The Art of Unit Testing* isn't a book about idealized scenarios. It teaches you how to test code as it exists in the field, how to take advantage of widely used frameworks, and, most importantly, how to write code that's far easier to test.

The Art of Unit Testing is an important title that should have been written years ago, but we weren't ready then. We are ready now. Enjoy.

MICHAEL FEATHERS