Foreword

Doing the right thing is rarely easy. Most of us should probably eat better, exercise more, and spend more time with our families. But each day, when confronted with the choice of being good or doing the easy thing, we often choose the easy thing because, well, it's easier.

It is the same way with software testing. We all know that we *should* spend more effort on testing, that more testing will make our code more reliable and maintainable, that our users will thank us if we do, that it will help us better understand our own programs, but each day when we sit down to the computer and choose between writing more tests and writing more application code, it is very tempting to reach for the easy choice.

Today, it is largely accepted that unit testing is the responsibility of developers, not QA, but this is a relatively recent development, one for which we can largely thank the JUnit testing framework. It is notable that JUnit had such an impact because there's not really very much to it—it's a simple framework, with not a lot of code. What enabled JUnit to change developer behavior where years of lecturing and guilt could not was that, for the first time, the pain of writing unit tests was reduced to a bearable level, making it practical for the merely responsible to include unit testing in our daily coding. Rather than make testing more desirable, which is not such an easy sell (eat those vegetables, they're good for you!), JUnit simply made it easier to do the right thing.

With all the righteousness of the newly converted, many developers proclaimed their zeal for testing, proudly calling themselves "test-infected." This is all well and good—few could argue that software developers were doing *too much* testing, so more is probably an improvement—but it is only the first step. There's more to testing than unit tests, and if we expect developers to take this next step we must provide testing tools that reduce the pain of creating them—and demand testability as a fundamental design requirement. If software engineering is ever to become a true engineering discipline, testing will form one of the critical pillars on which it will be built. (Perhaps one day, writing code without tests will be considered as professionally irresponsible as constructing a bridge without performing a structural analysis.)

This book is dedicated to the notion that we've only just begun our relationship with responsible testing. The TestNG project aims to help developers take the next step—the NG stands for "next generation"—enabling broader and deeper test coverage that encompasses not only unit tests but also acceptance, functional, and integration tests. Among other useful features, it provides a rich mechanism for specifying and parameterizing test suites, encompassing concurrent testing and a flexible mechanism for decoupling test code from its data source. (And, as proof that TestNG is succeeding, a number of its features have been adopted in more recent versions of JUnit.)

One challenge to more effective developer testing, no matter what tools are provided, is that writing effective tests requires different skills than writing effective code. But, like most skills, testing can be learned, and one of the best ways to learn is to watch how more experienced hands might do it. Throughout this book, Hani and Cédric share with you their favorite techniques for effectively testing Java applications and for designing applications and components for testability. (This last skill—designing for testability—is probably one of the most valuable lessons from this book. Designing code for testability forces you to think about the interactions and dependencies between components earlier, thereby encouraging you to build cleaner, more loosely coupled code.) Of course, the TestNG framework is used to illustrate these techniques, but even if you are not a TestNG user (and not interested in becoming one), the practical techniques presented here will help you to be a better tester and, in turn, a better engineer.

Brian Goetz
Senior Staff Engineer, Sun Microsystems