the art of

# UNIT TESTING

with Examples in .NET

MANNING

ROY OSHEROVE

## Early Praise for *The Art of Unit Testing*

"*The Art of Unit Testing* provides a thorough, incremental introduction to writing tests as part of the programming process. Programmers looking to write their first test will find easy-to-follow instructions, while those who have been testing for a while will find ideas for refining their technique"
—Kent Beck, Three Rivers Institute

"Beautifully crafted, detailed unit testing masterpiece. Bravo, Bravo, Bravo!"
—Mohammad Azam, Microsoft MVP, HighOnCoding

"This book tells all the truth about unit testing, even the unpleasant side of it."
—Franco Lombardo, Molteni Informatica

"Roy knows testing."
—Wendy Friedlander, Agile Solutions

"Unit testing, directly from theory to practice."
—Francesco Goggi, Software Engineer

# The Art of Unit Testing

## *with Examples in .NET*

Roy Osherove

MANNING

Greenwich
(74° w. long.)

For online information and ordering of this and other Manning books, please visit www.manning.com. The publisher offers discounts on this book when ordered in quantity. For more information, please contact:

Special Sales Department
Manning Publications Co.
Sound View Court 3B
Greenwich, CT 06830

fax: (609) 877-8256
email: manning@manning.com

©2009 by Manning Publications Co. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher. Figure 3.2 is reproduced with permission from NASA.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in the book, and Manning Publications was aware of a trademark claim, the designations have been printed in initial caps or all caps.

♾ Recognizing the importance of preserving what has been written, it is Manning's policy to have the books we publish printed on acid-free paper, and we exert our best efforts to that end. Recognizing also our responsibility to conserve the resources of our planet, Manning books are printed on paper that is at least 15% recycled and processed without the use of elemental chlorine.

Manning Publications Co.
209 Bruce Park Avenue
Greenwich, CT 06830

Development Editor: Nermina Miller
Copyeditor: Andy Carroll
Proofreader: Anna Welles
Typesetter: Marija Tudor
Cover designer: Leslie Haimes

ISBN: 978-1-933988-27-6

Printed in the United States of America
1 2 3 4 5 6 7 8 9 10 – MAL – 14 13 12 11 10 09

To my wife, Tal, and my sons,
Itamar and Aviv—my family

# Brief contents


PART 1 GETTING STARTED 1

1 ❍ The basics of unit testing 3

2 ❍ A first unit test 21

PART 2 CORE TECHNIQUES 47

3 ❍ Using stubs to break dependencies 49

4 ❍ Interaction testing using mock objects 82

5 ❍ Isolation (mock object) frameworks 99

PART 3 THE TEST CODE 139

6 ❍ Test hierarchies and organization 141

7 ❍ The pillars of good tests 171

PART 4 DESIGN AND PROCESS 217

8 ❍ Integrating unit testing into the organization 219

9 ❍ Working with legacy code 239

# Contents

*foreword* xv
*preface* xvii
*acknowledgments* xix
*about this book* xx
*about the cover illustration* xxiii

PART 1 GETTING STARTED 1

1 The basics of unit testing 3
1.1 *Unit testing—the classic definition* 4
The importance of writing "good" unit tests 5 ○ We've all written unit tests (sort of) 5
1.2 *Properties of a good unit test* 6
1.3 *Integration tests* 7
Drawbacks of integration tests compared to automated unit tests 9
1.4 *Good unit test—a definition* 11
1.5 *A simple unit test example* 12
1.6 *Test-driven development* 16
1.7 *Summary* 19

2 A first unit test 21
2.1 *Frameworks for unit testing* 22
What unit-testing frameworks offer 22 ○ The xUnit frameworks 25

2.2 Introducing the LogAn project 25
2.3 First steps with NUnit 26
Installing NUnit 26 ❍ Loading up the solution 26 ❍ Using the NUnit attributes in your code 29
2.4 Writing our first test 30
The Assert class 31 ❍ Running our first test with NUnit 32 ❍ Fixing our code and passing the test 33 ❍ From red to green 33
2.5 More NUnit attributes 34
Setup and teardown 34 ❍ Checking for expected exceptions 36 ❍ Ignoring tests 38 ❍ Setting test categories 39
2.6 Indirect testing of state 40
2.7 Summary 44

PART 2 CORE TECHNIQUES 47

3 Using stubs to break dependencies 49
3.1 Introducing stubs 50
3.2 Identifying a filesystem dependency in LogAn 51
3.3 Determining how to easily test LogAnalyzer 52
3.4 Refactoring our design to be more testable 55
Extract an interface to allow replacing underlying implementation 55 ❍ Inject stub implementation into a class under test 58 ❍ Receive an interface at the constructor level (constructor injection) 58 ❍ Receive an interface as a property get or set 64 ❍ Getting a stub just before a method call 66
3.5 Variations on refactoring techniques 74
Using Extract and Override to create stub results 75
3.6 Overcoming the encapsulation problem 77
Using internal and [InternalsVisibleTo] 78 ❍ Using the [Conditional] attribute 79 ❍ Using #if and #endif with conditional compilation 80
3.7 Summary 80

## 4 Interaction testing using mock objects 82

4.1 State-based versus interaction testing 83

4.2 The difference between mocks and stubs 84

4.3 A simple manual mock example 87

4.4 Using a mock and a stub together 89

4.5 One mock per test 94

4.6 Stub chains: stubs that produce mocks or other stubs 95

4.7 The problems with handwritten mocks and stubs 96

4.8 Summary 97

## 5 Isolation (mock object) frameworks 99

5.1 Why use isolation frameworks? 100

5.2 Dynamically creating a fake object 102

Introducing Rhino Mocks into your tests 102 ❍ Replacing a handwritten mock object with a dynamic one 103

5.3 Strict versus nonstrict mock objects 106

Strict mocks 106 ❍ Nonstrict mocks 107

5.4 Returning values from fake objects 108

5.5 Creating smart stubs with an isolation framework 110

Creating a stub in Rhino Mocks 110 ❍ Combining dynamic stubs and mocks 112

5.6 Parameter constraints for mocks and stubs 115

Checking parameters with string constraints 115 ❍ Checking parameter object properties with constraints 118 ❍ Executing callbacks for parameter verification 120

5.7 Testing for event-related activities 121

Testing that an event has been subscribed to 122 ❍ Triggering events from mocks and stubs 123 ❍ Testing whether an event was triggered 124

5.8 Arrange-act-assert syntax for isolation 126

5.9 Current isolation frameworks for .NET 130

NUnit.Mocks 130 ❍ NMock 131 ❍ NMock2 131 ❍ Typemock Isolator 132 ❍ Rhino Mocks 132 ❍ Moq 134

5.10 Advantages of isolation frameworks 134
5.11 Traps to avoid when using isolation frameworks 135
Unreadable test code 135 ❍ Verifying the wrong things 136 ❍ Having more than one mock per test 136 ❍ Overspecifying the tests 136
5.12 Summary 137

PART 3 THE TEST CODE 139

6 Test hierarchies and organization 141
6.1 Having automated builds run automated tests 142
Anatomy of an automated build 142 ❍ Triggering builds and continuous integration 144 ❍ Automated build types 144
6.2 Mapping out tests based on speed and type 145
The human factor of separating unit from integration tests 146 ❍ The safe green zone 147
6.3 Ensuring tests are part of source control 148
6.4 Mapping test classes to code under test 148
Mapping tests to projects 148 ❍ Mapping tests to classes 149 ❍ Mapping tests to specific methods 150
6.5 Building a test API for your application 150
Using test class inheritance patterns 151 ❍ Creating test utility classes and methods 167 ❍ Making your API known to developers 168
6.6 Summary 169

7 The pillars of good tests 171
7.1 Writing trustworthy tests 172
Deciding when to remove or change tests 172 ❍ Avoiding logic in tests 178 ❍ Testing only one thing 179 ❍ Making tests easy to run 180 ❍ Assuring code coverage 180
7.2 Writing maintainable tests 181
Testing private or protected methods 182 ❍ Removing duplication 184 ❍ Using setup methods in a maintainable manner 188 ❍ Enforcing test isolation 191 ❍ Avoiding multiple asserts 198 ❍

Avoiding testing multiple aspects of the same object 202 ○ Avoiding overspecification in tests 205

7.3 Writing readable tests 209

Naming unit tests 210 ○ Naming variables 211 ○ Asserting yourself with meaning 212 ○ Separating asserts from actions 214 ○ Setting up and tearing down 214

7.4 Summary 215

PART 4 DESIGN AND PROCESS 217

8 Integrating unit testing into the organization 219

8.1 Steps to becoming an agent of change 220

Be prepared for the tough questions 220 ○ Convince insiders: champions and blockers 220 ○ Identify possible entry points 222

8.2 Ways to succeed 223

Guerrilla implementation (bottom-up) 223 ○ Convincing management (top-down) 224 ○ Getting an outside champion 224 ○ Making progress visible 225 ○ Aiming for specific goals 227 ○ Realizing that there will be hurdles 228

8.3 Ways to fail 229

Lack of a driving force 229 ○ Lack of political support 229 ○ Bad implementations and first impressions 230 ○ Lack of team support 230

8.4 Tough questions and answers 231

How much time will this add to the current process? 231 ○ Will my QA job be at risk because of this? 233 ○ How do we know this is actually working? 234 ○ Is there proof that unit testing helps? 234 ○ Why is the QA department still finding bugs? 235 ○ We have lots of code without tests: where do we start? 235 ○ We work in several languages: is unit testing feasible? 236 ○ What if we develop a combination of software and hardware? 236 ○ How can we know we don't have bugs in our tests? 236 ○ I see in my debugger that my code works fine: why do I need tests? 237 ○ Must we do TDD-style coding? 237

8.5 Summary 238

**9 Working with legacy code 239**

*9.1 Where do you start adding tests? 240*

*9.2 Choosing a selection strategy 242*

Pros and cons of the easy-first strategy 242 ❍ Pros and cons of the hard-first strategy 243

*9.3 Writing integration tests before refactoring 244*

*9.4 Important tools for legacy code unit testing 246*

Isolate dependencies easily with Typemock Isolator 246 ❍ Find testability problems with Depender 248 ❍ Use JMockit for Java legacy code 248 ❍ Use Vise while refactoring your Java code 250 ❍ Use FitNesse for acceptance tests before you refactor 251 ❍ Read Michael Feathers' book on legacy code 253 ❍ Use NDepend to investigate your production code 253 ❍ Use ReSharper to navigate and refactor production code 253 ❍ Detect duplicate code (and bugs) with Simian 254 ❍ Detect threading issues with Typemock Racer 254

*9.5 Summary 254*

**Appendix A Design and testability 256**

**Appendix B Extra tools and frameworks 268**

**Index 284**

# Foreword

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
SENIOR CONSULTANT
OBJECT MENTOR

# Preface

One of the biggest failed projects I worked on had unit tests. Or so I thought. I was leading a group of programmers creating a billing application, and we were doing it in a fully test-driven manner—writing the test, then writing the code, seeing the test fail, making the test pass, refactoring, and starting all over again.

The first few months of the project were great. Things were going well, and we had tests that proved that our code worked. But as time went by, requirements changed. We were forced to change our code to fit those new requirements, and when we did, tests broke and had to be fixed. The code still worked, but the tests we wrote were so brittle that any little change in our code broke them, even though the code was working fine. It became a daunting task to change code in a class or method because we also had to fix all the related unit tests.

Worse yet, some tests became unusable because the people who wrote them left the project and no one knew how to maintain the tests, or what they were testing. The names we gave our unit-testing methods were not clear enough, and we had tests relying on other tests. We ended up throwing out most of the tests less than six months into the project.

The project was a miserable failure because we let the tests we wrote do more harm than good. They took more time to maintain and understand than they saved us in the long run, so we stopped using them. I moved on to other projects, where we did a better job writing our unit tests, and we had some great successes using them, saving huge amounts of debugging and integration time. Ever since that first failed project, I've been compil-

ing best practices for unit tests and using them on subsequent projects. I find a few more best practices with every project I work on.

Understanding how to write unit tests—and how to make them maintainable, readable, and trustworthy—is what this book is about, no matter what language or integrated development environment (IDE) you work with. This book covers the basics of writing a unit test, moves on to the basics of interaction testing, and then introduces best practices for writing, managing, and maintaining unit tests in the real world.

# Acknowledgments

A big thank you to Michael Stephens and Nermina Miller at Manning, who were patient with me every step of the long way in writing this book.

Thank you Jim Newkirk, Michael Feathers, Gerard Meszaros, and many others, who provided me with inspiration and the ideas that made this book what it is. And a special thank you to Michael for agreeing to write the foreword to the book.

The following reviewers read the manuscript at various stages during its development. I'd like to thank them for providing valuable feedback: Svetlana Christopher, Wendy Friedlander, Jay Flowers, Jean-Paul S. Boodhoo, Armand du Plessis, James Kovacs, Carlo Bottiglieri, Ken DeLong, Dusty Jewett, Lester Lobo, Alessandro Gallo, Gabor Paller, Eric Raymond, David Laribee, Christian Siegers, Phil Hanna, Josh Cronemeyer, Mark Seemann, Francesco Goggi, Franco Lambardo, Dave Nicolette, and Mohammad Azam. Thanks also to Rod Coffin, who did a technical proofread of the final manuscript shortly before it went to press.

A final word of thanks to the early readers of the book in Manning's Early Access Program for their comments in the online forum. You helped shape the book.

# About this book

## How to use this book

If you've never written unit tests before, this book is best read from start to finish so you get the full picture. If you already have experience, you should feel comfortable jumping into the chapters as you see fit.

## Who should read this book

The book is for anyone who writes code and is interested in learning best practices for unit testing. All the examples are written in C# using Visual Studio 2008, so .NET developers will find the examples particularly useful. But the lessons I teach apply equally to most, if not all, statically typed object-oriented languages (VB.NET, Java, and C++, to name a few). If you're a developer, team lead, QA engineer (who writes code), or novice programmer, this book should suit you well.

## Roadmap

The book is divided into four parts.

Part 1 takes you from zero to sixty in writing unit tests. Chapters 1 and 2 cover the basics, such as how to use a testing framework (NUnit), and introduce the basic automated test attributes, such as `[SetUp]` and `[TearDown]`. They also introduce the ideas of asserts, ignoring tests, and state-based testing.

Part 2 discusses advanced techniques for breaking dependencies: mock objects, stubs, mock frameworks, and patterns for refactoring your code to use them. Chapter 3 introduces the idea of stubs and shows how to

manually create and use them. Chapter 4 introduces interaction testing with handwritten mock objects. Chapter 5 merges these two concepts and shows how isolation (mock) frameworks combine these two ideas and allow them to be automated.

Part 3 talks about ways to organize test code, patterns for running and refactoring its structure, and best practices when writing tests. Chapter 6 discusses test hierarchies, how to use test infrastructure APIs, and how to combine tests in the automated build process. Chapter 7 discusses best practices in unit testing for creating maintainable, readable, and trustworthy tests.

Part 4 talks about how to implement change in an organization and how to work on existing code. Chapter 8 discusses problems and solutions you would encounter when trying to introduce unit testing into an organization. It also identifies and answers some questions you might be asked. Chapter 9 talks about introducing unit testing into existing code. It identifies a couple of ways to determine where to begin testing and discusses some tools for testing untestable code.

Finally, there are two appendixes. Appendix A discusses the loaded topic of designing for testability and the other alternatives that exist today. Appendix B has a list of tools you might find useful in your testing efforts.

## Code conventions and downloads

You can download the source code for this book from the book's site at www.ArtOfUnitTesting.com, as well as from the publisher's website at www.manning.com/TheArtofUnitTesting.

All source code in listings or in the text is in a `fixed-width font like this` to separate it from ordinary text. In listings, **`bold code`** indicates code that has changed from the previous example, or that will change in the next example.

Code annotations accompany some of the listings, highlighting important concepts. In some cases, numbered bullets link to explanations that follow in the text.

## Software requirements

To use the code in this book, you need at least Visual Studio C# Express (which is free), or a more advanced version of it (Professional or Team Suite, for example). You'll also need NUnit (an open source and free framework) and other tools that will be referenced where they're relevant. All the tools mentioned are either free, open source, or have trial versions you can use freely as you read this book.

## Author Online

The purchase of *The Art of Unit Testing* includes free access to a private forum run by Manning Publications where you can make comments about the book, ask technical questions, and receive help from the authors and other users. To access and subscribe to the forum, point your browser to www.manning.com/TheArtofUnitTesting. This page provides information on how to get on the forum once you are registered, what kind of help is available, and the rules of conduct in the forum.

Manning's commitment to our readers is to provide a venue where a meaningful dialogue between individual readers and between readers and the author can take place. It's not a commitment to any specific amount of participation on the part of the author, whose contribution to the book's forum remains voluntary (and unpaid). We suggest you try asking him some challenging questions, lest his interest stray!

The Author Online forum and the archives of previous discussions will be accessible from the publisher's website as long as the book is in print.

sample content of The Art of Unit Testing: With Examples in .Net

- read online Night Stalker book
- download AQA AS Chemistry Student Unit Guide : Unit 1 Foundation Chemistry book
- *download Eat Skinny, Be Skinny: 100 Wholesome and Delicious Recipes Under 300 Calories pdf, azw (kindle)*
- read online Kurtisane des Imperiums (Perry Rhodan Neo, Band 54; Arkon, Band 6) pdf, azw (kindle), epub

- http://qolorea.com/library/The-Codebreakers--The-Comprehensive-History-of-Secret-Communication-from-Ancient-Times-to-the-Internet.pdf
- http://yachtwebsitedemo.com/books/AQA-AS-Chemistry-Student-Unit-Guide---Unit-1-Foundation-Chemistry.pdf
- http://crackingscience.org/?library/Olivier.pdf
- http://twilightblogs.com/library/Pasta.pdf