



Roy
Osherove

The Art of
Unit Testing
Deutsche Ausgabe



Inhaltsverzeichnis

Vorwort	15
Einleitung	17
Über dieses Buch	18
Wie Sie dieses Buch verwenden	18
Wer dieses Buch lesen sollte	18
Meilensteine	18
Codekonventionen und Downloads	19
Softwareanforderungen	20
Danksagung	20
Teil I Erste Schritte	21
I Die Grundlagen des Unit Testings	23
I.1 Unit Testing – Die klassische Definition	23
I.1.1 Die Bedeutung »guter« Unit Tests	24
I.1.2 Wir alle haben schon Unit Tests geschrieben (irgendwie)	25
I.2 Eigenschaften eines »guten« Unit Tests	26
I.3 Integration Tests	27
I.3.1 Nachteile von Integration Tests im Vergleich zu automatisierten Unit Tests	29
I.4 »Gute« Unit Tests – Eine Definition	31
I.5 Ein einfaches Unit-Test-Beispiel	32
I.6 Testgetriebene Entwicklung	36
I.7 Zusammenfassung	39
2 Ein erster Unit Test	41
2.1 Frameworks für das Unit Testing	42
2.1.1 Was Unit Testing Frameworks bieten	42
2.1.2 Die xUnit Frameworks	45
2.2 Das LogAn-Projekt wird vorgestellt	45

2.3	Die ersten Schritte mit NUnit.	45
2.3.1	Die Installation von NUnit.	46
2.3.2	Das Laden der Projektmappe.	47
2.3.3	Die Verwendung der NUnit-Attribute in Ihrem Code.	49
2.4	Wir schreiben unseren ersten Test.	50
2.4.1	Die Klasse Assert	50
2.4.2	Wir führen unseren ersten Test mit NUnit aus.	51
2.4.3	Fehlerbehebung und ein erfolgreicher Testlauf.	52
2.4.4	Von Rot nach Grün	53
2.5	Weitere NUnit-Attribute	53
2.5.1	Aufbau und Abbau	53
2.5.2	Auf erwartete Ausnahmen prüfen	56
2.5.3	Das Ignorieren von Tests	58
2.5.4	Das Festlegen der Testkategorien	58
2.6	Indirekte Zustandstests.	59
2.7	Zusammenfassung	63

Teil II Zentrale Methoden 65

3	Die Verwendung von Stubs, um Abhängigkeiten aufzulösen	67
3.1	Die Stubs werden vorgestellt.	67
3.2	Die Identifizierung einer Dateisystemabhängigkeit in LogAn.	68
3.3	Die Entscheidung, wie LogAnalyzer am einfachsten getestet werden kann.	70
3.4	Design-Refactoring zur Verbesserung der Testbarkeit.	73
3.4.1	Extrahiere ein Interface, um die dahinter liegende Implementierung durch eine andere ersetzen zu können	73
3.4.2	Injiziere eine Stub-Implementierung in die zu testende Klasse	76
3.4.3	Übergebe dem Konstruktor ein Interface (Constructor Injection).	76
3.4.4	Übergebe einer Property ein Interface	82
3.4.5	Hole einen Stub unmittelbar vor einem Methodenaufruf	84
3.5	Variationen der Refactoring-Technik.	92
3.5.1	Die Verwendung von Extract and Override, um Stub-Resultate zu erzeugen.	93

3.6	Die Überwindung des Kapselungsproblems	95
3.6.1	Die Verwendung von internal und [InternalsVisibleTo]	96
3.6.2	Die Verwendung des Attributs [Conditional]	96
3.6.3	Die Verwendung von #if und #endif zur bedingten Kompilierung	97
3.7	Zusammenfassung	98
4	Interaction Testing mit Mock-Objekten	99
4.1	Zustandsbasiertes Testen gegenüber Interaction Testing	99
4.2	Der Unterschied zwischen Mocks und Stubs	101
4.3	Ein einfaches manuelles Mock-Beispiel	103
4.4	Die gemeinsame Verwendung von Mock und Stub	106
4.5	Ein Mock pro Test	111
4.6	Stub-Ketten: Stubs, die Mocks oder andere Stubs erzeugen	111
4.7	Die Probleme mit handgeschriebenen Mocks und Stubs	112
4.8	Zusammenfassung	113
5	Isolation (Mock-Objekt)-Frameworks	115
5.1	Warum überhaupt Isolation Frameworks?	116
5.2	Das dynamische Erzeugen eines Fake-Objekts	118
5.2.1	Die Einführung von Rhino Mocks in Ihre Tests	118
5.2.2	Das Ersetzen eines handgeschriebenen Mock-Objekts durch ein dynamisches	119
5.3	Strikte und nicht-strikte Mock-Objekte	122
5.3.1	Strikte Mocks	122
5.3.2	Nicht-strikte Mocks	122
5.4	Die Rückgabe von Werten aus Fake-Objekten	124
5.5	Das Erzeugen schlauer Stubs mit einem Isolation Framework	126
5.5.1	Das Erzeugen eines Stubs in Rhino Mocks	126
5.5.2	Die Kombination von dynamischen Stubs und Mocks	128
5.6	Parameter Constraints für Mocks und Stubs	131
5.6.1	Das Überprüfen von Parametern mit String Constraints	131
5.6.2	Das Überprüfen von Parameter Object Properties mit Constraints	133
5.6.3	Die Ausführung von Callbacks zur Parameterverifikation	135
5.7	Das Testen auf Event-bezogene Aktivitäten	136
5.7.1	Wir testen, ob ein Event abonniert wurde	137
5.7.2	Das Triggern von Events aus Mocks und Stubs heraus	138
5.7.3	Wir testen, ob ein Event ausgelöst wurde	139

5.8	Die Arrange-Act-Assert-Syntax für die Isolation	141
5.9	Die aktuellen Isolation Frameworks für .NET.	144
5.9.1	NUnit.Mocks.	145
5.9.2	NMock.	146
5.9.3	NMock2.	146
5.9.4	Typemock Isolator	146
5.9.5	Rhino Mocks.	147
5.9.6	Moq	148
5.10	Die Vorteile von Isolation Frameworks	149
5.11	Mögliche Fallstricke bei der Verwendung von Isolation Frameworks	149
5.11.1	Unlesbarer Testcode.	150
5.11.2	Die Verifizierung der falschen Dinge.	150
5.11.3	Die Verwendung von mehr als einem Mock pro Test.	150
5.11.4	Die Überspezifizierung von Tests.	150
5.12	Zusammenfassung	151

Teil III Der Testcode 153

6	Testhierarchie und Organisation	155
6.1	Mit automatisierten Builds automatisierte Tests laufen lassen	155
6.1.1	Die Anatomie eines automatisierten Builds.	156
6.1.2	Das Anstoßen von Builds und die kontinuierliche Integration.	158
6.1.3	Automatisierte Build-Typen.	158
6.2	Testentwürfe, die auf Geschwindigkeit und Typ basieren	159
6.2.1	Der menschliche Faktor beim Trennen von Unit und Integration Tests.	160
6.2.2	Die sichere grüne Zone	161
6.3	Stellen Sie sicher, dass die Tests zu Ihrer Quellcodekontrolle gehören.	161
6.4	Das Abbilden der Testklassen auf den zu testenden Code.	162
6.4.1	Das Abbilden von Tests auf Projekte	162
6.4.2	Das Abbilden von Tests auf Klassen	162
6.4.3	Das Abbilden von Tests auf bestimmte Methoden	164
6.5	Wir bauen eine Test-API für Ihre Applikation	164
6.5.1	Die Verwendung von Testklassen-Vererbungsmustern	165

6.5.2	Der Entwurf von Test-Hilfsklassen und -Hilfsmethoden	181
6.5.3	Machen Sie Ihre API den Entwicklern bekannt	182
6.6	Zusammenfassung	183
7	Die Säulen guter Tests	185
7.1	Das Schreiben vertrauenswürdiger Tests	185
7.1.1	Die Entscheidung, wann Tests entfernt oder geändert werden.	186
7.1.2	Vermeiden Sie Logik in Tests	191
7.1.3	Testen Sie nur eine Sache	192
7.1.4	Machen Sie es leicht, die Tests auszuführen	193
7.1.5	Stellen Sie die Code-Abdeckung sicher	193
7.2	Das Schreiben wartbarer Tests	195
7.2.1	Das Testen privater oder geschützter Methoden	195
7.2.2	Das Entfernen von Duplizitäten	197
7.2.3	Die Verwendung von Setup-Methoden in einer wartbaren Art und Weise.	201
7.2.4	Das Erzwingen der Test-Isolierung	204
7.2.5	Vermeiden Sie mehrfache Asserts	211
7.2.6	Vermeiden Sie es, mehrere Aspekte des gleichen Objekts zu testen	215
7.2.7	Vermeiden Sie eine Überspezifizierung der Tests	218
7.3	Das Schreiben lesbarer Tests	222
7.3.1	Die Benennung der Unit Tests	222
7.3.2	Die Benennung der Variablen.	223
7.3.3	Benachrichtigen Sie sich sinnvoll	225
7.3.4	Das Trennen der Asserts von den Aktionen	226
7.3.5	Aufbauen und Abreißen	227
7.4	Zusammenfassung	227
Teil IV Design und Durchführung		229
8	Die Integration von Unit Tests in das Unternehmen.	231
8.1	Schritte, um ein Agent des Wandels zu werden	231
8.1.1	Seien Sie auf die schweren Fragen vorbereitet	232
8.1.2	Überzeugen Sie Insider: Champions und Blockierer	232
8.1.3	Identifizieren Sie mögliche Einstiegspunkte	234

8.2	Wege zum Erfolg	235
8.2.1	Guerilla-Implementierung (Bottom-up)	235
8.2.2	Überzeugen Sie das Management (Top-down)	236
8.2.3	Holen Sie einen externen Champion	236
8.2.4	Machen Sie Fortschritte sichtbar.	237
8.2.5	Streben Sie bestimmte Ziele an	239
8.2.6	Machen Sie sich klar, dass es Hürden geben wird	240
8.3	Wege zum Misserfolg	241
8.3.1	Mangelnde Triebkraft.	241
8.3.2	Mangelnde politische Unterstützung	241
8.3.3	Schlechte Implementierungen und erste Eindrücke.	242
8.3.4	Mangelnde Teamunterstützung	242
8.4	Schwierige Fragen und Antworten.	243
8.4.1	Wie viel zusätzliche Zeit wird für den aktuellen Prozess benötigt?	243
8.4.2	Ist deswegen mein Job bei der QS in Gefahr?	245
8.4.3	Woher wissen wir, dass es wirklich funktionieren wird?	245
8.4.4	Gibt es denn einen Beweis, dass Unit Testing hilft?	246
8.4.5	Warum findet die QS immer noch Bugs?	246
8.4.6	Wir haben eine Menge Code ohne Tests: Wo fangen wir an?	247
8.4.7	Wir arbeiten mit mehreren Sprachen: Ist Unit Testing da praktikabel?	248
8.4.8	Was ist, wenn wir eine Kombination aus Soft- und Hardware entwickeln?	248
8.4.9	Wie können wir wissen, dass wir keine Bugs in unseren Tests haben?	248
8.4.10	Mein Debugger zeigt mir, dass mein Code funktioniert: Wozu brauche ich Tests?	248
8.4.11	Müssen wir Code im TDD-Stil schreiben?	249
8.5	Zusammenfassung	249
9	Der Umgang mit Legacy Code	251
9.1	Wo soll man mit dem Einbauen der Tests beginnen?	252
9.2	Bestimmen Sie eine Auswahlstrategie.	254
9.2.1	Vor- und Nachteile der Strategie »Einfaches zuerst«.	254
9.2.2	Vor- und Nachteile der Strategie »Schwieriges zuerst«.	255

9.3	Schreiben Sie Integration Tests, bevor Sie mit dem Refactoring beginnen	256
9.4	Wichtige Tools für das Unit Testing von Legacy Code	257
9.4.1	Abhängigkeiten isolieren Sie leicht mit Typemock Isolator	258
9.4.2	Testbarkeitsprobleme finden Sie mit Dependy	259
9.4.3	Verwenden Sie JMockit für Java-Legacy-Code	260
9.4.4	Verwenden Sie Vise beim Refactoring Ihres Java-Codes	262
9.4.5	Verwenden Sie FitNesse für Akzeptanztests, bevor Sie mit dem Refactoring beginnen	263
9.4.6	Lesen Sie das Buch von Michael Feathers zu Legacy Code	264
9.4.7	Verwenden Sie NDepend, um Ihren Produktionscode zu untersuchen	264
9.4.8	Verwenden Sie ReSharper für die Navigation und das Refactoring des Produktionscodes	265
9.4.9	Spüren Sie Code-Duplikate (und Bugs) mit Simian auf	265
9.4.10	Spüren Sie Threading-Probleme mit Typemock Racer auf	266
9.5	Zusammenfassung	266
A	Design und Testbarkeit	267
A.1	Warum sollte ich mir Gedanken um die Testbarkeit in meinem Design machen?	267
A.2	Designziele für die Testbarkeit	268
A.2.1	Deklarieren Sie Methoden standardmäßig als virtuell	269
A.2.2	Benutzen Sie ein Interface-basiertes Design	270
A.2.3	Deklarieren Sie Klassen standardmäßig als nicht versiegelt	271
A.2.4	Vermeiden Sie es, konkrete Klassen innerhalb von Methoden mit Logik zu instanziiieren	271
A.2.5	Vermeiden Sie direkte Aufrufe von statischen Methoden	271
A.2.6	Vermeiden Sie Konstruktoren und statische Konstruktoren, die Logik enthalten	272
A.2.7	Trennen Sie die Singleton-Logik und Singleton-Halter	272
A.3	Vor- und Nachteile des Designs zum Zwecke der Testbarkeit	274
A.3.1	Arbeitsumfang	274
A.3.2	Komplexität	275

A.3.3	Das Preisgeben von sensiblem IP	275
A.3.4	Manchmal geht's nicht.	276
A.4	Alternativen des Designs zum Zwecke der Testbarkeit	276
A.5	Zusammenfassung	277
B	Tools und Frameworks	279
B.1	Isolation Frameworks	279
B.1.1	Moq	280
B.1.2	Rhino Mocks	280
B.1.3	Typemock Isolator	280
B.1.4	NMock	281
B.1.5	NUnit.Mocks	281
B.2	Test Frameworks	281
B.2.1	Microsofts Unit Testing Framework	282
B.2.2	NUnit	282
B.2.3	MbUnit	283
B.2.4	Gallio	283
B.2.5	xUnit	283
B.2.6	Pex	284
B.3	Dependency Injection und IoC-Container	284
B.3.1	StructureMap	285
B.3.2	Microsoft Unity	285
B.3.3	Castle Windsor	285
B.3.4	Autofac	286
B.3.5	Common Service Locator Library	286
B.3.6	Spring.NET	286
B.3.7	Microsoft Managed Extensibility Framework	287
B.3.8	Ninject	287
B.4	Datenbanktests	287
B.4.1	Verwenden Sie Integration Tests für Ihre Datenschicht	287
B.4.2	Verwenden Sie Rollback-Attribute	288
B.4.3	Verwenden Sie TransactionScope für ein Rollback	288
B.5	Webtests	289
B.5.1	Ivonna	289
B.5.2	Team System Web Test	289
B.5.3	NUnitAsp	290
B.5.4	Watir	290

B.5.5	WatiN	290
B.5.6	Selenium	290
B.6	UI-Tests	291
B.6.1	NUnitForms	291
B.6.2	Project White	291
B.6.3	Visual Studio UI Test	292
B.7	Thread-bezogene Tests	292
B.7.1	Typemock Racer	292
B.7.2	Microsoft CHESSE	293
B.7.3	Osherove.ThreadTester	293
B.8	Akzeptanztests	293
B.8.1	FitNesse	294
B.8.2	StoryTeller	294
C	Leitfaden Test-Review	295
	Stichwortverzeichnis	297