



Roy
Osherove

The Art of
Unit Testing
Deutsche Ausgabe





Vorwort

Als mir Roy Osherove erzählte, er würde an einem Buch über das Unit Testing arbeiten, war ich sehr froh, dies zu hören. Das Test-Mem ist über die Jahre in der Industrie gewachsen, aber es gibt einen relativen Mangel an Material zum Unit Testing. Wenn ich mein Bücherregal betrachte, dann sehe ich Bücher zur testgetriebenen Entwicklung im Besonderen und Bücher zum Testen im Allgemeinen, aber bis jetzt gab es keine umfassende Referenz zum Unit Testing – kein Buch, das in das Thema einführt, den Leser an die Hand nimmt und von den ersten Schritten bis zu den allgemein akzeptierten besten Vorgehensweisen führt. Diese Tatsache ist verblüffend. Unit Testing ist keine neue Praxis. Wie konnte es dazu kommen?

Es ist beinahe ein Klischee, wenn man sagt, dass wir in einer neuen Industrie arbeiten, aber es ist wahr. Mathematiker legten die Grundlagen unserer Arbeit vor weniger als 100 Jahren, aber erst seit 60 Jahren wir haben die Hardware, die schnell genug ist, um ihre Erkenntnisse auch nutzen zu können. In unserer Industrie bestand von Anfang an eine Lücke zwischen Theorie und Praxis und erst jetzt entdecken wir, wie sich das auf unser Fachgebiet ausgewirkt hat.

In den frühen Jahren waren Rechenzeiten teuer. Wir ließen die Programme im Batchbetrieb laufen. Programmierer hatten ein planmäßiges Zeitfenster, sie mussten ihre Programme in Kartenstapel stanzen und in den Maschinenraum tragen. War ein Programm nicht in Ordnung, hatten sie ihre Zeit vergeudet, also überprüften sie es zuvor mit Papier und Stift am Schreibtisch und gingen in Gedanken alle möglichen Szenarien, alle Grenzfälle durch. Ich bezweifle, dass der Begriff des automatisierten Unit Testings zu dieser Zeit überhaupt vorstellbar war. Warum sollte man die Maschine zum Testen nutzen, wo sie doch zur Lösung bestimmter Probleme gebaut worden war? Der Mangel hielt uns in der Dunkelheit gefangen.

Später wurden die Maschinen schneller und wir berauschten uns am interaktiven Computing. Wir konnten einfach Code eintippen und ihn aus Jux und Dollerei ändern. Die Idee, den Code am Schreibtisch zu überprüfen, schwand allmählich dahin und wir verloren etwas von der Disziplin der frühen Jahre. Wir wussten, dass Programmieren schwierig war, aber das bedeutete nur, dass wir mehr Zeit am Computer verbringen mussten und Zeilen und Symbole änderten, bis wir die magische Zauberformel, die funktionierte, gefunden hatten.

Wir kamen vom Mangel zum Überfluss und verpassten den Mittelweg, aber nun erobern wir ihn zurück. Automatisiertes Unit Testing verbindet die Disziplin der Schreibtischprüfung mit einer neu gefundenen Wertschätzung des Computers als einer Entwicklungsressource. Wir können automatische Tests in der Sprache schreiben, in der wir entwickeln, um unsere Arbeit zu überprüfen – und das nicht nur einmal, sondern so oft wir in der Lage sind, die Tests laufen zu lassen. Ich glaube nicht, dass es irgendein anderes Verfahren in der Softwareentwicklung gibt, das derartig mächtig ist.

Während ich dies im Jahre 2009 schreibe, bin ich froh zu sehen, wie Roys Buch in den Druck geht. Es ist eine praktische Anleitung, die Ihnen helfen wird, loszulegen, und es ist auch eine großartige Referenz, wenn Sie Ihre Aufgaben beim Testing angehen. *The Art of Unit Testing* ist kein Buch über idealisierte Szenarien. Es zeigt Ihnen, wie man Code testet, der im praktischen Einsatz existiert, wie man Nutzen aus weit verbreiteten Frameworks zieht und – was das Wichtigste ist – wie man Code schreibt, der wesentlich einfacher zu testen ist.

The Art of Unit Testing ist ein wichtiger Titel, der schon vor Jahren hätte geschrieben werden sollen, aber damals waren wir noch nicht bereit dafür. Jetzt sind wir bereit. Genießen Sie es!

Michael Feathers

Senior Consultant

Object Mentor



Einleitung

Eines der größten Projekte, an dem ich mitgearbeitet habe und das schiefgelaufen ist, beinhaltete auch Unit Tests. Das dachte ich zumindest. Ich leitete eine Gruppe von Entwicklern, die an einer Abrechnungssoftware arbeiteten, und wir machten es komplett auf die testgetriebene Weise – wir schrieben zuerst die Tests, dann den Code, die Tests schlugen fehl, wir sorgten dafür, dass sie erfolgreich verliefen, führten ein Refactoring durch und fingen wieder von vorne an.

Die ersten paar Monate des Projekts waren großartig. Alles lief gut und wir hatten Tests, die belegten, dass unser Code funktionierte. Aber im Laufe der Zeit änderten sich die Anforderungen. Wir waren also gezwungen, unseren Code zu ändern, um diesen neuen Anforderungen gerecht zu werden, doch als wir das taten, wurden die Tests unzuverlässig und mussten nachgebessert werden. Der Code funktionierte immer noch, aber die Tests, die wir dafür geschrieben hatten, waren so zerbrechlich, dass jede kleine Änderung in unserem Code sie überforderte, auch wenn der Code selbst prima funktionierte. Die Änderung des Codes in einer Klasse oder Methode wurde zu einer gefürchteten Aufgabe, weil wir auch alle damit zusammenhängenden Unit Tests entsprechend anpassen mussten.

Schlimmer noch, einige Tests wurden sogar unbrauchbar, weil die Leute, die sie geschrieben hatten, das Projekt verließen und keiner wusste, wie deren Tests zu pflegen waren oder was sie eigentlich testeten. Die Namen, die wir unseren Unit-Testing-Methoden gaben, waren nicht aussagekräftig genug und wir hatten Tests, die auf anderen Tests aufbauten. Schließlich warfen wir nach weniger als 6 Monaten Projektlaufzeit die meisten der Tests wieder heraus.

Das Projekt war ein erbärmlicher Fehlschlag, weil wir zugelassen hatten, dass die Tests, die wir schrieben, mehr schadeten als nützten. Langfristig brauchten wir mehr Zeit, um sie zu pflegen und zu verstehen, als sie uns einsparten. Also hörten wir auf, sie einzusetzen. Ich machte mit anderen Projekten weiter und dort haben wir unsere Arbeit beim Schreiben der Unit Tests besser erledigt. Ihr Einsatz brachte uns großen Erfolg und sparte eine Menge Zeit beim Debuggen und bei der Integration. Seitdem dieses erste Projekt fehlschlug, trage ich bewährte Vorgehensweisen für das Unit Testing zusammen und wende sie auch im nachfolgenden Projekt an. Im Laufe jedes Projekts, an dem ich arbeite, entdecke ich weitere gute Vorgehensweisen.

Zu verstehen, wie man Unit Tests schreibt – und wie man sie wartbar, lesbar und vertrauenswürdig macht – ist das, wovon dieses Buch handelt. Egal, welche Sprache oder welche integrierte Entwicklungsumgebung (IDE) Sie verwenden. Dieses Buch behandelt die Grundlagen für das Schreiben von Unit Tests, geht dann auf die Grundlagen des Interaction Testings ein und stellt schließlich bewährte Vorgehensweisen für das Schreiben, das Verwalten und das Warten der Unit Tests in echten Projekten vor.

Über dieses Buch

Wie Sie dieses Buch verwenden

Wenn Sie noch nie Unit Tests geschrieben haben, dann lesen Sie das Buch am besten vom Anfang bis zum Ende durch, um einen kompletten Überblick zu erhalten. Sollten Sie bereits Erfahrung im Umgang mit Unit Tests haben, dann springen Sie ruhig direkt in die Kapitel, die Sie interessieren.

Wer dieses Buch lesen sollte

Dieses Buch ist für jeden geeignet, der Code entwickelt und daran interessiert ist, die besten Methoden für das Unit Testing zu erlernen. Alle Beispiele sind mit Visual Studio in C# geschrieben, so dass die Beispiele insbesondere für .NET Entwickler nützlich sein werden. Aber was ich unterrichte passt genauso gut auf die meisten, wenn nicht auf alle statisch typisierten, objektorientierten Sprachen (VB.NET, Java und C++, um nur ein paar zu nennen). Egal, ob Sie ein Entwickler sind, ein Teamleiter, ein QS-Ingenieur (der Code schreibt) oder ein Anfänger in der Programmierung, dieses Buch wurde für Sie geschrieben.

Meilensteine

Das vorliegende Buch ist in vier Hauptteile gegliedert.

Teil 1 bringt Sie beim Schreiben von Unit Tests von 0 auf 100. Kapitel 1 und 2 beschäftigen sich mit den Grundlagen, wie etwa der Verwendung eines Test Frameworks (NUnit), und führen die grundlegenden Testattribute ein, wie z.B. [SetUp] und [TearDown]. Darüber hinaus werden hier auch die Prinzipien der Assertion, des Ignorierens von Tests und des zustandsbasierten Testens erläutert.

Teil 2 diskutiert fortgeschrittene Methoden zum Auflösen von Abhängigkeiten: Mock-Objekte, Stubs, Mock Frameworks und Muster zum Refactoring Ihres Codes, um diese nutzen zu können. Kapitel 3 stellt das Konzept der Stubs vor und veranschaulicht, wie man sie von Hand erzeugen und benutzen kann. In Kapitel 4 wird das Interaction Testing mit handgeschriebenen Mock-Objekten beschrieben. Kapitel 5 führt diese beiden Konzepte schließlich zusammen und zeigt, wie sie

sich mithilfe von Isolation (Mock) Frameworks kombinieren lassen und ihre Automatisierung erlauben.

Teil 3 beschäftigt sich mit verschiedenen Möglichkeiten, den Testcode zu organisieren, mit Mustern, um ihn auszuführen und seine Strukturen umzubauen (Refactoring) und mit bewährten Methoden zum Schreiben von Tests. In Kapitel 6 werden Testhierarchien vorgestellt und es wird dargestellt, wie Testinfrastruktur-APIs verwendet und wie Tests in den automatisierten Build-Prozess eingebunden werden. Kapitel 7 diskutiert bewährte Vorgehensweisen beim Unit Testing, um wartbare, lesbare und vertrauenswürdige Tests zu entwickeln.

Teil 4 beschäftigt sich damit, wie sich Veränderungen in einem Unternehmen umsetzen lassen und wie man mit existierendem Code umgehen kann. In Kapitel 8 werden Probleme und Lösungen im Zusammenhang mit der Einführung des Unit Testings in einem Unternehmen diskutiert. Dabei werden auch einige Fragen beantwortet, die Ihnen in diesem Zusammenhang vielleicht gestellt werden. Kapitel 9 behandelt die Integration des Unit Testings in vorhandenen Code. Es werden mehrere Möglichkeiten aufgezeigt, um festzulegen, wo mit dem Testen begonnen werden sollte und es werden mehrere Tools vorgestellt, um »untestbaren« Code zu testen.

Schließlich gibt es auch noch zwei Anhänge. Anhang A diskutiert das vorbelastete Thema des Designs um der Testbarkeit willen und die Alternativen, die derzeit existieren. Anhang B listet eine Reihe von Tools auf, die Sie bei Ihren Testanstrengungen hilfreich finden könnten. Anhang C enthält einen Leitfaden für Test-Reviews.

Codekonventionen und Downloads

Sie können den Quellcode von der Verlagsseite unter www.mitp.de/9023 herunterladen oder ebenfalls von der Seite www.ArtOfUnitTesting.com, wo Sie auch weitere englischsprachige Informationen zum Buch, zu dessen Themen und zum Autor finden.

Sämtliche in diesem Buch enthaltenen Listings sind vom normalen Text klar zu unterscheiden und in einer anderen Schriftart wie dieser gesetzt. Manche Passagen sind **fett** gedruckt, um sie besonders hervorzuheben oder auch mit einem besonderen Hinweissymbol gekennzeichnet, wenn sie gesondert referenziert werden¹.

¹ Anmerkung des Übersetzers: Die Kommentare innerhalb der Listings und auch der größte Teil der Strings wurden – der größeren Klarheit und der besseren Lesbarkeit wegen – übersetzt. In dieser Hinsicht unterscheiden sich die abgedruckten Listings vom Quelltext, den Sie herunterladen können.

Softwareanforderungen

Um den Code in diesem Buch benutzen zu können, benötigen Sie zumindest Visual Studio C# Express (welches kostenlos ist) oder die umfangreicheren Versionen (wie Visual Studio 2010 Professional²). Darüber hinaus sind auch NUnit (ein freies Open Source Framework) und andere Tools erforderlich, auf die an den entsprechenden Stellen hingewiesen wird. Alle erwähnten Tools sind entweder kostenlos, als Open-Source-Versionen oder aber als Testversionen verfügbar, die Sie ausprobieren können, während Sie dieses Buch lesen.

Danksagung

Ein großes Dankeschön geht an Michael Stephens und Nermina Miller von Manning Publications für ihre Geduld auf dem langen Weg beim Schreiben dieses Buches.

Dank gebührt auch Jim Newkirk, Michael Feathers, Gerard Meszaros und vielen anderen, die mich mit Inspiration und Ideen unterstützt und dieses Buch zu dem gemacht haben, was es ist. Besonders danke ich dir, Michael, dafür, dass du es übernommen hast, das Vorwort für dieses Buch zu schreiben.

Die folgenden Rezensenten haben das Manuskript in verschiedenen Phasen seiner Entwicklung gelesen. Ich möchte ihnen für ihre wertvollen Anmerkungen danken: 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 Larabee, Christian Siegers, Phil Hanna, Josh Cronemeyer, Mark Seemann, Francesco Goggi, Franco Lambardo, Dave Nicolette, und Mohammad Azam. Dank auch an Rod Coffin, der das abschließende Manuskript vor der Drucklegung Korrektur gelesen hat.

Ein abschließendes Wort des Dankes auch an die ersten Leser des Buches in Mannings »Early Access Program« für ihre Kommentare im Online-Forum. Sie haben dabei geholfen, dem Buch seine endgültige Form zu geben.

² Anmerkung des Übersetzers: Zum Zeitpunkt der Veröffentlichung des amerikanischen Originaltitels war VS 2008 die aktuelle Version von Microsoft, auf die auch in diesem Buch Bezug genommen wurde. Hier wurden in der vorliegenden deutschen Ausgabe Verweise auf die inzwischen veröffentlichte Version 2010 ergänzt oder ersetzt.