Wahlfach Software-Qualität

# Mocking unter C#/.NET

Mit dem Schwerpunkt RhinoMocks

Alexander Busse
08.02.2011

# Inhaltsverzeichnis

# Abbildungsverzeichnis



# Listingverzeichnis

# 1 NMock

## 1.1 Homepage
http://www.nmock.org/

## 1.2 Lizenz
Das Framwork steht zur freien Verfügung unter dem Vorbehalt für keine entstehenden Schäden zu haften, die durch NMocks oder ein Modifikation von NMocks enstehen.

## 1.3 Kurzbeschreibung
Es handelt sich um eine dynamische Bibliothek für Mock-Objekte für das .NET-Framework.

Erwartungen werden schon beim Ausführen des Quelltextes durchgeführt, anstatt dies später über Asserts oder Ähnliches zu erledigen.

Gemockte Interfaces werden zur Laufzeit erstellt. Fehlermeldungen sollen eindeutig auf den Fehler führen und das Setzen von flexiblen Erwartungen an Methoden kann durch Benutzen einer Constraints-Bibliothek geschehen.

# 2 EasyMock.NET

## 2.1 Homepage
http://sourceforge.net/projects/easymocknet/

## 2.2 Lizenz
BSD-Lizenz

## 2.3 Kurzbeschreibung
Klassenbibliothek, um einfach Mock-Objekte für Interfaces zu erstellen.

Ist Teil des EasyMock Framworks.

# 3 TypeMock Isolator

## 3.1 Homepage
http://www.typemock.com/

## 3.2 Lizenz
Trial-Version kostenfrei, danach je nach Version und Paket: Entwickler-Lizenz kostet 799€ und 150€ jedes weitere Jahr.

## 3.3 Kurzbeschreibung
Einzig gefundenes kommerzielle Mocking-Tool.

Bietet unter anderem die Möglichkeit das Verhalten von statischen Methoden, versiegelten Klassen und privaten Methoden zu verändern, ohne angeblich den Quelltext direkt selbst groß zuverändern.

# 4 Moq

## 4.1 Homepage
http://code.google.com/p/moq/

### 4.2 Lizenz
neue BSD-Lizenz

### 4.3 Kurzbeschreibung
Spezielle für .NET 3.5 entwickelte Klassenbibliothek, um alle Features von .NET 3.5 nutzen zu können.

Die API soll extrem einfach und leicht zu bedienen sein. Vorwissen für Mocking soll nicht notwendig sein um Moq gut benutzen zu können.

## 5 Rhino Mocks

### 5.1 Homepage
http://www.ayende.com/projects/rhino-mocks.aspx

### 5.2 Dokumentation:
http://www.ayende.com/wiki/Rhino+Mocks+Documentation.ashx

### 5.3 Lizenz
Frei unter der BSD-Lizenz

### 5.4 Untersuchte Version
Rhino Mocks 3.5 - For .Net 3.5

04/10/2008 18:26:00 Published by Ayende Rahien

### 5.5 Letzter Untersuchtungszeitpunkt
02.01.2011

### 5.6 Kurzbeschreibung
Rhino Mocks ist eine Zusammenfassung von Klassenbibliotheken zum Mocken von Klassen unter C#. Verschiedene Versionen existieren für .NET 3.5, .NET 2.0 und Silverlight. Zur Testerstellung

wird außerdem NUnit oder MbUnit benötigt. Wahrscheinlich lassen sich auch äquivalente Tools benutzen. Getestet wurden allerdings nur die zwei Genannten.

Funktional hat Rhino Mocks Folgendes zu bieten:

- Verwendung vom expliziten Aufzeichnung-und-Wiederholungs-Modell
- Natural Arrange, Act, Assert-Syntax
- Setzen von Erwartungen von Verhalten von Klassen/Funktionen basierend auf :
    - Vergleich von Argumenten
    - Vergleich von Constraints
    - Callbacks zum Abgleichen von erwarteten Argumenten(später näher beschrieben)
    - Setzen von Aktionen von Methoden, Rückgabewerten, oder werfen von Ausnahmen

## 5.7 Installation

Vorrausgesetzt wird eine Entwicklungsumgebung und Nunit oder MbUnit. Zum Test wurde Mircosoft Visual Studio Ultimate 2010 als Entwicklungsumgebung verwendet.

Benötigt wird von NUnit/MbUnit die entsprechende Dynamic Link Library Datei.

Nachdem Download von RhinoMocks ( http://www.ayende.com/projects/rhino-mocks/downloads.aspx ) kann dies einfach in ein beliebiges, am besten gut wieder zu findendes Verzeichnis entpackt werden. Dort befindet sich die Datei Rhino.Mocks.dll .

Nun wird die Entwicklungsumgebung geöffnet und ein neues oder vorhandenes Projekt geladen. Als nächstes wird eine Klassenbibliothek erstellt, die der Test testen soll.

Um Rhino Mocks zu benutzen, wird die Framework.dll von NUnit/MbUnit und die herruntergeladene RhinoMocks.dll als Verweis/Reference der Klassenbibliothek hinzugefügt.

Zu sehen ist dies in Abbildung 1[1].

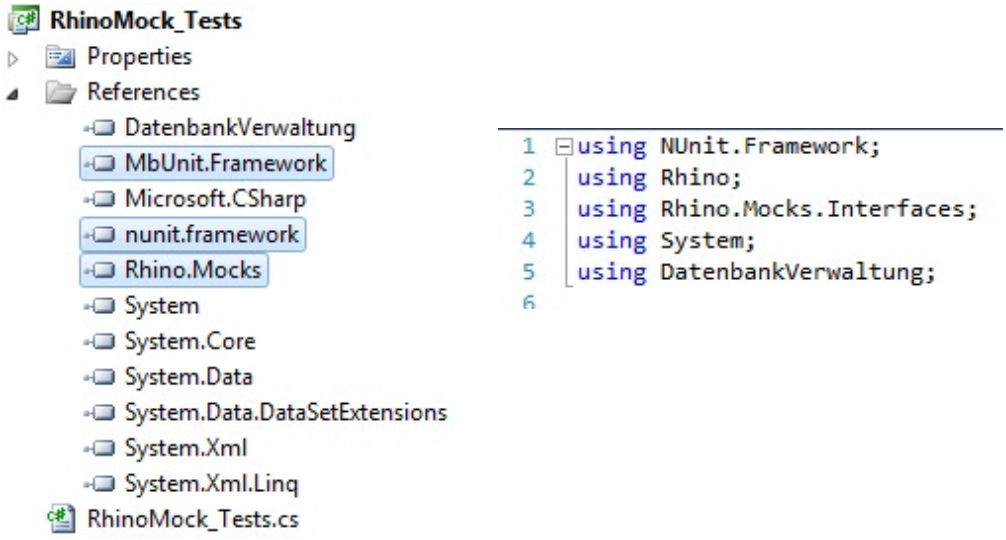


Abbildung 1: Verweise für Rhino-Mocks

## 5.8 Dokumentation

Die Dokumentation ist sehr umfassend und leicht zugänglich, soweit man etwas Erfahrung mit C# oder Java hat. Sie ist komplett auf Englisch gehalten, übersetzte Versionen gibt es scheinbar nicht.

Verschiedene Kapitel gehen schrittweise auf die Möglichkeiten der Nutzung von RhinoMocks ein. Dazu gibt es Quelltext-Beispiele, die aber bei den "komplexeren" Beispielen manchmal nicht voll funktionsfähig sind. Die Quick-Search-Funktion auf der Seite scheint nicht richtig zu funktionieren.

Weiterhin ist am unteren Rand eine Schnell-Referenz der Versionen 3.1(ohne Beispiele) und 3.3 (mit Beispielen) als PDF-Datei vorhanden. Auf der Hauptseite existiert noch eine "API-Documentation"( http://www.ayende.com/projects/rhino-mocks/api/files/MockRepository-cs.html ) , die zu einer Übersicht der verwendbaren Methoden führt. Zu den meisten dieser Methoden gibt ein kurzes Beispiel.

[1] Verwenden von mehreren Frameworks kann zu Konflikten führen, die zum Resultat führen, dass RhinoMocks nicht funktioniert. Das Hinzufügen des Verweises ist allerdings unproblematisch

Weiterhin hat der Autor auch ein Klassendiagramm der Version 2.5.2 (http://www.ayende.com/Images/RhinoDiagram.png) veröffentlicht.

Unter http://www.ayende.com/hibernating-rhinos.aspx ist ein Video zur Einführung von RhinoMocks zu finden. Dort werden die Grundlagen zu RhinoMocks in ca 52 Minuten auf englischer Sprache erklärt.

Abschließend ist zu sagen, dass sich die Autoren viel Mühe mit der Dokumentation gemacht haben und es dadurch gut möglich ist sich zu RhinoMocks Zugang zu verschaffen.

## 5.9 Wartung der Projektseite

Die Seite macht einen guten und übersichtlichen Eindruck.

Die letzte News ist vom 1.9.2009. Es gibt einen Blog, der direkt auf der Hauptseite verlinkt ist. Dieser scheint noch in Benutzung zu sein. Hier können Fragen und Kommentare abgegeben werden. Der letzte Eintrag ist von Anfang Februar 2011.

Erreichen kann man den Autor über Telefon und Email unter dem Punkt "About Me" auf der Hauptseite.

Die Download-Seite ist gut dokumentiert und das jeweilige Erscheinungsdatum wird immer angezeigt. Versionen in neuerer Zeit sind nicht erschienen, was daran liegen kann, das dass Tool ziemlich vollständig wirkt.

## 5.10 Intuitive Benutzbarkeit

Durch die gelungende Dokumentation und das Einführungsvideo lässt sich RhinoMocks recht schnell und gut nutzen. Die häufigen Quelltext-Beispiele sind dabei sehr hilfreich. Sofern man sich etwas mit Java oder C# auskennt, wird man sich schnell einarbeiten können.

## 5.11 Einführendes Beispiel und genereller Aufbau

Für diesen Abschnitt wird angenommen, dass die Installation wie oben dargestellt durchgeführt wurde.

Um eine Testklasse zu kennzeichnen, muss das Attribut '[TestFixture]' direkt vor der Klassendefinition eingetragen werden. Tests werden als Methoden geschrieben (in der Regel mit dem Rückgabewert void und ohne Parameter). Sie müssen mit dem Attribut '[Test]' gekennzeichnet werden. Es besteht die Möglichkeit Methoden zu schreiben, die vor und nach jedem Test ausgeführt werden. Gekennzeichnet werden sie mit '[Setup]' und '[TearDown]'[2].

Die Hauptklasse zur Benutzung von RhinoMocks ist 'MockRepository'. Mit dieser wird ein Objekt erzeugt, welches die Mock-Objekte für die Interfaces erzeugt: 'new MockRepository()'. Durch Aufrufen der Methode 'CreateMock<Interface>()' kann einem Interface der enstprechende Mock zugeordnet werden. Es gibt noch weitere Möglichkeiten ein Mock-Objekt zu erzeugen[3].

Nun müssen die erwarteten Aufrufe festgelegt werden:

```
'Expect.Call(objekt.methode).Return(rückgabewert)'
```

Bei Methoden ohne Parameter und ohne Rückgabewert kann die Return-Anweisung weggelassen werden.

[2] Für genauere Informationen siehe Kapitel NUnit
[3] Siehe dazu Kapitel 5.12 Detaillierte Beschreibung (Seite 9)

Sollte es sich um eine Methode mit Parametern und ohne Rückgabewert handeln, muss man die erwartete Methode als 'delegate' im Argument des Calls definieren (siehe „5.12.3 Erwartete Reihenfolge von Methodenaufrufen“, Listing 5).

Nachdem alle erwarteten Aufrufe beschrieben wurden, muss das erwartete Verhalten gegen den wirklichen Aufruf ausgewertet werden:

```
'mockrepository.ReplayAll()'
```

Danach muss die entsprechende Methode des entsprechenden Objekts aufgerufen werden.

```
'mocksrepository.VerifyAll()'
```

Damit werden die Erwartungen gegen die Aufrufe ausgewertet.

```
[TestFixture]
public class RhinoMock_Tests{
private MockRepository Mocks;

[SetUp]
public void TestSetup(){
     Mocks = new MockRepository();
}

[TearDown]
public void TestTearDown(){
     Mocks.VerifyAll();
}

[Test]
public void GeneralClassMocking(){
     //IDatabase database = mocks.DynamicMock<IDatabase>();
     //IDatabase database = mocks.StrictMock<IDatabase>();
     IDatabase database = Mocks.CreateMock<IDatabase>();
     Expect.Call(database.ConnectionEtasblished()).Return(true);
     Mocks.ReplayAll();
     Assert.True(database.ConnectionEtasblished());
}
```

Listing 1: Einführungsbeispiel von RhinoMocks

Um den Test nun aufzurufen, muss die entsprechende Datei kompiliert und anschließend mit NUnit ausgeführt werden.

```
public interface IDatabase{
     event EventHandler Load;
     void DeleteDatabaseByID(int id);
     bool ConnectionEtasblished();
     bool ConnectionTerminated();
     int MaximumConnections { get; set; }
}
```

Listing 2: IDatabase-Interface

## 5.12 Detaillierte Beschreibung

Nach diesem kleinen Beispiel, werden nun die Einsatzmöglichkeiten von RhinoMocks beschrieben. Die Themen sind angelehnt an die Dokumentation, da diese einen gut strukturierten Aufbau hat. Die Quelltext-Beispiele sind vollständig lauffähig und sollen die Beschreibungen verdeutlichen.

### 5.12.1 Verschiedene Arten zu mocken

Neben der Standardmöglichkeit Mock-Objekte zu erzeugen gibt es noch die Möglichkeit des Strict/Dynamic/Partial-Mocks.

Bei dem Strict-Mock muss jeder erwartete Methodenaufruf auch wirklich stattfinden, ansonsten wird eine Ausnahme geworfen und der Test wird als fehlerhaft abgeschlossen.

Beim Dynamic-Mock werden alle unerwarteten Aufrufe schlicht ignoriert. Die ist sinnvoll wenn egal ist was sonst noch für Aufrufe stattfinden. Generell ist der Strict-Mock öfters einsetzbar, da man sich Gedanken machen muss, wo welche Methode aufgerufen werden soll.

Partial-Mocks werden meist verwendet um nur bestimmte Teile von Klassen, wie zum Beispiel einzelne Methoden einer Klasse, zu mocken.

```
[Test]
public void GeneralClassMocking(){
     //IDatabase database = mocks.DynamicMock<IDatabase>();
     //IDatabase database = mocks.StrictMock<IDatabase>();
     IDatabase database = Mocks.CreateMock<IDatabase>();
     Expect.Call(database.ConnectionEtasblished()).Return(true);
     Mocks.ReplayAll();
     Assert.True(database.ConnectionEtasblished());
}
[Test]
public void PartialMockTest(){
     Server server = Mocks.PartialMock<Server>(); //partielles Mocken
     IDatabase database = Mocks.StrictMock<IDatabase>();
     Expect.Call(server.SelectDatabase(42)).Return(database);
     Mocks.ReplayAll();
     server.SelectDatabase(42);
}
```

Listing 3: Mocking-Typen unter RhinoMocks

### 5.12.2 Stubs

Durch Stubs lassen sich Variablen in Interfaces simulieren. Stubs simulieren getter und setter für Variablen in einem Interface.

```
[Test]
public void GenerateStubTest(){
     IDatabase database;
     // database = mocks.GenerateStub<IDatabase>(); -- GenerateStub Methode
     // database = (IDatabase) mocks.Stub(typeof(IDatabase), null);
     // Unter .Net 1.1
     database = Mocks.Stub<IDatabase>();
     database.MaximumConnections = 4;
     Assert.AreEqual(database.MaximumConnections, 4);
     Mocks.ReplayAll();
}
```

Listing 4: Stubs unter RhinoMocks

### 5.12.3 Erwartete Reihenfolge von Methodenaufrufen

Durch RhinoMocks lässt sich festlegen, ob Methodenaufrufe in bestimmter Reihenfolge erwartet werden. Die Standardeinstellung ist, dass sie in keiner bestimmten Reihenfolge erwartet werden.

Möchte man eine erwartete Reihenfolge festlegen geschieht dies durch:

```
'using (mockrepository.Ordered()){ Quelltext }'
```

Werden danach noch Aufrufe in beliebiger Reihenfolge erwartet kann man dies so beschreiben:

```
'using (mockrepository.Unordered()){ Quelltext }'
```

Werden Aufrufe dann in der falschen Reihenfolge aufgerufen, schlägt der Test fehl.

Es lassen sich 'Unordered' und 'Ordered' beliebig kombinieren.

```
[Test]
public void OrderedAndUnorderedTest(){
     IDatabase database = Mocks.DynamicMock<IDatabase>();
     using (Mocks.Ordered())
          Expect.Call(delegate { database.DeleteDatabaseByID(42); });
          Expect.Call(database.ConnectionEtasblished()).Return(true);
          Expect.Call(database.ConnectionTerminated()).Return(true);
          using (Mocks.Unordered())
               Expect.Call(database.ConnectionTerminated()).Return(false);
               Expect.Call(database.ConnectionEtasblished()).Return(true);
          }
     }
     Mocks.ReplayAll();

     database.DeleteDatabaseByID(42);
     database.ConnectionEtasblished();
     database.ConnectionTerminated();
     database.ConnectionEtasblished();
     database.ConnectionTerminated();
}
```

Listing 5: Erwartete Reihenfolge von Methodenaufrufen unter RhinoMocks

An diesem Beispiel wird gezeigt wie Methoden ohne Rückgabewert und mit Parametern deklariert werden müssen damit sie aktzeptiert werden.

Näheres zu diesem Problem : http://ayende.com/Blog/archive/2007/10/27/Using-Expect.Call-void-method.aspx

### 5.12.4 Delegate-Methoden

RhinoMocks bietet auch das Mocken von Delegate-Methoden an. Das Beispiel spricht größtenteils für sich. Die GenerateStub-Methode erzeugt eine Methode die als Parameter 2 Int-Werte hat. Bei dem ersten Wert handelt sich um den Übergabeparameter der zu erzeugenden Methode und der letzte Wert gibt den Rückgabetyp der Methode an. Der letzte Parameter ist immer der Rückgabetyp, sodass man theoretisch beliebigviele Parameter angeben kann.

Die Stub-Methode sagt, was passieren soll, wenn sie aufgerufen wird. In diesem Fall soll x den Wert 42 erhalten.

```
[Test]
public void DelegateMethods(){
     var stubMapper = MockRepository.GenerateStub<Func<int, int>>();
     int expectedResult = 42;
     stubMapper.Stub(x => x(42)).Return(expectedResult);
     BackUpServer backUpServer = new BackUpServer(stubMapper);
     int result = backUpServer.DoSomething(42);
     Assert.AreEqual(expectedResult, result);
}
```

Listing 6: Delegate-Beispiel unter RhinoMocks

### 5.12.5 Event Handling und IEventRaiser

Weiterhin bietet RhinoMocks das Mocken von Event und bietet dafür diverse Interfaces an.

Das Beispiel steht hier exemplarisch für eine der Möglichkeiten des Event Handlings.

Weitere Information zum Event Handling bietet die ausführliche Dokumentation.

```
[Test]
public void FakingRaisedEvent(){
     object obj = new object();
     EventArgs args = new EventArgs();

     IDatabase database = Mocks.DynamicMock<IDatabase>();
     database.Load += null;
     LastCall.IgnoreArguments();

     IEventRaiser raiseViewEvent = LastCall.GetEventRaiser();
     Mocks.ReplayAll();

     DatabaseCheck dataCheck = new DatabaseCheck(database);
     raiseViewEvent.Raise(obj,args);
     Assert.IsTrue(dataCheck.OnLoadCalled);
}
```

Listing 7: Event-Handling Beispiel unter RhinoMocks

```
public class DatabaseCheck{
     public bool  OnLoadCalled = false;

     public DatabaseCheck(IDatabase database){
          database.Load += ExecuteDatabaseCheck;
     }

     private void ExecuteDatabaseCheck(object sender, EventArgs e){
         OnLoadCalled = true;
     }
}
```

Listing 8: DatabaseCheck-Klasse

### 5.12.6 Rückgabewerte festsetzen

Wenn egal ist, wie oft eine Funktion aufgerufen wird und es gewünscht ist, dass sie immer den Rückgabewert innerhalb eines Tests hat, bietet RhinoMocks die Möglichkeit mit 'SetupResult' den Rückgabewert festzulegen.

```
[Test]
public void SetupResultTest(){
     IUser user = Mocks.StrictMock<IUser>();

     SetupResult.For(user.Anmelden(null)).Return(true);
     SetupResult.For(user.Abmelden()).Return(false);

     Mocks.ReplayAll();
     Assert.True(user.Anmelden(null));
     Assert.False(user.Abmelden());
}
```

Listing 9: Setzen von Rückgabewerten unter RhinoMocks

```
public interface IUser{
     bool Anmelden(IDatabase database);
     bool Abmelden();
    }
```

Listing 10: IUser-Interface

### 5.12.7 Alternativer Weg beim Setzen von Erwartungen mit 'With'

Der Autor bietet mit der With-Klasse eine Möglichkeit an bei einer Kette von Methoden den erwarteten Aufruf zu setzen, den Rückgabewert festzulegen und zu prüfen, ob die erwarteten Aufrufe auch so stattgefunden haben. Diese Schreibweise ist kompakter und setzt den Einsatz von delegate-Methoden vorraus. Als weiterer Ansatz gibt es dadurch die Möglichkeit, Tests zusammenzufassen

```
[Test]
public void WithTest(){
     IDatabase database = Mocks.StrictMock<IDatabase>();
     With.Mocks(Mocks).Expecting(delegate{
          Expect.Call(database.ConnectionTerminated()).Return(true);
     })
     .Verify(delegate{
          database.ConnectionTerminated();
     });
}
```

Listing 11: Test mit With

## 5.13 Weitere Möglichkeiten und Funktionen

Im Weiteren wird noch ein kleiner Überblick gegeben, welche Möglichkeiten RhinoMocks noch bietet.

Diese sind nur kurz aufgeführt, weil deren genaue Erläuterung zu tief gehen würde. Hier wird auf die umfangreiche Dokumentation verwiesen.

### 5.13.1 Generelles

RhinoMocks bietet eine große Anzahl von Constraints und Methodenoptionen wie zum Beispiel 'LastCall' (arbeitet auf dem letzten aufgerufenen Mock-Objekt). Diese sind meist sehr intuitiv, da die Namen für sich selbst sprechen. Details lassen sich in der Dokumentation und der API (http://www.ayende.com/projects/rhino-mocks/api/files/MockRepository-cs.html ) finden.

### 5.13.2 Events

Es wird die Möglichkeit angeboten, dass sich die zu testende Klasse an einem Event eines gemockten Interfaces anmeldet und zu überprüfen, ob dieses auch wirklich geschehen ist.

### 5.13.3 Properties

Das Setzen von gettern/settern bei Mock-Objekten kann recht lang werden.

Dieses lässt sich automatisieren, indem man an den erwarteten Aufruf einfach

```
.PropertyBehavior()
```

hängt. Damit simuliert RhinoMocks einfaches Verhalten von Variablen.

### 5.13.4 Callbacks

Mit Callbacks wird eine Möglichkeit geboten, RhinoMocks den Benutzer zu fragen, ob der Aufruf einer Methode in diesem Kontext wirklich erwartet wird. Der Benutzer muss verifizieren, dass der Aufruf an diesem Punkt beabsichtig wirklich war.

Dieses scheint eine sehr mächtige, aber auch nicht ganz ungefährliche Methode zusein.

Es wird hier empfohlen die Dokumentation zu lesen und/oder den Autor näher zu fragen, wenn man diese Methode sinnvoll einsetzen möchte.

### 5.13.5 Do-Handler

Der Do-Handler ergänzt einen Methodenaufruf. Wird ein erwarteter Aufruf einer Methode durchgeführt, wird auch der Do-Handler ausgeführt.

Syntax : '.Do(new delegateMethode(parameter))'

# 6 Quellen und hilfreiche Links

Hauptseite: http://www.ayende.com/projects/rhino-mocks.aspx

Dokumentation : http://www.ayende.com/wiki/Rhino+Mocks.ashx

Download: http://www.ayende.com/projects/rhino-mocks/downloads.aspx

Video-Tutorial: http://www.ayende.com/hibernating-rhinos.aspx

Void-Methode-mit-Parameter-Problem: http://ayende.com/Blog/archive/2007/10/27/Using-Expect.Call-void-method.aspx