

Unit-Tests bei SharePoint-2010-Projekten

Und es geht doch!

SharePoint ist zu einer Plattform herangewachsen, auf der inzwischen auch Geschäftsanwendungen umgesetzt werden. Defizite gibt es in diesem Umfeld auf Seiten der Entwickler allerdings noch beim Qualitätsmanagement – Stichwort Unit-Tests. Solche sind aber auch in SharePoint-Projekten möglich.

Auf einen Blick



Marcus Peters kennt .NET seit der ersten Stunde und ist bekennender SharePoint-Fan. Er verantwortet im Competence Center Portal Applications bei adesso aus Dortmund die Umsetzung von Business-Applikationen auf der SharePoint-Plattform. Sie erreichen ihn unter marcus.peters@adesso.de.

Inhalt

- › Testorientiertes Entwickeln auf der SharePoint-Plattform durch Isolieren von Code.
- › Mocking-Frameworks für SharePoint nutzen.
- › Unit-Tests auf SharePoint mit Pex parametrisieren.

dnpCode

A1012TestSharePoint

Für Entwickler gilt der Umgang mit SharePoint gemeinhin als nicht so einfach. Das Credo lautet häufig: „Zück nicht gleich Visual Studio, schau lieber erst mal, ob die Lösung durch Konfiguration oder Bordmittel zu erreichen ist.“ Wenn dann allerdings doch „echte Entwicklungsarbeit“ gefragt ist, stellt sich unter den hartgesottenen Kollegen schnell die Frage, ob das denn überhaupt geht.

Die Scheu vor dem Entwickeln für SharePoint ist in der Komplexität des Dokumentenservers und in der mangelnden Unterstützung durch geeignete Werkzeuge in der Vergangenheit zu suchen. Insbesondere im Zusammenhang mit Unit-Tests macht SharePoint es den Entwicklern nicht leicht.

In Unit-Tests werden die Abhängigkeiten des Produktivcodes von weiteren Systemen durch Code-Dummies isoliert. Das Ersetzen von Produktivcode wird gern als „mocking“ bezeichnet (Abbildung 1 und 2). Damit sich dies einfach bewältigen lässt, unterstützen entsprechende Frameworks ein „Simulieren“ von Code. Allerdings setzt dies eine schnittstellenorientierte Architektur des zu testenden Systems voraus [1].

SharePoint und Test-Driven Development

Zunächst ist zu klären, ob SharePoint die architektonischen Voraussetzungen für das Isolieren von Code erfüllt. Die Antwort ist ernüchternd und lautet „Nein“. Viele Klassen sind versiegelt, haben keinen öffentlichen Konstruktor oder erfordern ein echtes SharePoint-System. Allein der Umgang mit den Klassen *SPWeb* und *SPSite* er-

fordert einen Echtzeitzugriff auf SharePoint mit den entsprechenden tatsächlich vorhandenen Objekten.

```
SPSite site = new SPSite("http://abc");
```

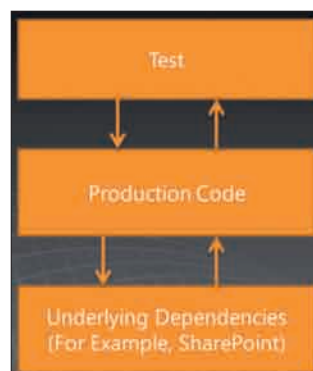
Diese Zeile erzeugt einen Fehler, wenn der übergebene URL auf eine Seite verweist, die auf SharePoint nicht vorhanden ist. Ähnliche Effekte treten auch an anderen Stellen im Umgang mit dem SharePoint-API auf.

Ein häufiger Ansatz für den Umgang mit Unit-Tests mit SharePoint ist das Separieren des eigenen Codes von SharePoint-Code. Andrew Woodward, MVP für MOSS 2007, beschreibt dies in [2]. Dieser Ansatz ist jedoch recht eingeschränkt. Er funktioniert nur, wenn der Produktionscode ohne SharePoint-API auskommt und wenn dieser nach erfolgreichen Unit-Tests zur Integration über das SharePoint-API aufgerufen wird. Ein triviales Beispiel für ein derartiges Szenario ist die Ausgabe einer Zeichenkette in einem Web Part (vergleiche [2], Seite 24), das sehr deutlich die eingeschränkten Möglichkeiten dieses Ansatzes zeigt.

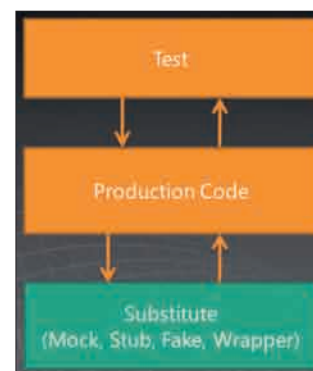
Der Weg zu leistungsfähigen Unit-Tests ist also nur über eine echte Isolation durch Substitution des SharePoint-APIs zu erreichen.

Typische Werkzeuge im Einsatz haben Tücken

Zur Strategie müssen auch Werkzeuge passen. Jeremy Thake beschreibt unter anderem den Einsatz der „üblichen Verdächtigen“ wie xUnit, MSTest und NUnit [3]. Interessant ist vor allem, dass



[Abb. 1] Das Zusammenspiel Produktionscode mit Abhängigkeiten.



[Abb. 2] Die Isolation des Codes durch Substitution.

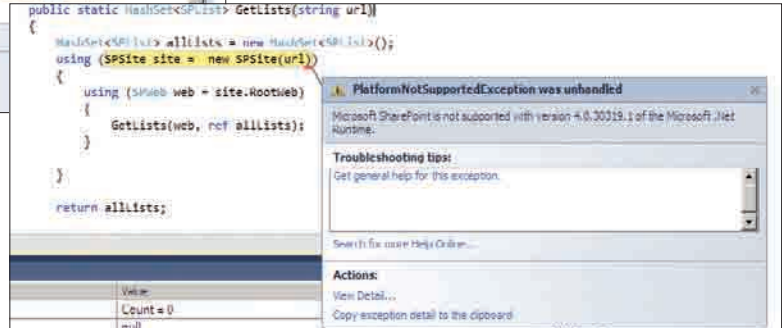


[Abb. 3] Der Aufruf des SharePoint-APIs aus einem 32-Bit-Prozess führt zu einem Fehler.

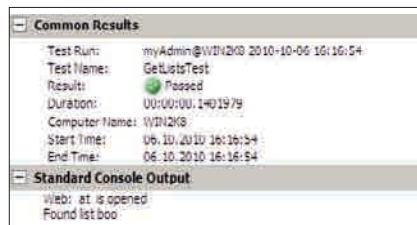
MSTest als sofort einsetzbares Framework nicht richtig mitspielen will:

- MSTest läuft nicht als 32-Bit-Prozess. Das führt zur Fehlermeldung in Abbildung 3.
- Das Tool läuft in Visual Studio 2010 nur mit dem Ziel-Framework .NET 4. In Verbindung mit SharePoint 2010 führt dies zu der Fehlermeldung, die in Abbildung 4 zu sehen ist.

Durch den geschickten Einsatz von Test-Driven.NET, NUnit und ReSharper lassen sich zwar Testfälle durchlaufen, aber es handelt sich dabei streng genommen ausschließlich um Integrationstests.



[Abb. 4] Der Aufruf des SharePoint-APIs mit .NET 4 klappt ebenfalls nicht.



[Abb. 5] Ein erfolgreicher Unit-Test mit Typemock.

Typemock isoliert Produktionscode

Nach all der Ernüchterung ist es nun an der Zeit für Erfolgsmeldungen. Für eine echte Substitution des SharePoint-APIs war bislang nur Typemock [4] bekannt; Chris Keyser von Microsofts Patterns and Practices Group hat das Tool auf der SharePoint Conference 2009 in Las Vegas als geeignet bezeichnet. Typemock isoliert den Produktionscode über Techniken der aspektorientierten Programmierung, wobei Typemock Aufrufe auf die Bibliotheken des Ziel-APIs von Typemock umleitet [5].

Damit ist im Prinzip jedes auf diesem Konzept beruhende Mocking-Framework für Unit-Tests unter SharePoint geeignet. Ein Beispiel soll den Umgang mit Typemock zeigen. Listing 1 enthält eine Funktion, die rekursiv alle zur Verfügung stehenden SharePoint-Listen eines gegebenen *SPWeb* ermittelt.

Durch den Einsatz von Typemock lässt sich dieser Code wie in Listing 2 isolieren und testen.

Interessant sind die Zeilen im *//Arrange*-Abschnitt des Listings. Hier wird zunächst eine isolierte Instanz von *SPWeb* erzeugt,

deren Objektgraph mit Dummy-Objekten gefüllt ist:

```
SPWeb web = Isolate.Fake.Instance<SPWeb>(
    Members.ReturnRecursiveFakes());
```

Dann wird eine Umleitung für die Eigenschaft *Title* des ersten Listen-Items von *web* eingerichtet:

```
Isolate.WhenCalled(() =>
    web.Lists[0].Title).WillReturn(title);
```

Im *//Act*-Block wird das Dummy-*web* der zu testenden Methode übergeben; der *//Assert*-Block prüft, ob die SharePoint-Liste mit dem Titel *boo* in dem Container gelandet ist:

```
Assert.AreEqual(
    allListsExpected[0].Title, title);
```

Das Ergebnis des Tests ist in Abbildung 5 zu sehen und zeigt seinen erfolgreichen Ausgang.

Listing 1

Rekursives Ermitteln aller SharePoint-Listen.

```
public static void GetLists(SPWeb web, ref List<SPList> allLists)
{
    //Dispose the given web at the end of this scope
    using (web)
    {
        SPListCollection webLists = web.Lists;
        foreach (SPList list in webLists)
            allLists.Add(list);

        SPWebCollection subWebs = web.Webs;
        foreach (SPWeb w in subWebs)
            GetLists(w, ref allLists);
    }
}
```

Mocking à la Microsoft mit Moles

Ein weiterer Ansatz für die Isolation von Produktionscode kommt aus dem Hause Microsoft, heißt Moles und hat sehr interessante Aspekte [6]. Der Ansatz dieses Frameworks leitet mithilfe von .NET-Delegaten auf das Ziel-API um. Somit ist das Moles-Framework in der Lage, Aufrufe auf jede .NET-Methode inklusive nicht virtueller und statischer Methoden in versiegelten Klassen zu lenken.

Das Beispiel in Listing 3 wird in vielen Videos und Demos von Moles verwendet und veranschaulicht die Leistungsfähigkeit von Moles. Es zeigt das Einrichten einer Umleitung via Delegat für die statische Eigenschaft *Now* des Typs *DateTime*.

Listing 2

Ein Unit-Test mit Typemock.

```
[TestMethod(), Isolated]
public void GetListsTest()
{
    //Arrange
    SPWeb web = Isolate.Fake.Instance<SPWeb>(Members.ReturnRecursiveFakes);

    string title = "boo";
    Isolate.WhenCalled(() => web.Lists[0].Title).WillReturn(title);
    List<SPList> allListsExpected = new List<SPList>();

    //Act
    Program.GetLists(web, ref allListsExpected);

    //Assert
    Assert.AreEqual(allListsExpected[0].Title, title);
}
```

Listing 3

Umleiten eines Aufrufs mit Moles auf DateTime.Now.

```
...
using Microsoft.Moles.Framework;
[assembly: MoledType(typeof(System.DateTime))]
...

[TestMethod]
[HostType("Moles")]
public void Y2KBugTest()
{
    System.Moles.MDateTime.NowGet = () => new DateTime(2000, 1, 1);

    if (DateTime.Now == new DateTime(2000, 1, 1))
        Assert.Fail("y2k bug");
}
```

Listing 4

Ein von Pex generierter Test.

```
...
[TestMethod]
[PexGeneratedBy(typeof(ProgramTest))]
[ExpectedException(typeof(InvalidOperationException))]
public void ReverseStringThrowsInvalidOperationException766()
{
    string s;
    s = this.ReverseString("Hello World");
}
...
```

Interessant ist vor allem die Zeile, welche die Umleitung einrichtet:

```
System.Moles.MDateTime.NowGet = () =>
    new DateTime(2000, 1, 1);
```

Moles erzeugt für jeden Typ einen eigenen Mole-Typ, der dem Originaltyp entspricht und durch das Präfix „M“ gekennzeichnet ist. So wird aus dem CLR-Typ *System.DateTime* der Mole-Typ *System.Moles.MDate-*

Time. Weitere Details sind in [7] und [8] zu finden. Bei der Installation von Moles können entsprechende Assemblies für die CLR direkt generiert werden. Für benutzerdefinierte Typen genügt ein einfacher Rechtsklick auf die Assembly und die Auswahl des Menüpunkts *Add Moles Assembly*.

Um ähnlich gefüllte Objektgraphen wie im Typemock-Beispiel zu erhalten, stellt Moles sogenannte „Behaviors“ bereit:

```
using Microsoft.SharePoint.Behavior
...

//Arrange
string title = "boo";
BSPWeb web = new BSPWeb();
web.Webs.SetEmpty();
BSPList list = web.Lists.SetOne(title);

//Act
Program.GetLists(web, ref allListsExpected);

//Assert
Assert.AreEqual(allListsExpected[0].Title,
title);
...
```

Parametrisierte Unit-Tests mit Pex und Moles

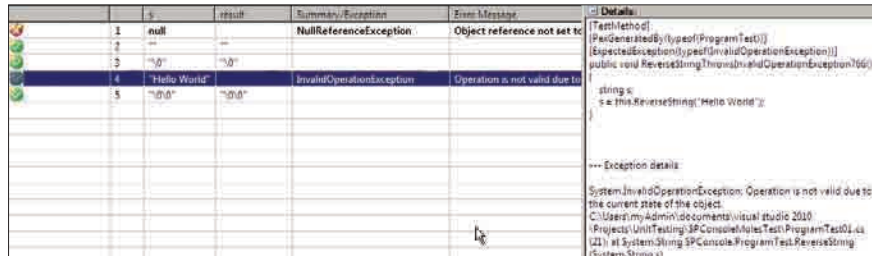
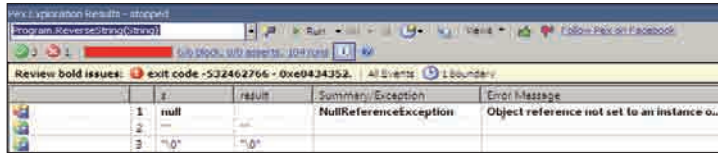
Pex ist ein Add-in für Visual Studio, das parametrisierte Unit-Tests ermöglicht und gemeinsam mit Moles in die Visual Studio Power Tools eingeflossen ist [9]. Genau genommen führt Pex eine Whitebox-Analyse der Befehle auf MSIL-Ebene durch. Die Konzepte dieses Tools und deren Hintergründe wie die symbolische Ausführung sind in [10] beschrieben. Vereinfacht ausgedrückt erzeugt Pex aus dem Ausführungsbaum einer Methode mögliche Parameter und testet die Konsequenzen einer Übergabe dieser Parameter.

Als Beispiel dient die im folgenden Code dargestellte Methode für das Umdrehen einer Zeichenkette. Beim Aufruf mit Pex wird sie auf mögliche Eingabeparameter analysiert und das Ergebnis in einer Tabelle wie in Abbildung 6 dargestellt.

```
public string ReverseString(string s)
{
    StringBuilder sb = new StringBuilder();
    Char[] chars = s.ToCharArray();
    for (int i = chars.Length - 1; i >= 0; i--)
        sb.Append(chars[i]);
    return sb.ToString();
}
```

Die Input-Parameter der Ergebnistabelle erscheinen zwar nachvollziehbar, aber irgendwie doch zufällig. Also wird Pex mit der folgenden Zeile herausgefordert, den Ausführungsbaum neu zu analysieren und

[Abb. 6] Die Analyse der Methode aus ReverseString() mit Pex.



[Abb. 7] Pex erzeugt einen zusätzlichen Test für den erweiterten Ausführungsbaum.

die Zeichenkette „Hello World“ als Parameter auszuprobieren:

```
if(s == "Hello World")
    throw new InvalidOperationException();
```

Und genau das tut Pex, indem es einen entsprechenden Unit-Test erzeugt, dessen Ergebnis Abbildung 7 illustriert.

Bei der Erstellung des zusätzlichen Tests geht Pex davon aus, dass die Ausnahme auf die Eingabe von „Hello World“ als Erwartung definiert ist, und fügt ein entsprechendes Attribut zum Test hinzu (Listing 4).

Durch Einsatz von Pex und Moles lässt sich die Qualität des Codes aus Listing 1 sukzessive mit parametrisierten Unit-Test sichern, da Moles für SharePoint auf Pex abgestimmt sind. So erzeugt die Zeile *MSPSite.BehaveAsNotImplemented()*; durch das Setzen einer allgemeinen Verhaltens-

konfiguration für alle Methoden von *SPSite* eine *MoleNotImplementedException*. Hierdurch wird der Entwickler benachrichtigt, wenn der zu testende Code via *SPSite* auf SharePoint zugreifen will und einen Bereich markiert, der isoliert werden sollte.

Fazit

Unit-Tests sind auch im SharePoint-Umfeld möglich. Allerdings ist die Isolation des Produktionscodes vom SharePoint-API eine Voraussetzung dafür. Wenn durch Refaktorisierung keine weitere Isolation zu erreichen ist, helfen Mocking-Frameworks wie Typemock oder Moles. Der Einsatz von Moles lässt sich durch Pex weiter „veredeln“, da in dieser Kombination auch parametrisierte Unit-Tests möglich sind. Außerdem ist Pex durch Analyse des Ausführungsbaums in der Lage, sinnvolle Testparameter zu generieren und damit eine hohe

Codeabdeckung zu ermöglichen. Als gute Einführung in das Arbeiten mit Pex und Moles im SharePoint-Umfeld sind unbedingt [11] und [12] zu empfehlen. [jp]

- [1] Wikipedia, Test-driven development, Fakes, mocks and integration tests, www.dotnetpro.de/SL1012TestSharePoint1
- [2] Andrew Woodward, Unit Test SharePoint Solutions – The Basics, Beginners Guide to Test Driven Web Part Development, www.dotnetpro.de/SL1012TestSharePoint2
- [3] Jeremy Thake, Unit Testing with SharePoint 2010 Development, www.dotnetpro.de/SL1012TestSharePoint3
- [4] Typemock, www.typemock.com
- [5] How Typemock Isolator Simplifies Unit Testing, www.dotnetpro.de/SL1012TestSharePoint4
- [6] Moles – Isolation framework for .NET, www.dotnetpro.de/SL1012TestSharePoint5
- [7] Unit Testing with Microsoft Moles, www.dotnetpro.de/SL1012TestSharePoint6
- [8] Patterns & Practices SharePoint Guidance 2010, Testing SharePoint Solutions, www.dotnetpro.de/SL1012TestSharePoint7
- [9] Pex and Moles – Isolation and White Box Unit Testing for .NET, www.dotnetpro.de/SL1012TestSharePoint8
- [10] Advanced Concepts, Parameterized Unit Testing with Microsoft Pex, www.dotnetpro.de/SL1012TestSharePoint9
- [11] Unit Testing SharePoint Foundation with Microsoft Pex and Moles, www.dotnetpro.de/SL1012TestSharePoint10
- [12] Pex – Unit Testing of SharePoint Services that Rocks!, www.dotnetpro.de/SL1012TestSharePoint11

CodeMeter: Sicherer & flexibler Softwareschutz!



- Softwareschutz hardwarebasiert mit CodeMeter oder softwarebasiert mit CodeMeterAct
- Softwareschutz für viele Betriebssysteme: Windows, Windows Embedded, Windows CE, Mac OS X, Linux, Sun Solaris und Programmiersprachen: .NET, Java
- Flexibles und umfangreiches Lizenzmanagement: Netzwerk, Einzelplatz, Pay-Per-Use, Overflow, u.v.m.
- Verfügbar in vielen Bauformen: USB, PC-, Express-, CF-, SD- und µSD-Card

Bestellen Sie noch heute Ihr kostenloses CodeMeter Software Development Kit unter: www.wibu.de/sdk



Besuchen Sie uns auf der SPS/IPC/Drives 2010 im Messezentrum Nürnberg vom 23. - 25.11.2010 Halle 7, Stand 530

PERFECTION IN SOFTWARE PROTECTION DOCUMENT

WIBU-SYSTEMS AG
Rüppurrer Straße 52-54
D-76137 Karlsruhe
Tel.: 0721-93172-0
www.wibu.de

