Unit testen binnen Silverlight-projecten

Silverlight Unit Test framework maakt het makkelijk

Maurice de Beijer

Er zullen maar weinig mensen zijn die betwisten dat unit tests een belangrijk onderdeel zijn van een modern software project. Ook voor Silverlight-projecten zijn unit testen een belangrijk onderdeel. Maar daar waar Visual Studio 2008 professional en hoger de nodige hulpmiddelen aan boord heeft om de gewone .NET ontwikkelaar te helpen, is het voor een Silverlight ontwikkelaar een ander verhaal.

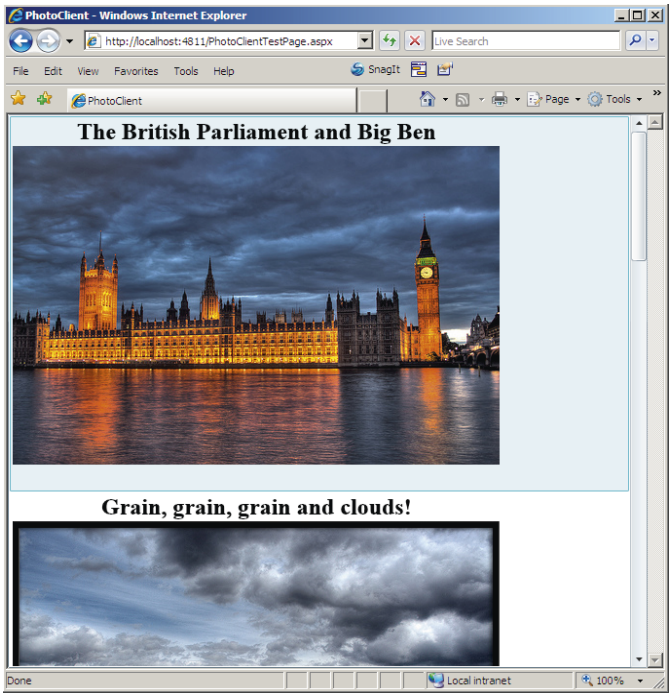
De standaard Visual Studio hulpmiddelen werken namelijk alleen met code die voor het standaard .NET geschreven is, terwijl Silverlight met een veel beperkter versie van het .NET framework werkt. Om toch unit tests op Silverlight code te kunnen uitvoeren, hebben we iets anders nodig. Het Microsoftteam dat de Silverlight Toolkit ontwikkelt had ook de behoefte om unit tests op hun code uit te voeren en heeft hier het Silverlight Unit Test Framework voor gemaakt dat als een onderdeel van de gratis Silverlight Toolkit verspreid wordt.

Silverlight Unit Test Framework downloaden

Het Silverlight Unit Test framework is geen onderdeel van de standaard Silverlight installatie, dus voordat we kunnen beginnen moeten we het eerst downloaden. Er is geen aparte download, maar het is een onderdeel van de Silverlight Toolkit die op CodePlex te vinden is . Overigens bevat de Silverlight Toolkit een hoop waardevolle controls. Op CodePlex vinden we twee versies van de Silverlight Toolkit, één met en de andere zonder de broncode. Het Silverlight Unit Test Framework wordt alleen bij de broncode meegeleverd. Dit is de versie om te downloaden. Als we deze zip file openen vinden we in de Source\Binary folder twee dll bestanden, Microsoft.Silverlight.Testing.dll en Microsoft.VisualStudio.QualityTools.UnitTesting.Silverlight.dll. Dit zijn de twee dll's die we nodig hebben om de unit tests te maken en uit te voeren. In de download van CodePlex zijn alleen de twee dll's te vinden maar geen documentatie of projecttemplates. Die documentatie en project templates zijn er wel, maar moeten van een andere plaats namelijk de Microsoft Silverlight Unit Test Framework home pagina op de MSDN Code Gallery gedownload worden.

Een eerste Silverlight unit test maken

Nu we het Microsoft Silverlight Unit Test Framework hebben, kunnen we het gebruiken om een eerste unit test te maken. Als eerste stap moeten we een Silverlight-applicatie maken om te testen. In dit geval heb ik een kleine fotobrowser gemaakt. De klasse hierbinnen die we gaan testen is de PhotoCollection. Deze PhotoCollection bevat in de Add functie een test of de toe te voegen foto wel van een URL voorzien is.



AFBEELDING 1: DE FOTO BROWSER IN ACTIE.

```
using System;
using System.Collections.ObjectModel;
using System.Windows;

namespace PhotoClient
{
    public class Photo
    {
        public string Title { get; set; }
        public string Url { get; set; }
        public double Height { get; set; }
```

```
        public double Width { get; set; }
        public Size Size
        {
            get
            {
                return new Size(Width, Height);
            }
        }
    }

    public class PhotoCollection : ObservableCollection<Photo>
    {
        public PhotoCollection()
        {
        }

        public new void Add(Photo photo)
        {
            if (string.IsNullOrEmpty(photo.Url))
                throw new ArgumentException("Photo Url cannot be
empty.");

            base.Add(photo);
        }

        public void LoadPhotosFromFlickr()
        {
            ThreadPool.QueueUserWorkItem(state =>
                {
                    // Similate loading the photos from:
                    // http://www.flickr.com/photos/mauricedb/
                    Thread.Sleep(TimeSpan.FromMilliseconds(500));
                    Add(new Photo()
                    {
                        Title = "The British Parliament and Big
Ben",
                        Url = "Images/BigBen.jpg",
                        Width = 500,
                        Height = 332
                    });
                    Thread.Sleep(TimeSpan.FromMilliseconds(500));
                    Add(new Photo()
                    {
                        Title = "Grain, grain, grain and clouds!",
                        Url = "Images/Grain.jpg",
                        Width = 500,
                        Height = 338
                    });
                });
        }
    }
}
```

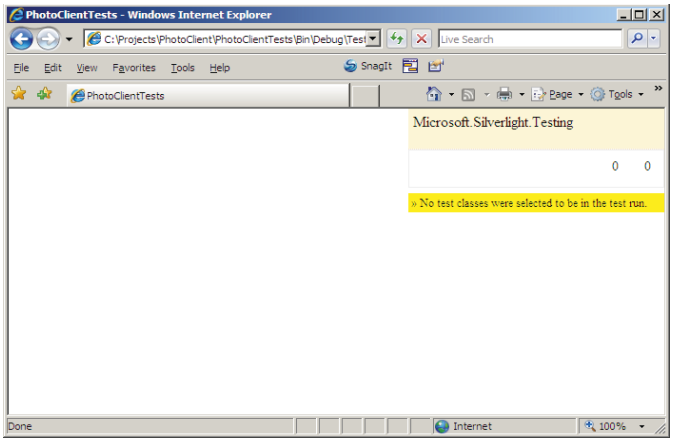
CODELISTING 1: DE PHOTO EN DE PHOTOCOLLECTION KLASSES.

Als eerste moeten we een tweede Silverlight project toevoegen om de tests in uit te voeren. Noem dit project PhotoClient-Tests. Op de vraag of we een ASP.NET Web project toe willen voegen, automatisch een test pagina willen genereren of dat deze aan de bestaande website toegevoegd moet worden, moeten we de tweede optie - automatische test pagina - kiezen. Als eerste moeten we nu een referentie toevoegen naar de twee eerder genoemde DLL's van het Silverlight Unit Test Framework.

Nadat deze referenties toegevoegd zijn, moeten we nog één ding doen voor we onze eerste tests kunnen maken en dat is de RootVisual van de applicatie naar de standaard testpagina te zetten. Deze hoofdpagina wordt aangemaakt door de statische CreateTestPage() functie van de Microsoft.Silverlight.Testing.UnitTestSystem klasse. Zie codelisting 2 voor deze code. Op dit moment kunnen we het testproject starten wat het scherm in afbeelding 2 zal tonen.

```
using System;
using System.Windows;
using Microsoft.Silverlight.Testing;

namespace PhotoClientTests
{
    public partial class App : Application
    {

        public App()
        {
            this.Startup += this.Application_Startup;
            this.Exit += this.Application_Exit;
            this.UnhandledException += this.Application_Unhandle-
dException;

            InitializeComponent();
        }

        private void Application_Startup(object sender, StartupE-
ventArgs e)
        {
            this.RootVisual = UnitTestSystem.CreateTestPage();
        }

        // Rest van de code verwijderd voor de leesbaarheid
    }
}
```

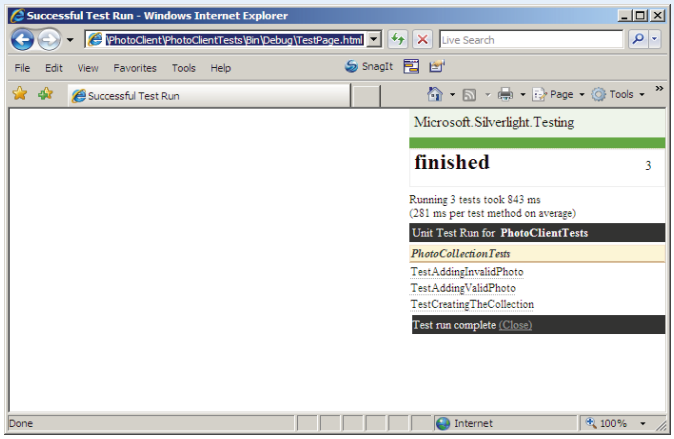
CODELISTING 2: DE APPLICATIE KLASSE IN HET TEST PROJECT.



AFBEELDING 2: HET LEGE MAAR WEL GECONFIGUREERDE TEST PROJECT.

Het maken van de eerste echte unit test binnen dit nieuwe project zal de meeste .NET ontwikkelaars heel bekend voorkomen. De reden is dat dezelfde attributen en klassen gebruikt worden om een test te maken. Als eerste moeten we een referentie toevoegen naar het te testen Silverlight project, de PhotoClient. De test zelf komt in een klasse met het attribuut TestClass. Bij het toevoegen van dit attribuut moeten we even oppassen als we de juiste namespace 'Microsoft.VisualStudio.TestTools.UnitTesting' nog niet met een using statement toegevoegd hebben. Visual Studio probeert namelijk de naam te resolven naar zowel TestClass als TestClassAttribute en beide bestaan, zij het in verschillende name spaces. Helaas staat de verkeerde namespace bovenaan en is het makkelijk om die per ongeluk toe te voegen.

De test functie, TestCreatingTheCollection in dit geval, is net als anders een public void functie die voorzien is van het TestMethod attribuut. Deze eerste unit test controleert twee verschillende dingen. De eerste controle is of de collectie goed aangemaakt kan worden en dit gebeurt automatisch als er geen exceptie optreed. De tweede controle is of er standaard vijf foto's aanwezig zijn. Dit wordt gedaan door de Assert klasse te gebruiken.



AFBEELDING 3: DE BROWSER NA EEN GOED VERLOPEN TEST.

Uiteraard kunnen we ook testen of verwachte foutsituaties optreden. De unit test TestAddingInvalidPhoto is een voorbeeld van een test waar we iets ongeldigs doen, in dit geval een foto zonder URL toevoegen. Aangezien we verwachten dat er een exceptie op gaat treden, kunnen we de test voorzien van het ExpectedException attribuut met de te verwachten exceptieklasse. Deze exceptie moet optreden, voordat de test als goed gemarkeerd wordt. Ook hier worden weer standaard .NET attributen en klassen gebruikt, zodat bestaande unit test kennis gebruikt kan worden.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using PhotoClient;
using System;

namespace PhotoClientTests
{
    [TestClass]
    public class PhotoCollectionTests
    {
        [TestMethod]
        public void TestCreatingTheCollection()
        {
            PhotoCollection photos = new PhotoCollection();
            Assert.AreEqual(5, photos.Count);
        }

        [TestMethod]
        public void TestAddingValidPhoto()
        {
            PhotoCollection photos = new PhotoCollection();
            photos.Add(new Photo()
            { Url = "http://somewhere.com/image.jpg" });
            Assert.AreEqual(6, photos.Count);
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentException))]
        public void TestAddingInvalidPhoto()
        {
            PhotoCollection photos = new PhotoCollection();
            photos.Add(new Photo());
        }
    }
}
```

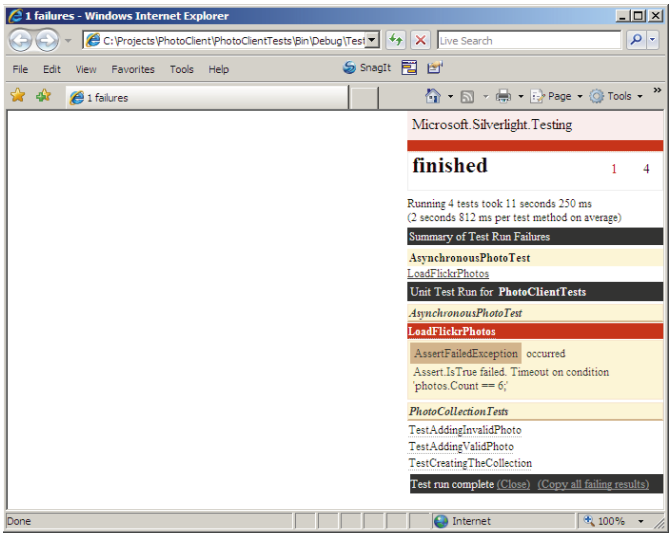
CODELISTING 3: DE KLASSE OM DE PHOTOCOLLECTION TE TESTEN.

Het testen van asynchrone code

Met de bovenstaande tests is het gemakkelijk om synchroon lopende code te testen. Door de manier waarop Silverlight communiceert met externe programmaonderdelen, namelijk asynchroon, zal het hiermee niet mogelijk zijn om alle code te testen. Binnen Visual Studio is het testen van asynchrone code wel mogelijk, ondermeer door gebruik te maken van de AutoResetEvent klasse, maar niet altijd even gemakkelijk. Om het leven binnen Silverlight makkelijker te maken, is er binnen het Silverlight Unit Test Framework een speciale klasse voor dit soort asynchrone code gekomen: de SilverlightTest uit de Microsoft.Silverlight.Testing namespace. Een asynchrone test functie wordt vervolgens ook met het AsynchronousAttribute gemarkeerd. De testomgeving zal nu anders met de test omgaan en als de test functie klaar is niet doorgaan met de volgende test maar wachten tot de test klaar is. Binnen een unit test kan nu een aantal functies, die allemaal met Enqueue beginnen, gebruikt worden om het verdere verloop van de test te beïnvloeden.
De functie LoadPhotosFromFlickr() in de PhotoCollection simuleert het asynchroon laden van foto's van Flickr. Omdat ik dit artikel wil beperken tot unit tests en niet op de complexiteiten van netwerk IO in wil gaan, gebruik ik hier de ThreadPool met een vertraging om de nodige foto's toe te voegen. Zie codelisting 1 voor de LoadPhotosFromFlickr() functie. De code om dit asynchrone gedrag te testen is te vinden in codelisting 4.

```
using Microsoft.Silverlight.Testing;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using PhotoClient;

namespace PhotoClientTests
{
    [TestClass]
    public class AsynchronousPhotoTest : SilverlightTest
    {
        [Asynchronous]
        [TestMethod]
        public void LoadFlickrPhotos()
        {
            PhotoCollection photos = new PhotoCollection();
            photos.Clear();
            Assert.AreEqual(0, photos.Count);

            photos.LoadPhotosFromFlickr();

            EnqueueConditional(() => photos.Count == 3);
            EnqueueTestComplete();
        }

    }
}
```

CODELISTING 4: HET TESTEN VAN ASYNCHRONE CODE.



AFBEELDING 4: DE BROWSER MET EEN FOUT NA EEN ASYNCHRONE UNIT TEST MET TIMEOUT.

De asynchrone test code gebruikt de EnqueueConditional() functie om te wachten tot een bepaalde conditie waar is. In dit geval of er drie foto's geladen zijn. Daarna wordt de EnqueueTestComplete() functie gebruikt om aan te geven dat de unit test klaar is.

De unit test in codelisting 4 werkt prima zolang er maar minimaal drie foto's geladen worden. Maar stel dat er iets verandert in de code en er worden maar twee foto's geladen. In dat geval zal het Silverlight Unit Test Framework blijven wachten tot er een derde foto is en worden de overige tests nooit uitgevoerd. Binnen Visual Studio is er een TimeoutAttribute om te voorkomen dat tests te lang duren. Als de opgegeven timeout verlopen is, zal Visual Studio de test afbreken. Helaas gebruikt het Silverlight Unit Test Framework dit attribuut niet en moeten we zelf zorgen dat een test niet te lang kan duren. De makkelijkste manier om dit te doen is een extra test toe te voegen aan de EnqueueConditional() lambda expressie zoals in codelisting 5 is gebeurt.

```
[Asynchronous]
[TestMethod]
public void LoadFlickrPhotos()
{
    PhotoCollection photos = new PhotoCollection();
    photos.Clear();
    Assert.AreEqual(0, photos.Count);

    photos.LoadPhotosFromFlickr();

    DateTime startTime = DateTime.Now;
    EnqueueConditional(() =>
    {
        Assert.IsTrue((DateTime.Now - startTime) < TimeSpan.From-
Seconds(10),
            "Timeout on condition 'photos.Count == 3;'");

        return photos.Count == 3;
    });
    EnqueueTestComplete();
}
```

CODELISTING 5: EEN EXTRA CONTROLE OM DEADLOCKS TE VOORKOMEN.

Het testen van user interface controls

Met het Silverlight Unit Test Framework is het niet alleen mogelijk om gewone code te testen, het is ook mogelijk om zelfgemaakte controls te testen. Speciaal om dit te doen heeft de SilverlightTest klasse een property TestPanel die toegang geeft tot een panel waar de controls toegevoegd kunnen worden. Om dit te demonstreren heb ik een eenvoudige control gemaakt, te zien in codelistings 6 en 7, waar de gebruiker een foto URL aan kan toevoegen. De controle in de code is dat alleen een URL die over http gaat geldig is, in de andere gevallen moet de toevoeg knop uitgeschakeld blijven.

```
<UserControl x:Class="PhotoClient.MyControl"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentati-
on"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="400" Height="300">
    <StackPanel>
        <TextBox x:Name="txtPhotoUrl"
                 TextChanged="txtPhotoUrl _ TextChanged" />
        <Button x:Name="cmdAdd"
                Click="cmdAdd _ Click"
                Content="Add" IsEnabled="False" />
    </StackPanel>
</UserControl>
```

CODELISTING 6: THE XAML VOOR DE USERCONTROL.

```
public partial class MyControl : UserControl
{
    public MyControl()
    {
        InitializeComponent();
        Photos = new List<string>();
    }

    public IList<string> Photos { get; set; }

    private void cmdAdd _ Click(
        object sender, RoutedEventArgs e)
    {
        string url = txtPhotoUrl.Text;
        Photos.Add(url);
    }

    private void txtPhotoUrl _ TextChanged(
        object sender, TextChangedEventArgs e)
    {
        string url = txtPhotoUrl.Text ?? "";
        Uri uri = new Uri(url);

        if (string.IsNullOrEmpty(url))
            cmdAdd.IsEnabled = false;
        else if (string.IsNullOrEmpty(uri.Scheme))
            cmdAdd.IsEnabled = false;
        else if (uri.Scheme != "http")
            cmdAdd.IsEnabled = false;
        else
            cmdAdd.IsEnabled = true;
    }
}
```

CODELISTING 7: DE CODE BEHIND VOOR DE USERCONTROL.

Om de TestPanel te kunnen gebruiken, moeten we weer afleiden van de SilverlightTest klasse die we ook voor de asynchrone tests gebruikten. Aangezien elke test in deze klasse met de user control gaat werken, kunnen we deze aan het begin van elke test aan de panel toevoegen door gebruik te maken van een functie met het TestInitialize attribuut. Zie codelisting 8 voor het aanmaken van onze user control.

```
[TestClass]
public class MyControlTests : SilverlightTest
{
    private MyControl _ myControl;

    [TestInitialize]
    public void TestSetup()
    {
        _ myControl = new MyControl();
        TestPanel.Children.Add( _ myControl);
    }
}
```

CODELISTING 8: HET AANMAKEN VAN DE USER CONTROL VOOR ELKE TEST IN DE TEST INITIALIZE FUNCTIE.

Bij het maken van een user interface test zullen we in het test project met de gedefinieerde controls moeten werken. Aangezien deze standaard als internal gegenereerd worden, moeten we hiervoor het InternalsVisibleTo attribuut gebruiken om de internals van het te testen project zichtbaar te maken in onze test code. Nadat we dit gedaan hebben, kunnen we een referentie krijgen naar de textbox en de bijbehorende knop om een foto toe te kunnen voegen. Nu is het verleidelijk om direct met deze objecten te gaan werken. Het is echter beter om de AutomationPeers te gebruiken. Met deze wrappers, die voor extern aansturen van applicaties zoals bij braillelezers gebruikt worden, kunnen we alles doen, maar krijgen we gelijk de controle of een object dit wel kan cadeau. Als we zelf de achterliggende functies aan zouden roepen, moeten we

dit zelf doen en zou onze test niet meer correct zijn als de click handler van de knop naar een andere functie gezet werd. Met de AutomationPeers hebben we deze problemen allemaal niet. In codelisting 9 is te zien hoe we een foto URL in de tekstbox kunnen zetten en daarna de knop aan kunnen roepen om die foto in de lijst toe te voegen.

Door gebruik te maken van de EnqueueCallback() en de EnqueueConditional() kunnen we, net als in de asynchrone tests, zorgen dat de Silverlight runtime de mogelijkheid heeft om alle benodigde acties uit te voeren terwijl onze test code wacht.

```
[TestMethod]
[Asynchronous]
public void TestWeCanAddAValidUrl()
{
    TextBoxAutomationPeer textBoxPeer =
        new TextBoxAutomationPeer(_myControl.txtPhotoUrl);
    IValueProvider valueProvider = (IValueProvider)textBoxPeer;
    ButtonAutomationPeer buttonPeer =
        new ButtonAutomationPeer(_myControl.cmdAdd);
    IInvokeProvider buttonInvoker = (IInvokeProvider)buttonPeer;

    EnqueueCallback(() =>
        valueProvider.SetValue(
        "http://farm4.static.flickr.com/3085/3092376392_dc1aaf9eb6.
jpg"));
    EnqueueConditional(() => buttonPeer.IsEnabled());
    EnqueueCallback(() => buttonInvoker.Invoke());

    EnqueueTestComplete();
}
```

CODELISTING 9: EEN POSITIEVE TEST WAARBIJ DE KNOP AAN STAAT.

Net zo goed als we een positief resultaat willen testen, willen we ook een negatief resultaat kunnen testen. In dit geval is dat een test waarbij een foto URL die niet van http gebruik maakt aan de lijst toegevoegd wordt. In codelisting 10 wordt een dergelijke test uitgevoerd. Hierbij wordt net als in de vorige test de URL in de tekstbox gezet en de knop geklikt. Alleen in dit geval zal de AutimationPeer een ElementNotEnabledException fout geven, omdat de knop disabled is. Dat dit gebeurt kunnen we testen door de testfunctie te voorzien van een ExpectedException attribuut.

```
[TestMethod]
[Asynchronous]
[ExpectedException(typeof(ElementNotEnabledException))]
public void TestTheButtinIsDisabledWithALocalUrl()
{
    TextBoxAutomationPeer textBoxPeer =
        new TextBoxAutomationPeer(_myControl.txtPhotoUrl);
    IValueProvider valueProvider = (IValueProvider)textBoxPeer;
    ButtonAutomationPeer buttonPeer =
        new ButtonAutomationPeer(_myControl.cmdAdd);
    IInvokeProvider buttonInvoker = (IInvokeProvider)buttonPeer;

    EnqueueCallback(() =>

valueProvider.SetValue(@"file:c:\images\3092376392_dc1aaf9eb6.
jpg"));
    EnqueueCallback(() => buttonInvoker.Invoke());
}
```

CODELISTING 10: EEN NEGATIEVE TEST VAN EEN FOTO DIE NIET AAN DE LIJST TOEGEVOEGD MAG WORDEN.

Mocking frameworks en afhankelijkheden

Veel ontwikkelaars gebruiken mocking frameworks zoals RhinoMocks, Moq of TypeMock om afhankelijkheden in hun code buiten een test te houden. Helaas zijn er op dit moment nog geen mocking frameworks voor Silverlight beschikbaar. Het is echter zaak om dat regelmatig te controleren aangezien de makers van zowel Moq als TypeMock aan het kijken zijn of zij een versie kunnen maken die binnen Silverlight werkt. Op dit moment zijn we dus nog aangewezen om het handmatig maken van stubs om afhankelijkheden op te lossen.

Er is echter wel een mogelijkheid om bij het testen van Silverlight code gebruik te maken van de bestaande mock frameworks en dat is door de code vanuit Visual Studio onder de normale CLR te testen in plaats van binnen Silverlight. Dit kan omdat het wel mogelijk blijkt te zijn een referentie vanuit een normale .NET assembly te zetten naar een Silverlight, iets wat andersom niet gaat. Zodra we dat doen, hebben we uiteraard de volledige kracht van het .NET framework en alle utilities tot onze beschikking. Toch is dit geen aan te bevelen werkwijze, omdat we nu onze code testen onder een andere runtime, die zich mogelijk anders gedraagt.

Een andere manier om met afhankelijkheden om te gaan binnen onze Silverlight applicatie en testen, namelijk door gebruik te maken van een 'Inversion Of Control Container' geeft meer mogelijkheden. De keuze is nog niet erg groot, maar er zijn al een paar mogelijkheden. Op dit moment zijn er van zowel Ninject als Unity versies te downloaden die binnen Silverlight werken.

Conclusie

Binnen een modern softwareproject is het eigenlijk ondenkbaar om geen unit test te maken. Binnen Silverlight was dat een probleem, totdat het Silverlight Unit Test framework samen met de Silverlight controls vrijgegeven werd. Met dit Silverlight Unit Test framework is het makkelijk om een unit test te maken die, net zoals alle Silverlight code, binnen de browser draait. Een serieuze Silverlight ontwikkelaar kan dus eigenlijk niet zonder het Silverlight Unit Test framework.

Links

- Silverlight Toolkit: http://www.codeplex.com/Silverlight
- Microsoft Silverlight Unit Test Framework: http://code.msdn.microsoft.com/silverlightut/
- Documentatie: http://code.msdn.microsoft.com/Project/Download/FileDownload.aspx?ProjectName=silverlightut&DownloadId=3535
- Project templates: http://code.msdn.microsoft.com/Project/Download/FileDownload.aspx?ProjectName=silverlightut&DownloadId=3529
- Ninject: http://www.ninject.org
- Unity: http://www.codeplex.com/unity

Maurice de Beijer, is freelance .NET-ontwikkelaar en DevelopMentor trainer. Hij isMcrosoft MVP sinds 2005. Voor meer info zie zijn websites (http://www.theproblemsolver.nl en http://www.windowsworkflowfoundation.eu/) en zijn blog (http://msmvps.com/blogs/theproblemsolver/default.aspx).