Dieder Timmerman werkt als senior software-engineer bij Ordina.

Testen met Visual Studio 2005

GEBRUIK VAN UNITTESTS EN CODE COVERAGE

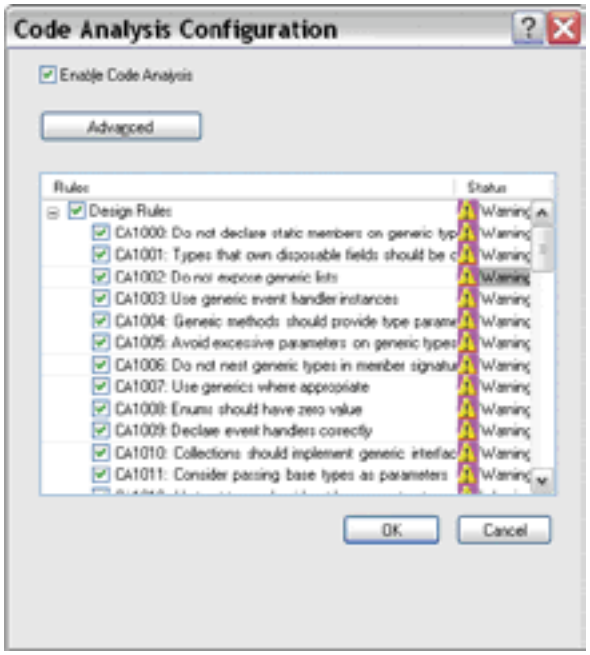
Het testen van software was ooit een lastige bezigheid. De eindgebruiker of tester moest veel tijd vrij maken en diverse malen een bètaversie van een net geïmplementeerde applicatie testen. Je kunt je afvragen hoe iemand hen dit kon aandoen. Testen waarborgt niet alleen de kwaliteit van de code, maar is ook onontbeerlijk om een acceptatietest positief te kunnen afsluiten. Helaas was het testen, meestal verricht door de eindgebruiker, voornamelijk bedoeld om hem het systeem te laten accepteren en niet om een foutvrije applicatie op te leveren.

Gebruikers raakten gelukkig snel door hun testsets heen, en werden ook snel gefrustreerd, ongerust en geïrriteerd door nieuwe fouten die in de software voorkwamen. Deze fouten ontstonden ook door het oplossen van andere fouten. De eindgebruiker moest de applicatie zelf vele malen testen om een goed eindresultaat te bereiken. Gelukkig hoort deze nachtmerrie nu tot het verleden met Visual Studio 2005. Deze bevat met Visual Studio Team Test meer testmogelijkheden, waardoor het gemakkelijker wordt om de performance gedurende de uitvoer van de applicatie te meten. Het gaat te ver om alle aspecten van het testen in één artikel uit te leggen. Dit artikel beperkt zich daarom tot de uitleg van unittests en code coverage.

Testtools onder een noemer gebracht

Met Nunit is het opleveren al een stuk aangenamer geworden voor de eindgebruiker. Het testframework stelt de ontwikkelaar in staat om de meest basale fouten direct eruit te pikken. De ontwikkelaar kan zelf een basistestset genereren op bijna elk niveau in de applicatie, waardoor hij er zelf een groot aantal fouten uit kan halen. Dat is iets waar eindgebruikers blij van worden. Zij kunnen zich daardoor meer richten op het functioneel testen van de applicatie. Helaas is het budgettair een lastige kwestie om het maken van de tests te verantwoorden. Het maken van tests kost namelijk relatief veel tijd. Toch bespaart het schrijven van unittesten uiteindelijk meer tijd en is het zijn geld dus meer dan waard. Gelukkig zijn ze daar bij Microsoft ook achtergekomen. Het testen met Visual Studio 2005 gaat een stapje verder dan met Nunit. Voor Nunit waren uitbreidingen geschreven zoals Testdriven, NCover, NcoverBrowser, NUnitAsp. In Visual Studio 2005 is deze functionaliteit als één geheel geïntegreerd. Er is goede support om websites te kunnen testen. Daarbij kunnen meer gebruikers ingesteld worden. De URL's van een test kunnen opgenomen worden en de data, die via de post- of get-methode wordt uitgewisseld, kunnen aan een webpagina worden meegegeven.

Behalve om te weten of een test goed verloopt, is het belangrijk om te bekijken in hoeverre de test de gemaakte code dekt. In Visual Studio 2005 kun je de gemaakte code direct bekijken.



Afbeelding 1. Code analysis

Soort unittest	Beschrijving
Basic flow	Unittest die de basisfunctionaliteit test.
Alternative flow	Unittest die een alternatief pad test.
Verwachte fout	Unittest die verwachte exceptie test.
Functie	Unittest die een simpele methode van één class test.
Keten	Unittest die de vanuit een root-aanroepmethode de werking tussen verschillende classes binnen één of meer namespaces test.
Interface	Unittest die de interactie met een externe component test.
User Interface	Unittest die de gebruikerinteractie nabootst en zo automatisch de applicatie test.
Web User Interface	Idem als User Interface, maar dan voor webapplicaties.
Executie	Unittest die het opstarten van de applicatie test.
Autorisatie	Unittest die de rechten van de gebruikersrollen test.
DatabaseDriven	Gebruikt de database om testgevallen op te halen en deze voor de test te gebruiken.
Negative Test	Unittest die test of de code tegen 'onverwachte' inputparameters kan.

Tabel 1. Soorten unittests

Testsectie attributen	Beschrijving
TestClass	Class die gebruikt wordt om te testen.
TestInitialize	Initialiseren van de test per methode.
ClassInitialize	Initialiseren van de test per class. De set-up van de test wordt als eerste uitgevoerd om de test uit te kunnen voeren. Deze wordt gebruikt om databaseconnecties te initialiseren, transacties te starten en configuratiebestanden.
AssemblyInitialize	Idem als ClassInitialize, maar deze wordt uitgevoerd voor een eenmalige initialisatie van alle testen binnen de assembly.
TestCleanup	Het opruimen van veranderingen die door het uitvoeren van een test zijn ontstaan. Hierbij kan bijvoorbeeld de transactie afgebroken worden, de connectie worden gesloten en of bewerkingen ongedaan gemaakt worden.
ClassCleanUp	Zelfde als TestCleanup, maar dan per class. Deze methode wordt aan het einde van alle testmethoden in de class uitgevoerd.
AssemblyCleanUp	Idem als ClassCleanUp, maar deze wordt aan het eind van alle testen in de assembly uitgevoerd.
TestMethod	Methode die code bevat die getest moet worden.

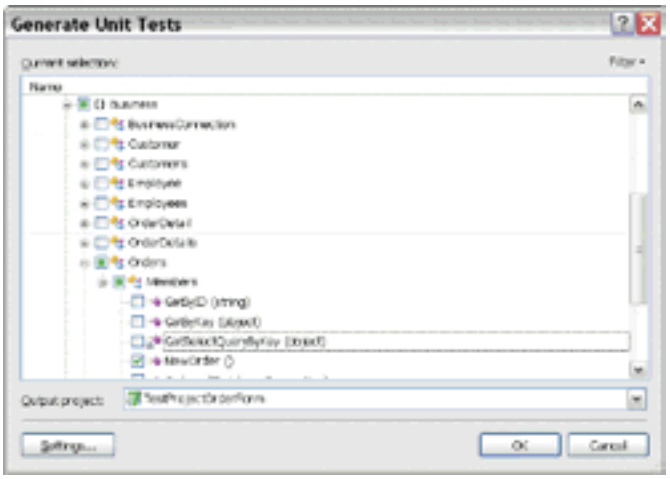
Tabel 2. Overzicht testattributen

De groene code is uitgevoerde code en de rode code is niet uitgevoerde code. Een andere manier van testen is het onderzoeken of de code voldoet aan gestelde programmeer- en naamgevingstandaarden. FxCop is een tool die bekend staat om zijn harde aanpak en hij zorgt ervoor dat ontwikkelaars zich aan deze standaard moeten houden. In Visual Studio 2005 is FxCop vervangen door code analysis. Code analysis bevat diverse rulesets die eisen stellen waaraan de code moet voldoen. Zo wordt de ontwikkelaar gedwongen om variabelen, voordat een waarde van de variabele wordt opgevraagd, te initialiseren en variabelen op te ruimen die helemaal niet gebruikt worden. Net als FxCop zijn rules makkelijk uit te zetten, óók op specifieke codelocaties. Zie afbeelding 1.

Soorten unittests

Voor een totaaltest van een applicatie zijn meer soorten unittest nodig. De basisprocestest moet het basisproces testen. Het basisproces is het proces, waarbij een taak wordt uitgevoerd zonder op alternatieven en foutsituaties te letten. Voor elke alternatieve vertakking in het proces dient een aparte test te worden geschreven. Er zijn immers foutsituaties die tot afbreken van het proces leiden. Voor de verwachte foutsituaties kan ook een test worden geschreven. Het schrijven kan voor, tijdens en na de implementatie van de applicatiecode gedaan worden. Het schrijven van unittests vóór implementatie is een lastig karwei. Het design, de procesflow, maar ook de code dienen van te voren helder gespecificeerd te zijn. Tijdens het schrijven moet skeleton-code gemaakt worden voor classes, methoden en hun parameters. Pas als de unittest af is, krijgen de classes en methoden een body. Het voordeel is dat de unittest het meest voldoet aan functionele specificaties. Het schrijven van unittests tijdens de implementatie gaat makkelijker. Men is zich bovendien meer bewust van mogelijke foutsituaties. Het nadeel is dat de test aangepast wordt aan de werking van de code en minder voldoet aan het functionele aspect. Dit kan ertoe leiden dat de designer of de tester de code niet accepteert. Achteraf schrijven van de unittest is iets moeilijker. Alle afwijkende procesflows dienen opnieuw onderkend te worden. En voor elke flow dient een test geschreven te worden.

Het continu kunnen uitvoeren van de unittest zorgt voor minder fouten in de code. Welke keuze ook wordt gemaakt, het vraagt een goede helikopterview van de ontwikkelaar om de test te laten voldoen aan het functioneel gevraagde en tegelijkertijd



Afbeelding 2. Creëren van een test in Visual Studio 2005

de broncode afdoende moet worden gedekt. Het is erg bewerkelijk om voor elke class een test te schrijven en vaak is er ook niet de tijd voor. Classes echter die uitgesloten worden van unittests, dienen correct te werken. Het blijft dus een kwestie van goede afwegingen maken en de code coverage per assembly op peil te houden. Met hulp van databasegegevens uit de database kan een stuk code in meer toestanden getest worden en daardoor is werking van de code betrouwbaarder. Ook testen met een andere invalshoek verbetert de betrouwbaarheid van de code. Een test kan slagen indien enkel de aanroep van de code slaagt. Ook het resultaat van gewijzigde data echter kan gecheckt worden. Dit kan kort met slechts enkele beweringen, maar ook uitgebreid worden gedaan. In tabel 1 staat een overzicht met diverse soorten unittests met elk een eigen invalshoek.

Maken van een unittest

Een test bestaat uit een drietal delen. Een unittest begint met een set-up. Hiermee kunnen taken worden uitgevoerd die nodig zijn om de test te voor te bereiden. Vervolgens kunnen de diverse testen uitgevoerd worden. Als laatste kan het één en ander worden gedaan om de sporen uit te wissen die de test heeft achtergelaten. Het initialiseren en opruimen van de test is niet verplicht. De verschillende delen worden met een testattribuut op een methode binnen een testmethode, testclass of assembly aangegeven; zie tabel 2.

Het genereren van een nieuwe test is niet moeilijk. Binnen een bestaande codeclass kan via het contextmenu een 'create test' gedaan worden. Microsoft heeft ervoor gekozen een testproject te genereren. Daardoor is het niet nodig om libraries voor het testen op te nemen in het project of een nieuwe configuratie aan te maken die specifiek bedoeld is voor het testen. De code en tests worden na elkaar gecompileerd. De test kan daarna via de testmanager worden uitgevoerd. Dit uitvoeren gebeurt met hulp van invocation. Het betekent dat er een programmaatje meedraait dat doet of hij de actor is. Deze actor voert de

```
Person = new Person();
//waar want persoon is niet via het Null-pattern gecreëerd.
Assert.IsNotNull(Person);
//waar want een nieuw persoon heeft nog geen naam
AssertIsTrue(Person.Naam==String.Empty);
AssertIsFalse(Person.Naam="Bert");
//test of de opgetreden fout een database fout is, zo ja breek de test af
Assert.AreNotEqual<Type>(typeof(sqlException),
   exception.GetType(),"Database fout");
```

Codevoorbeeld 1.
Gebruik van Assert

```
[TestFixture]
public class Exceptionest
{
   [Test]
   [ExpectedException(typeof(System.NotImplementedException))]
   public void TestImportantFunction()
   {
      throw new NotImplementedException("Function under construction");
   }
}
```

Codevoorbeeld 2.

Test die een verwachte fout moet opleveren

testset(s) uit. Ook het aansturen van de front-end kan worden geregeld voor zowel web- als Windows-applicaties. Bij het aanmaken van een 'create test' (zie afbeelding 2) kunnen de klassen en methoden binnen de solutie gekozen worden. Deze aanroep kan via het testmenu worden gestart. Elke nieuwe test kan in hetzelfde of in een nieuw testproject uitgevoerd worden. Na het selecteren van de te testen onderdelen worden automatisch skeleton-testmethoden en testclasses aangemaakt. Het is nu aan de ontwikkelaar of tester om invulling te geven aan deze skeletons. Een goed advies is om niet klakkeloos alles aan te vinken en ook vooral te denken aan de verschillende soorten unittests.

Binnen de testmethode kunnen beweringen getest worden. Indien niet aan een bewering wordt voldaan, zal de test afgebroken worden en heeft de test gefaald. Dit kan een aantal dingen betekenen. Er is een fout gemaakt in de applicatie en die moet hersteld worden. Het kan ook zijn dat de bewering niet klopt. Ten derde kan het zijn dat de omgeving veranderd is, waardoor de uitkomst van de bewering moeilijk te testen is. Dit kan worden vertaald als een onbepaalde bewering. De beweringen worden via de Assert-class aangeroepen. Deze class bevindt zich in de nogal lange namespace 'Microsoft.VisualStudio.TestTools.UnitTesting'. De Assert-classes maken het mogelijk instanties en eigenschappen te testen op waarden en verwachte objecttypen.

Codevoorbeeld 1 toont het gebruik van de Assert-class. De assert-functies kunnen worden uitgebreid met een 'message' en met verscheidene 'message parameters'. Als aan een bewering niet voldaan wordt, kan het bericht extra informatie aan de ontwikkelaar geven. In deze string kunnen berichtparameters worden opgegeven tussen accolades. De berichtparameters worden in de boodschap getoond.

Een collectie is ook in het testframework makkelijk te testen. Collecties zijn een veelgebruikt mechanisme en bevatten veel

Assert-method	Omschrijving
AreEqual	Test of waarden overeenkomen.
AreEqual<T>	Test of het object niet aan het objecttype voldoet.
AreNotEqual	Test of waarden niet overeen komen.
AreNotEqual<T>	Test of het object niet aan het objecttype voldoet.
AreSame	Test of objecten hetzelfde zijn.
AreNotSame	Test of objecten niet hetzelfde zijn.
IsFalse	Test of conditie onwaar is.
IsTrue	Test of conditie waar is.
IsNull	Test of object geen waarde heeft.
IsNotNull	Test of object een waarde heeft.
IsInstanceOfType	Test of object van een specifiek type is.
IsNotInstanceOfType	Test of object niet aan een specifiek type voldoet.
Inconclusive	Geeft het besluit van de test terug aan de gebruiker. Het is meer een opmerking. De test faalt niet, maar slaagt ook niet. Gooit altijd een 'AssertInConclusiveException' in de lucht.

Table 3. Overzicht van methods van de Assert-class

```
[Test]
[Ignore]
public void TestNotReady()
{
   throw new NotImplementedException("Test under construction");
}
```

Codevoorbeeld 3.

Test, die niet uitgevoerd wordt.

items. In een test zou je de collectie ook willen meenemen. Specifiek voor een collectie is een CollectionAssert-class beschikbaar. Enkele functies gaan uit van een resultaatcollectie, die vergeleken wordt met een *verwachte* resultaatcollectie. Tabel 4 toont de 'CollectionAssertClass' met de mogelijke statische methoden.

Er is ook een StringAssert-class beschikbaar. Deze kent enkele methoden om te kijken of een bepaalde waarde in de string zit, of de string gelijk is aan een andere string, enzovoort. Naast het testen op waarden en verwachte typen kan er ook getest worden op verwachte fouten. De verwachte fout wordt boven de testmethode in een verwacht exceptie-attribuut aangegeven. Indien de fout niet optreedt tijdens de test, zal de test falen. Aangezien de testmethode vaak een methode in de applicatie aanroept, dient deze methode de exceptie niet zelf af te handelen. Zie codevoorbeeld 2.
Het komt voor dat een test om een bepaalde reden niet uitgevoerd kan worden. Dit kan zijn omdat de buildserver de test per interval zal uitvoeren en de test nog niet operationeel is. Of de test levert een verandering in het systeem op, die zonder gebruikersinterventie kan leiden tot het falen van andere testen of resulteert in een andere uitkomst bij het herhalen van de test. In dit soort gevallen kan een test beter (tijdelijk) uitgeschakeld worden. Het 'Ignore'-attribuut zorgt ervoor dat een bepaalde test niet zal worden uitgevoerd. Zie codevoorbeeld 3.

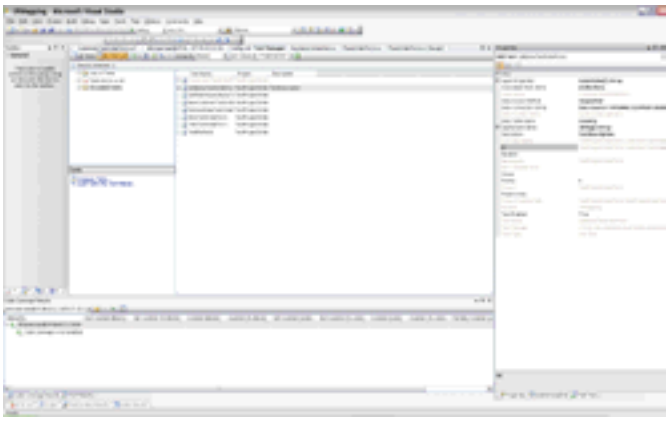
Bij het misgaan van een testbewering wordt deze test direct afgesloten door een 'AssertFailedException'. Wanneer tijdens de test een onverwachte exceptie optreedt, wordt de test ook direct afgebroken. De testmanager zal andere testen dan nog wel uitvoeren. Testen kunnen in debugmode uitgevoerd worden, zodat gekeken kan worden waarom de test faalt. In een 'testresult tab' worden de resultaten van de test getoond. Door op een test te klikken worden de details van de test weergegeven, waaronder de opgetreden exceptie. Zie afbeelding 3.

Bij een test kan een aantal eigenschappen worden gezet. Zo kan de beschrijving worden gezet door het attribuut 'Description'

Collection Assert-methode	Beschrijving
AllItemsAre-InstancesOfType	Test of de items van een collectie allemaal van het opgegeven type zijn.
AllItemsAreNotNull	Test of er geen lege items in de collectie zitten.
AllItemsAreUnique	Test of er geen twee dezelfde items in de collectie zitten.
AreEqual / AreNotEqual	Test of de collectie dezelfde items bevatten/ niet dezelfde items bevatten in dezelfde volgorde.
AreEquivalent/ AreNotEquivalent	Test of collectie dezelfde/ niet dezelfde items bevat.
Contains / DoesNotContain	Test of de collectie een item bevat/ niet bevat.
IsSubsetOf	Alle elementen in de eerste collectie ook in de tweede collectie zitten.
IsNotSubsetOf	De eerste collectie items bevat die niet in de tweede collectie zitten.

Tabel 4. Statische methoden van de CollectionAssertClass

Afbeelding 3. Testresultaten



Afbeelding 4. Managen van een test

aan de testmethode toe te voegen. Het zetten van deze attributen gaat het makkelijkst via de TestManager. Dit scherm in Visual Studio toont alle gedefinieerde testen. Elke test kan afzonderlijk geselecteerd worden. Alleen geselecteerde onderdelen zullen getest worden. Microsoft gebruikt het 'DataTable'-object voor het ophalen en bewerken van databasegegevens. De 'DataTable' gebruikt een dataconnectie om de database aan te roepen. Beide objecten kunnen bij een testmethode dan ook als attribuut worden opgenomen. Dit maakt het gemakkelijk om testgegevens voor de test uit een database te halen. Ook kan opgegeven worden dat de gegevensrijen sequentieel of at random worden opgehaald. Dit maakt de test minder voorspelbaar en dus reëler. De attributen van een testmethode vormen de 'TestContext'. Deze variabele wordt standaard als eigenschap opgenomen van een gegenereerde 'TestClass'. Zie afbeelding 4 met informatie over de tests.

De attributen, die op een test gezet worden, zijn ook in code zichtbaar. Zie codevoorbeeld 4.

De 'TestContext'-property kent een aantal eigenschappen dat tijdens de test gebruikt kan worden. Zo heeft de context een timer. Deze timer begint te lopen zodra de test begint en eindigt zodra de test stopt. Tijdens de test kunnen deze waarden uitgevraagd worden. Er kan ook een regel worden geschreven naar de trace log. Hierbij kan de eigenschap 'TestName' worden opgevraagd, zodat het terugzoeken later makkelijker verloopt. De eigenschap 'TestDir' geeft aan in welke directory de test wordt uitgevoerd. De testdirectory wordt door Visual Studio zelf gegenereerd. Het is niet aan te raden om configuratiebestanden naar de testdirectory te kopiëren. De test weet namelijk niet van zijn directory. Beter is het om een vaste directory te gebruiken, zoals die waar de broncode staat. Er bestaat voor het loggen van testgegevens naar een bestand de eigenschap 'TestLogsDir', waar het schrijven van testloggegevens wordt bijgehouden. Ook is er de eigenschap 'DeploymentDir', waar de dll of executable van de testapplicatie staat. De 'TestContext' bevat ook support voor een database-driven test. Via de testmanager kunnen de 'DataTable' en de 'DatabaseConnection' op een testmethode gezet worden. Het 'DataSource'-attribuut van de testmethode krijgt hierdoor de waarde van de connectiestring en de naam van de 'DataTable'. De 'DataConnection' en de DataRow kunnen via de 'TestContext' gedurende de test worden uitgelezen. De 'DataRow'-eigenschap is de pointer binnen de datasource, die de huidige rij van de 'DataTable' aangeeft. Deze kan een belangrijke rol vervullen bij het niet-sequentieel doorlopen van de 'DataTable'-rijen. De TestProperty is een attribuut dat een fictieve property symboliseert. Op deze eigenschap kan bij declaratie van het attribuut een waarde gegeven worden. De waarde is echter read-only. Het gebruik van de testproperty kan van pas komen bij een reflectietest, waarbij een waarde van een eigenschap wordt uitgelezen.

Testen met websites

Voor het testen van een website begin je een nieuw project in de solution. Via het menu 'test' kun je een nieuw testproject starten met een template voor een webtest. De recorder van de webtest wordt gelijk gestart. Geef in de URL een verwijzing op naar website die getest moet worden. Je kunt door meer pagina's en websites surfen. De recorder kan gestopt worden en op een later tijdstip weer aangezet worden. Resultaten worden in een bestand met de extensie 'webtest' weggeschreven. De webtest kan vervolgens gestart worden, waarna alle opgenomen pagina's worden gestart. Via een venster kan de browseractiviteit worden waargenomen. Alle form-parameters, die tijdens de opname worden ingevuld, worden eveneens opgevoerd. De lijst met opgenomen URL's en form-variabelen kunnen met de hand aangepast worden. De responsetijden en grootte van de pagina - en ook die van zijn onderdelen - worden gemeten. Ook de request- en response-informatie worden overzichtelijk bijgehouden. Hierdoor kan precies worden gezien wat er over de lijn gecommuniceerd wordt. Zie afbeelding 5.

Aan een URL kunnen vier soorten validation-rules toegevoegd worden, die de inhoud van de pagina testen; zie tabel 4. Aan de URL kunnen vijf soorten extraction-rules toegevoegd worden, die de query-string of de http van de pagina testen. De uitkomst(en) wordt in de WebContext-eigenschap gezet; zie tabel 5.

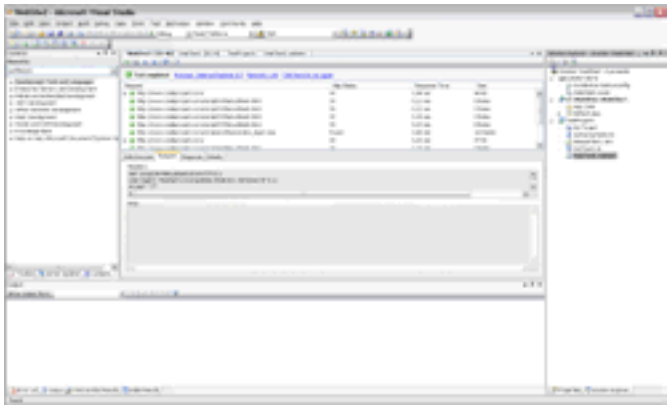
De volgende stap is het creëren van de testclass die de waarden uit de Extraction Rule kan gebruiken. De testclass wordt geken-

```
[TestMethod(),
Description("TestDescription"),
DataSource("System.Data.SqlClient", "Data Source=XP1BHGC1J;Initial
   Catalog=Northwind;Integrated Security=True", "Country",
DataAccessMethod.Sequential),
Priority(1)]
public void GetByKeyTestOrderForm()
   ...
```

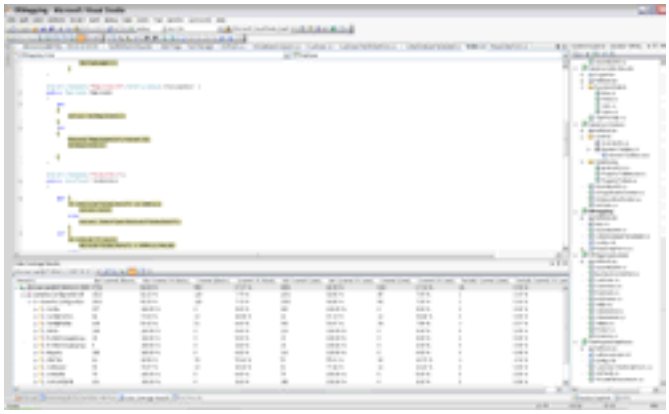
Codevoorbeeld 4.
Attributen die op een testmethode gezet kunnen worden

Validation Rule	Omschrijving
ValidationRuleFindText	De pagina moet deze tekst bevatten.
ValidationRuleRequired-AttributeValue	Een attribuut van een opgegeven tag moet een bepaalde waarde bevatten.
ValidationRuleRequestTime	De request van de pagina moet binnen deze responsetijd beantwoord zijn.
ValidationRuleRequiredTag	De HTML-pagina moet één of meer keren een specifieke tag bevatten.

Tabel 4. Validation-rules voor webtest



Afbeelding 5: Uitvoeren van een webtest



Afbeelding 6. Weergave van code coverage

merkt door een TestClassAttribuut; zie tabel 6. De TestClass kan de TestContext uitlezen en zo de waarden van de extraction-rules uitlezen.Er kunnen testmethodes gemaakt worden die een bepaalde pagina testen. Deze testmethodes hebben extra attributen nodig om een pagina te testen. De standaardwaarden kunnen ook via de testrun-configuratie gezet worden.

Met page.FindControl kunnen controls worden opgespoord. Aangezien de gevonden controls van een verwacht type zijn, kan de uitkomst naar een control getypeerd worden. Deze controls kunnen vervolgens geëdit worden via de PrivateObject-class. Deze class kan via reflectie niet-public methoden en variabelen bewerken. In de constructor kan de te reflecteren class, in dit geval Page, worden opgenomen. Met de invoke kan een event van een control getriggered worden en ook eventargs kunnen eventueel worden meegegeven. Met deze insteek kan automatisch ingelogd worden, maar er zijn talloze andere mogelijkheden. Zie codevoorbeeld 5.

Binaire data versturen

Het versturen van binaire data naar een pagina is minder eenvoudig. Standaard wordt gebruik gemaakt van de FormPostHTTPBody. Je zou willen dat er zo iets bestond als een BinaryPostHTTP body-class. De belangrijkste methode is de WriteHTTPBody-methode, die het feitelijke werk uitvoert. Een dergelijke class is te maken door de class af te leiden van de IHttpBody-class. De bron voor deze code is te vinden op http://blogs.msdn.com/joshch/archive/2005/08/24/455726.aspx. Zie codevoorbeeld 6, de class kan gebruikt worden zoals te zien is in codevoorbeeld 7.

Code coverage

Hoe hou je in de gaten of de unittests ook alle code testen? Dat kan door de dekkingsgraad te meten. De code coverage kan niet in de debugmodus gestart worden. Na het starten van de test(s) wordt de totale code coverage gemeten. Deze code coverage is zeer gedetailleerd. Zo kan per assembly, per class, per functie de coverage per codeblock of per coderegel gemeten worden. Ook kan worden weergegeven of een regel deels gedekt is. Een voorbeeld van dit laatste is het uitvoeren van een OR-statement.

ExtractionRule	Omschrijving
ExtractAttributeValue	Test de naam in een querystring of de waarde van een attribuut.
ExtractHTTPHeader	Test een header-parameter.
ExtractRegularExpression	Zoekt tekst in de response op basis van een regular expression.
ExtractText	Zoekt tekst in het document op basis van een zoekpatroon zoals een regular expression.
ExtractHiddenFields	Zoekt hidden fields van de response.

Table 5. Extraction-rules voor webtest

Hierbij kan het zijn dat het eerste wel uitgevoerde statement reeds voldeed, waardoor het tweede statement op dezelfde regel niet is uitgevoerd. Het verschil tussen regels en blocks is dat blocks alle statements en code meepakt, terwijl bij regels het aantal wel en niet uitgevoerde regels telt.

Een interessant snufje is dat de geraakte code in de kleuren groen, lichtrood of rood te bewonderen zijn; zie afbeelding 6. Zo krijg je direct inzicht welke code door de tests wel of niet geraakt zijn. De developer kan hierdoor bepalen of extra unittests nodig zijn, want het gebeurt maar al te vaak dat sommige code bij hoge uitzondering wordt uitgevoerd. En juist deze code is daarom foutgevoelig. Van te voren kan worden afgesproken hoe hoog de dekkingsgraad minimaal moet zijn. Voor framework-achtige zaken mag 90% geëist worden, omdat dit een fundament is voor de hele applicatie. Voor back-end-zaken 80%. Op de back-end worden belangrijke services aangeroepen, business-rules gecheckt en data gepersisteerd. Voor de front-end - exclusief userinterface - zo'n 60%. Dit is de meest dynamische code en kan ook visueel getest worden. De genoemde percentages zijn slechts voorbeeldpercentages. Het maximale van 100% is lastig te halen, omdat het bijvoorbeeld juist de bedoeling is om geen exceptie af te laten gaan. En hoe simuleer je dat de harde schijf vol is, waardoor een exceptie wordt geworpen. Omgekeerd kan het zijn dat sommige functies 0% scoren. Voor deze functies kunnen aparte unittests geschreven worden. Mijn advies: ga er pragmatisch mee om, want unittests maken kost veel tijd.

Omvallen van testen

Testen hebben de neiging vaak om te vallen. Dit betekent dat ze door een bepaalde oorzaak opeens niet meer werken. Hier zijn verschillende redenen voor. De applicatie verandert, waardoor nieuwe situaties ontstaan. Ook heeft de vulling van gebruikte testdata een rol. Belangrijk is om een vaste en consistente testvulling te gebruiken, net als dat het testen de inhoud van deze vulling dient te controleren. Tests die een wijziging in de omgeving maken, waardoor ze zichzelf of andere unittests laten omvallen, dienen aangepast te worden of handmatig na de test gestart te worden. Ook de unittest dient dan aangepast te worden. Ontwikkelaars doen er goed aan de verschillende testen te categoriseren. Kritische unittests mogen niet omvallen. En

Attribuutnaam	Omschrijving
HostType	Default of ASP.NET
UrlToTest	URL-pagina.
WebServerType	Test in de developer webserver of IIS.
WebapplicationRoot	De rootdirectory van de website.
PathToWeb	MapPath of de URL. Dit is de werkelijke folder van de URL.

Tabel 6. Attributen van testclass

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using Microsoft.VisualStudio.QualityTools.UnitTesting.Framework;
using Microsoft.VisualStudio.QualityTools.UnitTesting.Web;
using System.IO;
using System.Collections;

[TestClass]
public class UnitTestLogon
{
   private TestContext testContextInstance;
   public TestContext TestContext
   {
      get { return testContextInstance; }
      set { testContextInstance = value; }
   }

   [TestMethod]
   [HostType("ASP.NET")]
   [UrlToTest(http://localhost/Test/)]
   [WebServerType(WebServerType.WebDev)]
   [WebApplicationRoot("/Test")]
   [PathToWeb(@"C:\Projects\Website\VisualStudio2005Beta2\Test")]
   public void UnitTestMethodLogon()
   {
      Label lblLogin = (Label)page.FindControl("lblUserName");
      TextBox loginTextBox = (TextBox)page.FindControl("txtUserName");
      TextBox passwordTextBox = (TextBox)page.FindControl("txtPassword");
      Button loginButton = (Button)page.FindControl("cmdLogon");

      loginTextBox.Text = "UserName";
      passwordTextBox.Text = "Password";
      PrivateObject po = new PrivateObject(page);
      po.Invoke("cmdLogon_Click", loginButton, EventArgs.Empty);
      Assert.AreEqual("Invalid login.", errorLabel.Text);
   }
}
```

Codevoorbeeld 5.

```
public class BinaryHttpBody : IHttpBody
{
   private string _contentType;
   private Stream _stream;

   public BinaryHttpBody(string contentType, byte[] bytes)
      : this(contentType, new MemoryStream(bytes, false)) {}

   public BinaryHttpBody(string contentType, Stream stream) {
      _contentType = contentType;
      _stream = stream;
   }

   public BinaryHttpBody() { } : base()

   public string ContentType {
      get { return _contentType; }
      set { _contentType = value; }
   }

   public Stream Stream {
      get { return _stream; }
      set { _stream = value; }
   }

   public void WriteHttpBody(WebTestRequest request,
        System.IO.Stream bodyStream) {
      if (_stream != null && _stream.CanRead) {
         try {
            byte[] buffer = new byte[8192];
            int bytesRead;
            while ((bytesRead = _stream.Read(buffer, 0, buffer.Length)) > 0)
               { bodyStream.Write(buffer, 0, bytesRead);  }
            }
         finally { _stream.Close();  }
      } //if
   }//method
   public object Clone() {  throw new NotImplementedException(); }
}
```

Codevoorbeeld 6.

```
WebTestRequest request1 = new WebTestRequest(
   "http://localhost/testUpload.aspx");
request1.Method = "POST";
BinaryHttpBody binaryBody = new BinaryHttpBody("application/octet-stream",
        new byte[] { 0x07, 0xA2, 0xFF, .. });
request1.Body = binaryBody;
```

Codevoorbeeld 7.

unittests die behoort bij de opgeleverde functionaliteit mag ook niet omvallen. Andere tests kunnen onderverdeeld worden in: 'under construction', 'under investigation' en 'Manual'.

Eén geheel

Er is veel mogelijk met unittesten. In Visual Studio 2005 worden de bekendste open source-tools overbodig. Het testen, voorheen met diverse tools, vormt nu één geheel. Het unittesten kan voor zowel web- als Windows-applicaties ingezet worden. Unittesten kunnen met een verschillende insteek gebruikt worden, zoals een keten- of functie-unittest. Met diverse Assert-functies of webpage-rules kan de verwachte uitkomst van variabelen getest worden. Ook de verwachte excepties kunnen getest worden. Een testclass kan de webcontext-eigenschap gebruiken om de context van de test te zetten of uit te lezen, en biedt support voor DataDriven-tests. Code coverage is een goede hulp om te meten hoeveel de unittest de geschreven code raakt.

Dieder Timmerman werkt als senior software-engineer bij Ordina (http://www.ordina.nl). Hij is MCSD .Net-gecertificeerd en legt zich voornamelijk toe op het ontwikkelen van .NET-applicaties met gebruik van ontwikkelstraattechnologie. Voor vragen en opmerkingen is Dieder te bereiken op via dieder.timmerman@ordina.nl

Nuttige internetadressen

Josh Chrisitie blog: http://blogs.msdn.com/joshch/archive/2005/08/24/455726.aspx

A Unit Testing Walkthrough with Visual Studio Team Test: http://msdn.microsoft.com/library/en-us/dnvs05/html/VSTSUnitTesting.asp