Dick Dijkstra en Christian Siegers

Test Driven development bij integratieoplossingen

TESTEN BIJ BIZTALK-PROJECTEN

Het goed testen van software is essentieel om een goed product te kunnen opleveren. Uit eigen ervaring blijkt dat er lang niet altijd goed of niet goed genoeg getest wordt. Dat geldt ook voor BizTalk-integratieoplossingen. Vlak voor het in productie gaan moeten er dan nog problemen opgelost worden. Dit artikel beschrijft welke tooling beschikbaar is om de kwaliteit van een BizTalk-applicatie te verbeteren.

BizTalk-applicaties vormen vaak de ruggengraat van organisaties. Kwaliteit en stabiliteit zijn daarom erg belangrijk. Als zo'n oplossing te veel down-time heeft, kan dit leiden tot grote technische of zelfs politieke problemen. Goed testen is dus van groot belang. Waarom gebeurt dit dan niet altijd? Eén van de redenen is de complexiteit van BizTalk-applicaties. Deze applicaties bestaan uit verscheidene programma's die vaak op verschillende platforms en met verschillende technieken zijn gebouwd. Bovendien zijn de applicaties meestal ook nog eens verspreid over verschillende locaties. De tweede reden heeft met tijd te maken. Vaak wordt een testfase achteraan in het project, na de constructiefase gepland. Het risico hierbij is dat de testfase bij eventuele uitloop niet of nauwelijks wordt uitgevoerd, omdat de deadline toch gehaald moet worden. Een bijkomend probleem is dat het verifiëren van testen voor veel handwerk zorgt. Stel dat door BizTalk informatie uit een bericht in een SQL-database wordt geschreven. Een tester kan dan gebruik maken van Query Analyzer om de informatie te controleren met behulp van een SQL-statement. Als dit één keer moet plaatsvinden, is dat niet problematisch, maar elke test wordt veel vaker dan één keer uitgevoerd. Dat zorgt ervoor dat het testen een weinig uitdagende en bovendien foutgevoelige procedure wordt. Om de beschreven problematiek te ondervangen, kunnen we gebruikmaken van bestaande tooling. Deze tooling ondersteunt zowel bij het unit-testen als bij het functioneel testen van BizTalk-applicaties. De volgende paragrafen geven aan welke tooling er is en hoe deze ingezet kan worden.

Tooling voor unit-testen

Unit-testen wordt toegepast op componenten binnen een BizTalk-applicatie. Volgens de testdriven development-methodiek, in .NET Magazine 19 beschreven door Gerard van der Pol, worden unit-tests ontwikkeld voordat de functionele code wordt gebouwd. Diezelfde methodiek kan worden toegepast in de constructiefase van een project waarin een BizTalk-applicatie wordt ontwikkeld: voordat bijvoorbeeld een custom pipeline-component wordt gebouwd, moeten eerst unit-tests worden gedefinieerd. Deze tests falen initieel en de ontwikkelaar bouwt functionele code totdat de tests slagen. De tools voor het unit-testen van de componenten zijn NUnit of Visual Studio 2005 Unit Testing. Beide tools bieden goede oplossingen om unit-tests te maken en uit te voeren. Daarnaast bieden ze een goede interface die op eenduidige wijze het resultaat van test weergeeft, wat essentieel is binnen de testdriven development-methodiek. Deze methodiek schrijft namelijk voor dat in één oogopslag duidelijk moet zijn of de test geslaagd is of niet. Het volgende codevoorbeeld laat zien hoe een unit-test wordt opgebouwd wanneer NUnit wordt ingezet. Met deze unit-test wordt een library getest die binnen een orchestration wordt aangeroepen. Deze library bevat onder andere een methode die op basis van het bericht dat wordt meegegeven als parameter een zekere status retourneert.

De klasse waarin unit-tests worden opgenomen, heeft het attribuut *TestFixture* (Visual Studio Unit Testing gebruikt hier *TestClass*) en elke methode die een unit-test beschrijft heeft het attribuut *Test* (Visual Studio Unit Testing gebruikt hier *TestMethod*). Het attribuut *SetUp* (Visual Studio Unit Testing gebruikt hier *ClassInitialize*) wordt gebruikt om aan te geven dat de methode bijbehorend bij dit attribuut uitgevoerd moet worden voordat de daadwerkelijke tests plaatsvinden. In de methode waar de test plaatsvindt, wordt gebruikgemaakt van de klasse *Assert*; dit is een hulpmiddel binnen het NUnit-framework dat helpt bij het checken of de door de pipeline-component getransformeerde stream gelijk is aan de verwachte stream.

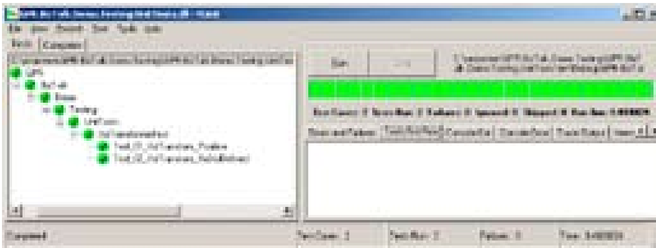
De resultaten van de unit-test staan weergegeven in een GUI die wordt meegeleverd met het NUnit-framework; afbeelding 1. Als de assembly met de unit-test gecompileerd, is kan deze in de GUI worden geladen. De GUI geeft op eenduidige wijze aan of een test slaagt met behulp van een rode bol of een groene bol. Verder biedt de GUI een goede manier om unit-tests te groeperen. Naast de GUI

```
[TestFixture]
public class UnitTests
{
   [Test]
   public void XslTransformTest()
   {
      //read streams from files
      Stream xml = File.Open(@"xmlfiles\initial.xml", FileMode.Open,
        FileAccess.Read);
      Stream html = File.Open(@"htmlfiles\expected.htm", FileMode.Open,
        FileAccess.Read);

      //do transform
      Stream transformed = XslTransformer.TransformMessage(xml,
        @"xslfiles\transformSubmarine.xsl");

      //the transformed stream should be equal to the input html stream
      Assert.IsTrue(StreamUtils.Compare(transformed, html));
   }
}
```

Codevoorbeeld. 1



Afbeelding 1. NUnit GUI

biedt NUnit de mogelijkheid de unit-test-assembly aan te roepen vanaf de console en de resultaten daarin te tonen. Dat komt zeer van pas wanneer unit-tests worden opgenomen in het buildproces.

Tooling voor functioneel testen

Functioneel testen spitst zich toe op het verifiëren van een compleet end-to-end scenario in de BizTalk-applicatie. De BizTalk-applicatie is hier een black box; bij het uitvoeren van een functionele test gaat een bericht met bekende inhoud naar een bekende URL. Vervolgens controleert een tester of de applicatie naar verwachting werkt. De tester kijkt bijvoorbeeld of een bericht op de juiste bestandslocatie staat en of dit bericht de verwachte inhoud heeft. Bij functioneel testen is het belangrijk om alle mogelijke paden te testen. Dat betekent dat niet alleen de goedsituaties, maar ook de foutsituaties moeten worden getest. Bovendien moeten alle input- en output-url's, alle soorten berichten, alle (custom) code en alle orchestrations worden geraakt.

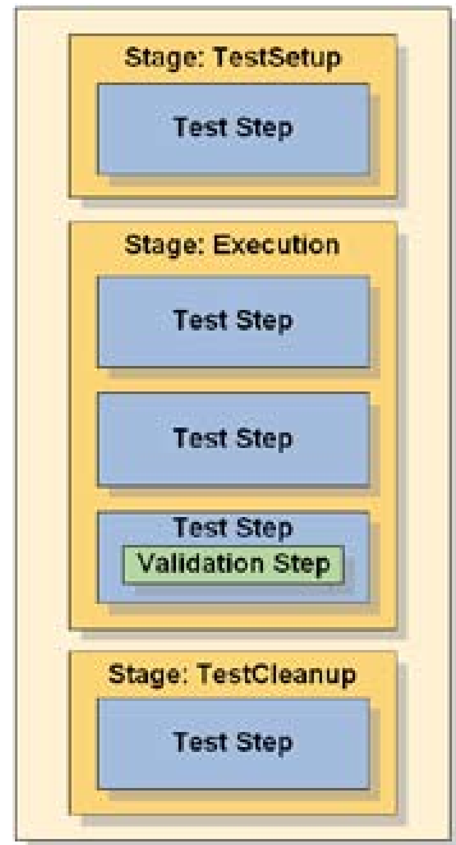
Functioneel testen gebeurt vaak met de hand. Dit kost veel tijd en is bovendien erg foutgevoelig. Daarom is het verstandig om niet alleen het unit-testen, maar ook het functioneel testen te automatiseren. Wanneer van geautomatiseerd functioneel testen gebruik wordt gemaakt, wordt al in het begin van het constructieproces in een project geïnvesteerd in het ontwikkelen van tests. Elke keer als de tests daarna plaatsvinden, betaalt dit zich terug. Het bouwen van testcases mag niet extreem veel inspanning kosten. Daarom vindt de ontwikkeling van functionele tests vaak plaats binnen een framework dat het mogelijk maakt de testcases op eenvoudige wijze snel te definiëren.

Een dergelijk framework, genaamd BizUnit, is bij uitstek geschikt voor het testen van BizTalk-applicatieen. Overigens kan BizUnit ook prima gebruikt worden om het testen binnen niet-BizTalk-applicaties te ondersteunen, bijvoorbeeld een enkele webservice. BizUnit is een framework en moet daarom gehost worden binnen een andere applicatie. De tools NUnit en Visual Studio Unit Testing zijn hiervoor uitermate geschikt. Het onderstaande codevoorbeeld beschrijft hoe een BizUnit-testcase wordt aangeroepen vanuit Visual Studio Unit Testing.

Het BizUnit-framework maakt gebruik van XML-bestanden om testcases in te definiëren. Een dergelijke testcase is opgedeeld in drie stadia die getoond worden in afbeelding 2:

1. TestSetup: hierin worden stappen uitgevoerd die nodig zijn voordat de werkelijke test plaatsheeft, bijvoorbeeld het klaarzetten van een bestand dat tijdens de TestExecution-fase moet worden gebruikt.
2. TestExecution: dit stadium bevat stappen voor de werkelijke test.
3. TestCleanup: stadium waarin stappen worden opgenomen die moeten worden uitgevoerd, nadat de werkelijke tests zijn uitgevoerd, bijvoorbeeld het leegmaken van een folder.

In elk stadium kunnen stappen worden opgenomen waarmee een specifieke taak wordt uitgevoerd, bijvoorbeeld het versturen van een bericht via HTTP. Er bestaan stappen om onder meer berichten uit te lezen van bijvoorbeeld een MSMQ-queue en er is een mogelijkheid databases uit te lezen. Bovendien biedt het BizUnit-framework specifieke BizTalk-stappen, die er bijvoorbeeld voor zorgen dat hosts starten of stoppen. Ook kent het framework stappen om berichten te valideren, de zogenaamde validation steps.



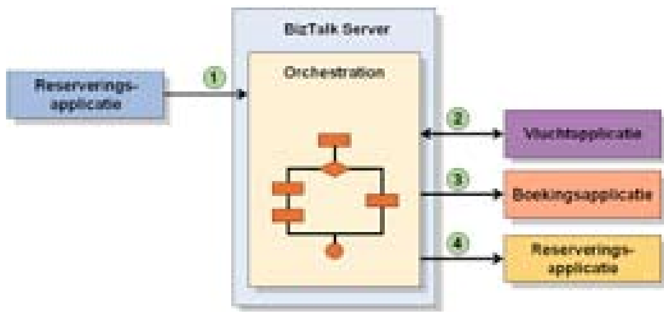
Afbeelding 2. Opbouw testcase

Deze kunnen worden beschouwd als substappen, omdat ze binnen een andere stap genest worden. Ze worden bijvoorbeeld opgenomen in een stap die een XML-bericht van een bestandslocatie uitleest en dit bericht vervolgens valideert tegen XSD. Validation steps kunnen ook de inhoud van een bericht valideren met behulp van XPath-queries. Het BizUnit-framework bevat standaard een groot aantal stappen en als het nodig is, kunnen vrij eenvoudig eigen stappen worden ontwikkeld en toegevoegd. Om de werking van BizUnit verder duidelijk te maken, laat het volgende voorbeeld zien hoe een specifiek scenario kan worden onderworpen aan een functionele test die is opgebouwd met BizUnit. De stappen in de tekst corresponderen met de stappen in afbeelding 3, waarin het scenario wordt getoond. In dit scenario gaat het om een reserveringsapplicatie voor vliegreizen. Wanneer een klant een reis boekt (stap 1), gaat een bericht naar BizTalk. Door de ontvangst van dit bericht start een orchestration. Stap 2 bestaat uit het controleren of de vlucht nog beschikbaar is. Hiervoor wordt een vluchtapplicatie op synchrone wijze geraadpleegd. Wanneer de gekozen vlucht beschikbaar is, wordt deze definitief vastgelegd in een boekingsapplicatie (stap 3) door een bericht naar deze applicatie te verzenden. Ten slotte gaat informatie over de vlucht terug naar de reserveringsapplicatie (stap 4), zodat de klant de status van de boeking kan inzien.

Codevoorbeeld 3 toont een testcase voor het gegeven voorbeeld.

```
[TestClass]
public class FunctionalTests
{
   [TestMethod]
   public void Test_04_Ticketing_Workflow_Default()
   {
      BizUnit bizUnit = new
            BizUnit(@"TestCases\Test_01_ Ticketing_Workflow_Default.xml");
      bizUnit.RunTest();
   }
}
```

Codevoorbeeld 2



Afbeelding 3. Orchestration

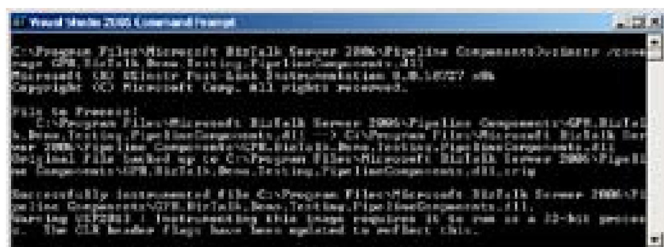
De *HttpPostStep* zorgt dat een bericht asynchroon via het http-protocol wordt verzonden. Vervolgens is een vertraging ingebouwd met een *DelayStep*: in deze tijd moet de Vluchtapplicatie, die op synchrone wijze communiceert, een antwoord hebben gegeven. Daarna raadpleegt de *DBQueryStep* de boekingsapplicatie. De stap verifieert of de onderliggende database een record met de verwachte informatie bevat. De laatste stap in deze testcase bestaat uit het verifiëren van een bericht dat naar de reserveringsapplicatie is verzonden. De *FileValidateStep* leest het bericht uit dat in een bepaalde map staat. De *XmlValidationStep* valideert het bericht tegen de bijbehorende XSD en vergelijkt de inhoud met de verwachte waarden.

Coverage

Als de functionele tests van goede kwaliteit zijn, raken ze alle code in de BizTalk-applicatie. Dat betekent niet alleen dat alle paden in custom code moeten worden geraakt, maar ook dat alle paden in alle orchestrations moeten worden uitgevoerd in de tests.
De coverage van custom code die gebruikt wordt in een BizTalk-applicatie kan goed worden gemeten met de Visual Studio Code Coverage Tool. Deze tool kan via de console worden aangeroepen en zal een coverage-bestand opleveren. Dit bestand laat, als het in Visual Studio wordt geladen, op duidelijke wijze zien hoeveel code is geraakt. Het onderstaande proces beschrijft hoe deze tooling wordt ingezet om een custom pipeline-component te testen.

1. Het commando *VSInstr.exe –coverage* voegt code aan een assembly toe die wordt gebruikt om te bepalen welke statements worden uitgevoerd. Er wordt een back-up gemaakt van de originele assembly (afbeelding 4).
2. Het commando *start VSPerfmon –coverage* maakt een coverage-bestand aan; afbeelding 5.
3. In VSPerf Console Profile Monitor, die geopend is door het vorige commando, is te zien dat de assembly die in stap 1 is beschreven, wordt geraakt op het moment dat de test draait. De monitor geeft aan welk proces de assembly aanroept, in dit geval is dat het BizTalk-proces; afbeelding 6.
4. Het coverage-bestand uit stap 2, dat is geopend in Visual Studio, geeft duidelijk aan welke delen van de code wel en niet zijn geraakt; afbeelding 7. In dit geval is geen exceptie opgetreden, dus wordt dit deel van de code rood gearceerd.

Door gebruik te maken van de bovenstaande methode kan de coverage van custom code worden bepaald, maar niet van orchestrations. Om de coverage van orchestrations te bepalen, wordt gebruikgemaakt van de tool Orchestration Profiler. Deze tool genereert een chm-rapport waarin grafisch wordt getoond welke

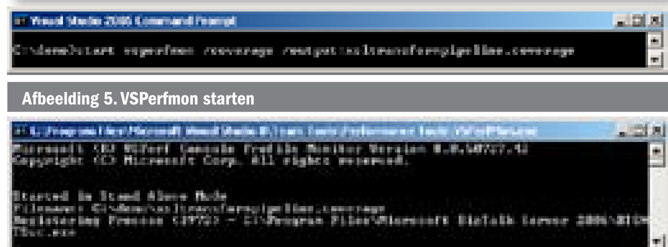


Afbeelding 4. VSInstr.exe uitvoeren

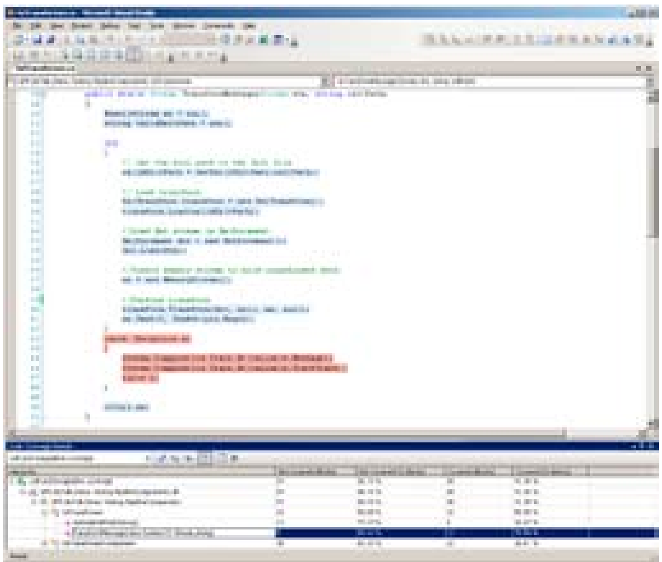
```
<TestCase testName="Test_ReservationProcedure">
  <TestSetup>
  </TestSetup>
  <TestExecution>
    <TestStep assemblyPath="" typeName="Microsoft.Services.
        BizTalkApplicationFramework.BizUnit.HttpPostStep">
      <SourcePath>>.\TestMessages\Flight.xml</SourcePath>
      <DestinationUrl>http://biztalk_server:15000/BTSHTTPReceive.dll
      </DestinationUrl>
      <RequestTimeout>10000</RequestTimeout>
    </TestStep>
    <TestStep assemblyPath="" typeName="Microsoft.Services.
        BizTalkApplicationFramework.BizUnit.DelayStep">
      <Delay>10000</Delay>
    </TestStep>
    <TestStep assemblyPath="" typeName="Microsoft.
        Services.BizTalkApplicationFramework.BizUnit.DBQueryStep">
      <DelayBeforeCheck>1</DelayBeforeCheck>
      <ConnectionString>Persist Security Info=False;Integrated
Security=SSPI;database=Bookings;server=DBServer_01;Connect Timeout=30
      </ConnectionString>
      <SQLQuery>
        <RawSQLQuery>SELECT * FROM Booking WHERE flightID =  'kl-756-8642'
        </RawSQLQuery>
      </SQLQuery>
      <Rows>
        <Columns>
          <Passenger>Siegers</Passenger>
          <Status>Confirmed</Status>
        </Columns>
      </Rows>
    </TestStep>
    <TestStep assemblyPath="" typeName="Microsoft.Services.
      BizTalkApplicationFramework.BizUnit.FileValidateStep">
      <Timeout>3000</Timeout>
      <Directory>.\ReserveringsApp\</Directory>
      <SearchPattern>Reservation_*.xml</SearchPattern>
      <DeleteFile>true</DeleteFile>
      <ValidationStep assemblyPath="" typeName="Microsoft.Services.
      BizTalkApplicationFramework.BizUnit.XmlValidationStep">
        <XmlSchemaPath>.\XSD\Reservation\Reservation.xsd
        </XmlSchemaPath>
        <XmlSchemaNameSpace>
          http://schemas.reservation.nl/reservation/2004/09
        </XmlSchemaNameSpace>
        <XPathList>
          <XPathValidation query="/*[local-name()='Reservation'
and namespace-uri()='http://schemas.reservation.nl/
reservation/2004/09']/*[local-name()='Passenger' and namespace-
uri()='']">Siegers</XPathValidation>
        </XPathList>
      </ValidationStep>
    </TestStep>
  </TestExecution>
  <TestCleanup>
  </TestCleanup>
</TestCase>
```

Codevoorbeeld 3. Testcase orchestration

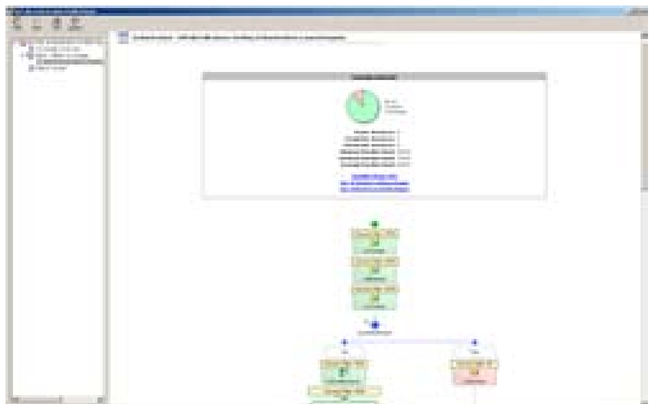
Afbeelding 5. VSPerfmon starten



Afbeelding 6. Het proces BTSNTSvc.exe roept de pipeline aan



Afbeelding 7. Coverage-bestand in Visual Studio



Afbeelding 8. Orchestration profiler

shapes in een orchestration zijn geraakt tijdens de uitvoering van een test. Een dergelijk rapport wordt getoond in afbeelding 8.

Onmisbaar voor succes

BizTalk-applicaties zijn lastig om te testen, omdat meestal verschillende systemen op verschillende locaties bij zijn betrokken. Meestal schiet het testen in ontwikkelprojecten er bij in, omdat een testfase achteraan in een project wordt gepositioneerd en vaak moet lijden onder de uitloop van voorafgaande fasen. Om beide redenen worden tests vaak niet of nauwelijks uitgevoerd, wat een negatief effect heeft op de kwaliteit van de BizTalk-applicatie. Er bestaat geen goede reden om het testen van software zomaar over te slaan. De in dit artikel beschreven tooling biedt goede ondersteuning voor het testen en is in onze ogen onmisbaar om een BizTalk-applicatie tot een succes te maken

.

Dick Dijkstra en Christian Siegers zijn BizTalk-specialisten. Voor vragen en opmerkingen kunnen Dick en Christian bereikt worden op respectievelijk dick.dijkstra@getronics.com of c.siegers@stater.com

Referenties

Keven Smith's Weblog: http://blogs.msdn.com/kevinsmi/

BizUnit – Framework for Automated Testing: http://www.codeplex.com/bizunit/

Orchestration Profiler: http://www.codeplex.com/BiztalkOrcProfiler

Blog Dick Dijkstra: http://www.dickdijkstra.com/

Blog Christian Siegers: http://blog.christiansiegers.nl/