Bart Waeterschoot is .NET consultant bij Compuware

FxCop: Join the code police

SCHRIJF ZELF EEN RULE OM AUTOMATISCH CODE TE ANALYSEREN IN VISUAL STUDIO

Er zijn verschillende technieken beschikbaar om codekwaliteit te verhogen. Je kunt onder andere documenten met coderichtlijnen aanmaken, codereviews inplannen en in pair programming voorzien. Maar omdat deze technieken veel extra tijd in beslag nemen tijdens development worden ze meestal vermeden. De techniek die het minste tijd in beslag neemt, doordat ze geautomatiseerd is, is statische codeanalyse. In dit artikel gaat de auteur nader in op het analyseren van broncode waarbij hij zich toespitst op FxCop.

Wat is statische codeanalyse? Statische codeanalyse probeert fouten en mogelijke problemen op te sporen die pas zichtbaar worden tijdens het draaien van de applicatie. De engine die deze analyse uitvoert, heeft hiervoor enkel de code nodig. Dit is dus eigenlijk een aanvulling op de fouten en waarschuwingen die je reeds krijgt van de compiler zelf. Voor het analyseren van broncode zijn verschillende tools verkrijgbaar zoals FxCop, ReSharper en DevPartner. Deze tools beschikken allemaal over een standaard set van regels waaraan code moet voldoen en bij sommige tools (waaronder FxCop) kun je deze set uitbreiden met eigen regels.

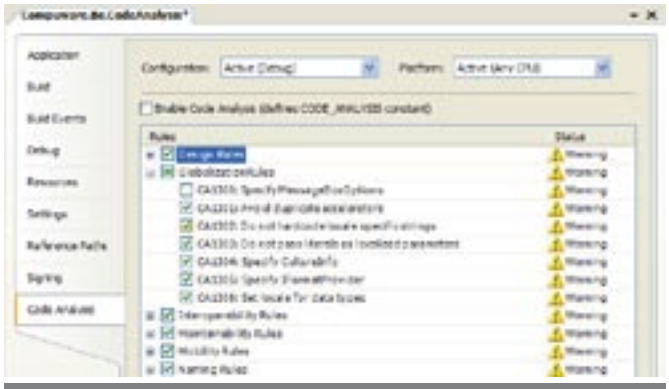
FxCop

FxCop is de populairste tool voor statische codeanalyse in .NET. FxCop is beschikbaar in een GUI- en consoleversie. FxCop bevat standaard zo'n tweehonderd rules die onderverdeeld zijn in elf categorieën. Er zijn bijvoorbeeld aparte categorieën voor beveiliging, performance en naamgeving. Deze tool wordt ondertussen meegeleverd in Visual Studio Team System onder de naam Managed Code Analysis Tool. Deze is volledig geïntegreerd in de IDE. Je kunt jouw project in Team System laten analyseren door de FxCop-engine. Rules kun je individueel aan- en afvinken per project in de 'Code Analysis'-tab in de 'Project Settings'; zie afbeelding 1. Je kunt op deze tab ook per regel instellen of het niet volgen tot een waarschuwing of error leidt. Deze code-instellingen kunnen gekoppeld worden aan een buildconfiguratie. Zo kun je bijvoorbeeld een strengere set van regels hanteren voor een release-build dan voor een debug-build. Na het aanzetten en instellen van de codeanalyse kun je in het menu Build – Run Code Analysis aanklikken om de analyse te starten of gewoon compileren. Na de analyse krijg je een lijst van waarschuwingen in de output-window van Visual Studio als er bepaalde regels niet gevolgd worden; zie afbeelding 2. Deze waarschuwingen bevatten meestal ook een oplossing om wel te voldoen aan de regel. Dit zorgt er voor dat je de richtlijnen niet zomaar moet opvolgen. Op deze manier kun je ook geen fouten missen.

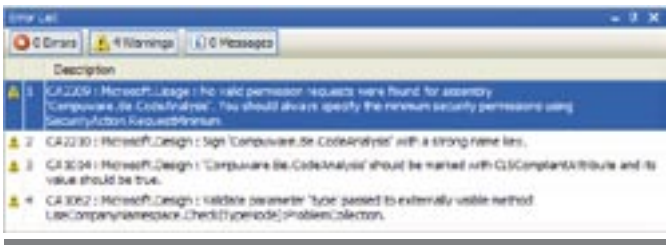
De output van de analyse wordt ook opgeslagen in de Debug- of Release-map van jouw project. Het bestand noemt <ProjectNaam>.CodeAnalysisLog.xml. Deze xml kun je eventueel gebruiken in combinatie met een XSL-bestand om een overzichtelijk rapport te genereren. Sommige regels kunnen een waarschuwing geven terwijl het niet terecht is. Als je zeker bent dat de regel niet van toepassing is op jouw stuk code, dan kun je rechtsklikken op de waarschuwing en voor 'Suppress Message(s)' kiezen. Er wordt dan een SuppressMessage-attribuut toegevoegd op het stuk code dat gekoppeld is aan de waarschuwing. Dit stuk code wordt dan voor de desbetreffende regel tijdens volgende validaties genegeerd. Als je het probleem zeker moet oplossen, dan kun je rechtsklikken op de waarschuwing en zo een work-item aanmaken in Team System. Dit kan alleen maar als jouw Visual Studio gekoppeld is aan een Team Foundation Server.

Wat kunnen we analyseren?

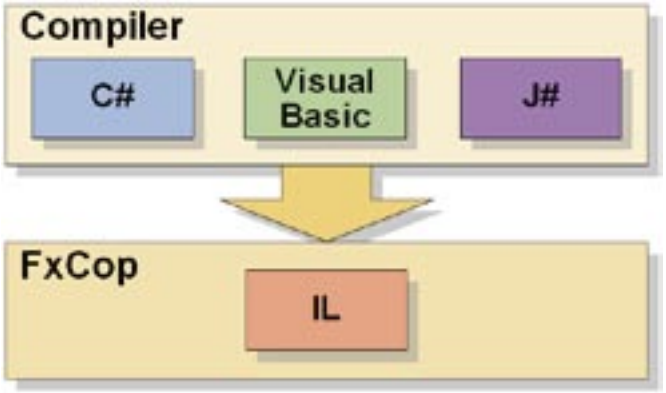
Belangrijk om weten is dat FxCop enkel IL-code kan analyseren en geen broncode in C# of Visual Basic; zie afbeelding 3. Je kunt vanuit het standpunt van FxCop dus geen codecommentaar nakijken of de lay-out van de originele broncode opvragen. De IL-code wordt bekeken via introspection. Introspection kun je vergelijken met reflection. Introspection kan namelijk ook info ophalen over types in een assembly, maar doet dit sneller dan reflection en vergt geen lock op het assembly-bestand. FxCop bouwt in geheugen ook



Afbeelding 1. Rules kun je individueel aan- en afvinken per project



Afbeelding 2. Lijst van waarschuwingen in de output-window van Visual Studio



Afbeelding 3. FxCop analyseert de IL-code

```
using System;
using Microsoft.FxCop.Sdk.Introspection;

namespace Compuware.Be.CodeAnalysis
{
  public abstract class BaseRule : BaseIntrospectionRule
  {
    protected BaseRule(string name)
      : base(name, "Compuware.Be.CodeAnalysis.Rules",
      typeof(BaseRule).Assembly)
    {
    }
  }
}
```

Codevoorbeeld 1.

een callgraph-analyse op. Deze analyse bevat de relaties tussen de verschillende types en methodes. We kunnen de callgraph bijvoorbeeld gebruiken om niet gebruikte private methods en types te ontdekken. De standaard regels zijn een goed vertrekpunt. Maar er zullen steeds meer specifieke eisen zijn per bedrijf of project. We kunnen dankzij de FxCop SDK eigen regels en categorieën toevoegen. Als eenvoudig voorbeeld gaan we een rule creëren die nakijkt of alle types in een assembly binnen een namespace zitten die start met de bedrijfsnaam. Het concept namespaces bestaat niet in IL-code dus we moeten de volledige naam van de types analyseren. Concreet betekent dit dat alle types en methods met "Compuware.Be." moeten starten.

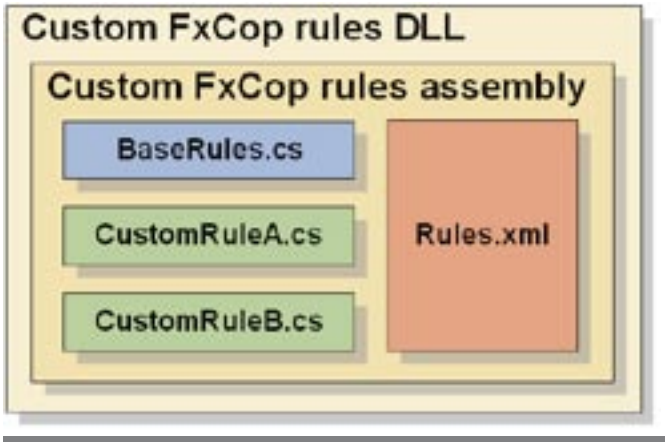
De onderdelen van een regel

Een FxCop-assembly bestaat uit twee delen: de implementatie in code en een resource xml-bestand. Het xml-bestand bevat de omschrijving van alle rules binnen de assembly en bevat een verwijzing naar de klasse die gekoppeld is met een bepaalde regel; zie afbeelding 4. We gaan starten met de implementatie in C#.

Implementatie

Om te starten creëren we een nieuw class library-project. Vervolgens voorzien we een referentie naar de FxCop SDK (FxCopSdk.dll) en de Common Compiler Infrastructure (Microsoft.Cci.dll) assembly. Deze bestanden vind je terug in de Visual Studio 2005-installatiefolder onder "Team Tools\Static Analysis Tools\FxCop". De FxCop SDK-dll bevat klassen die verband houden met de standaard FxCop-concepten (bvb rules). Alle code die gerelateerd is met introspection zelf zit in de CCI-dll.
Creëer nu een BaseRule-klasse binnen het project die overerft van de BaseIntrospectionRule-klasse; zie codevoorbeeld 1. We voegen deze BaseRule-klasse toe, omdat elke regel in dit project op dezelfde manier wordt geïnitialiseerd. De constructor zorgt er voor dat de regel correct gekoppeld wordt met de xml-definitie die we straks toevoegen. De BaseIntrospectionRule implementeert de IIntrospectionRule-interface die de vaste 'Check'-methodes bevat. Er zijn check-overloads voor assemblies, types, enzovoort; zie afbeelding 5. De code analysis-engine zorgt er voor dat voor elk onderdeel van de te analyseren assembly de correcte Check-methode wordt opgeroepen.

We gaan de TypeNode-overload gebruiken voor onze regel. We voegen nu de echte UseCompanyNamespace-regel toe die overerft van de BaseRule-klasse; zie codevoorbeeld 2. We implementeren hier de type 'Check'-methode om zo alle types binnen de te analyseren assembly te bekijken. Per type kunnen we dan nakijken of het voldoet aan onze voorwaarde. Als het type niet start met de company-naam, dan voegen we een item toe in de problems-collectie. De functie GetResolution leest de resolution-omschrijving uit de bijhorende xml uit en voegt eventuele parameters in waar er placeholders voorzien zijn. Dit is te vergelijken met het gebruik van string.Format. Hier geven we als parameter de volledige typenaam mee om de foutboodschap te verduidelijken. Deze foutboodschap verschijnt na de analyse in de lijst met waarschuwingen. Als een gebruiker dubbelklikt op deze waarschuwing, dan zal het bestand waarin de fout is gevonden, automatisch geopend worden met de cursor op de declaratie van het type met de foute namespace. Ten slotte creëren we een nieuw xml-bestand genaamd Rules.xml; zie codevoorbeeld 3. De rules-xml-node kan meerdere rule-nodes bevatten. In de rule-definitie vinden we de typenaam terug van de klasse die we hebben geschreven. De category kun je invullen om jouw regels duidelijker te groeperen in de foutmeldingen. De CheckId bevat de unieke code van de regel. Je kunt hier best de afkorting van jouw firmanaam gebruiken met een oplopende nummering. Als verdere eigenschappen van een rule kun je hier ook een naam en omschrijving meegeven. De CheckId, category en FriendlyName worden gebruikt voor het overzicht van de set van regels; zie afbeelding 1.
De resolution-tag bevat een voorstel om het probleem op te lossen. Je kunt hier de standaard string format placeholders voorzien om nog zaken aan te vullen die je pas te weten komt tijdens de analyse. De parameters die je meegeeft met de GetResolution-functie zullen hier dan ingevuld worden. Dit is trouwens de boodschap die getoond wordt bij de foutmeldingen in Visual Studio. Het is dus belangrijk om deze boodschap zo duidelijk en volledig mogelijk te maken. Messagelevel-certainty geeft aan hoe zeker we er van zijn dat de regel van toepassing is. Het kan namelijk zijn dat een bepaalde regel om een geldige reden niet gevolgd wordt. Deze waarschijnlijkheid wordt aangegeven met een getal van 0 tot 99. Hoe hoger het getal, hoe groter de kans is dat de waarschuwing terecht is. Een waarschijnlijkheid van 75 betekent bijvoorbeeld dat gemiddeld drie op de vier waarschuwingen voor die regel terecht zijn. Bijna alle standaard regels hebben de maximale waarschijnlijkheid. Bij onze regel gaan we ook 99 instellen, omdat er geen randgevallen zijn waar onze regel niet van toepassing is. De messagelevel zelf wordt gebruikt om aan de duiden hoe erg het niet volgen van de regel is. Dit is een standaard enumeratie met de



Afbeelding 4. De onderdelen van een FxCop-assembly

volgende waarden: Information, Warning, CriticalWarning, Error of CriticalError. Je kunt ook nog optionele informatie invullen. De url-tag bevat een link waar je meer informatie kunt vinden over de regel. Email en Owner bevatten de gegevens van een contactpersoon waar je extra info kunt verkrijgen.
Opgelet: Dit bestand moet gecompileerd worden als embedded resource! Dit doe je door het xml-bestand te selecteren in de solution explorer en de properties te openen. Daar kun je dan bij Build Action de optie Embedded Resource selecteren. Als je nu compileert, dan zal de xml als metadata toegevoegd worden aan de assembly.

De rule gebruiken

Als we het project gecompileerd hebben, kunnen we deze nieuwe assembly kopiëren naar de vaste map voor 'Rules' in Visual Studio, namelijk 'C:\Program Files\Microsoft Visual Studio 8\Team Tools\Static Analysis Tools\FxCop\Rules'. Je moet een nieuwe instantie van Visual Studio starten om de nieuwe regel actief te maken. Je kunt nu bij de code analysis-settings zien dat er een nieuwe categorie is bijgekomen met één regel. Visual Studio haalt deze info uit het definitiebestand. Bij het starten van een nieuwe codeanalyse zul je merken dat de nieuwe regel zijn werk doet. Het kan zijn dat je al onterechte waarschuwingen krijgt in het testproject. In ons voorbeeld hebben we namelijk nog geen rekening gehouden met automatisch gegenereerde code. Gegenereerde code kan niet steeds voldoen aan onze eis om te starten met onze company-naam. Dus moeten we de checkfunctie herschrijven om gegenereerde code niet te analyseren. Dit kunnen we door van elk type de attributen uit te lezen en te zoeken naar het GeneratedCodeAttribute. Dit is vooral handig om asp.net- en dataset-gegenereerde code te omzeilen.

Unit-testing

Je kunt gemakkelijk unit-testen schrijven voor FxCop-rules door de checkmethodes op te roepen. Per geldige checkmethode kun je de problemcollection returnvalue evalueren om te bepalen of de unit-test geslaagd is. De geldige en ongeldige types waarop je wil testen, kun je als klassen toevoegen in een map in jouw testproject.

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Cci;
using Microsoft.FxCop.Sdk;
using Microsoft.FxCop.Sdk.Introspection;

namespace Compuware.Be.CodeAnalysis
{
  public class UseCompanyNamespace : BaseRule
  {
    private const string CompanyNamespace = "Compuware.Be.";

    public UseCompanyNamespace()
      : base("UseCompanyNamespace")
    {
    }

    public override ProblemCollection Check(TypeNode type)
    {
      if (!type.FullName.StartsWith(CompanyNamespace,
        StringComparison.InvariantCultureIgnoreCase))
      {
        Problems.Add(new Problem(GetResolution(type.FullName)));
      }

      return Problems;
    }
  }
}
```

Codevoorbeeld 2.

```
<?xml version="1.0" encoding="utf-8" ?>
<Rules FriendlyName="Custom Code Analysis Rules">
  <Rule TypeName="UseCompanyNamespace" Category="Custom Rules"
    CheckId="CPWR0001">
    <Name>Use company namespace</Name>
    <Description>
      All custom code should reside in our company namespace.
    </Description>
    <Resolution>
      Make sure that the '{0}' namespace starts with 'Compuware.Be.'.
    </Resolution>
    <MessageLevel Certainty="99">Warning</MessageLevel>
    <FixCategories>NonBreaking</FixCategories>
    <Url>/Custom/UseCompanyNamespace.html</Url>
    <Email>bart.waeterschoot@compuware.com</Email>
    <Owner>Bart Waeterschoot</Owner>
  </Rule>
</Rules>
```

Codevoorbeeld 3.

Build-integratie

Vanuit MsBuild kun je de commandline-versie van FxCop oproepen, zijnde FxCopCmd.exe. Hiermee kun je het hele analyseproces automatiseren en de FxCop-output als rapport bij jouw versie voegen. Het gebruik van FxCop in het buildproces of als checkin-policy garandeert consistente code over het totale project. Dit is dus zeker de moeite waard om eens verder te bekijken.

Reflector is jouw beste vriend

Om vlot van start te gaan met nieuwe fxcop-rules, kun je het beste de bestaande regels in de team system 'Rules'-map bekijken in reflector. Deze bevatten al de meeste codevoorbeelden die nodig zijn om jouw eigen regels te ontwikkelen. Reflector en ILDASM zijn ook onmisbaar om patronen te vinden in te analyseren assemblies.

Veel waar voor jouw geld

Het is zoals je merkt zeer eenvoudig om zelf FxCop-rules te schrijven. De uitdaging is om ze waterdicht te maken. In ons voorbeeld hebben we bijvoorbeeld nog geen rekening gehouden met gegenereerde code. Het kan zijn dat er klassen in ons te analyseren project zitten, waarbij we geen invloed hebben op de code. Om de regel te verbeteren, moeten we dus de checkfunctie al uitbreiden door de GeneratedCodeAttribute uit te lezen en de gegenereerde types te negeren. Het schrijven van deze regels blijft hoe dan ook een eenmalige inspanning die daarna bij elk project hergebruikt kan worden. Het is dus een beperkte investering die veel kan opleveren.

Bart Waeterschoot is werkzaam als .NET consultant bij Compuware Belgium (www.compuware.be). Momenteel houdt hij zich graag bezig met software factories, de CLR verkennen en nieuwe technologieën. Je kunt Bart via email bereiken op: bart.waeterschoot@compuware.com

Referenties:

The FxCop-teamblog: http://blogs.msdn.com/fxcop/
MSDN article: http://msdn.microsoft.com/msdnmag/issues/04/06/bugslayer/
.NET Reflector: www.aisto.com/roeder/dotnet/