Robert van der Kleij is technisch directeur bij Qurius ETX.

Productiviteitsverbeteringen in nieuwe Visual Studio

TIEN MANIEREN OM NOG SNELLER CODE TE SCHRIJVEN IN VISUAL STUDIO 2005

Bij de voorgaande versies van de Visual Studio IDE geeft Microsoft al blijk van aandacht voor productiviteitsverbeteringen die ons werk gemakkelijker en vooral leuker maken. IntelliSense is hier een mooi voorbeeld van. In Visual Studio 2005 worden deze inspanningen nog een keer ruimschoots overtroffen.

In deze release zijn namelijk verschillende handige features opgenomen die getuigen van een duidelijke focus op gebruiksgemak en werkplezier voor de ontwikkelaar, zowel op individuele basis als in teamverband. Kortom, voor ieder wat wils: de tien manieren om nog sneller code te schrijven in de IDE Visual Studio 2005.

1. Expansions

Bij het schrijven van code zijn er altijd terugkerende coderegels die de developer dan iedere keer opnieuw moet typen. De using- en import-statements boven elk codebestand zijn hiervan een duidelijk voorbeeld. Visual Studio 2005 ondersteunt expansions waarmee herhalende stukjes code snel kunnen worden toegevoegd. Je gebruikt deze expansions door een afkorting in te tikken en met Tab de tekst te laten expanderen. Door bijvoorbeeld prop en Tab in te voeren, verschijnt er al een standaardtekst voor een property, inclusief private variabele declaratie. Zo uitgelegd lijkt het misschien veel op de code snippets die nu al in Visual Studio zitten en hetzelfde zou bereikt kunnen worden met ouderwets knippen en plakken. Expansions in Visual Studio 2005 zijn echter slimmer. Zij vervangen een keyword niet alleen door een stuk code, maar bevatten ook velden die kunnen worden vervangen met zelf te bepalen waarden. Met Tab kun je snel van het ene naar het andere gele invoerveld springen en daar bijvoorbeeld het type van de variabele en de naam van de property invullen; zie afbeelding 1. Als het veld vaker voorkomt, worden alle andere verwijzingen automatisch vervangen. Door deze velden te gebruiken kun je expansions in veel meer situaties nuttig inzetten dan wanneer je gebruik maakt van knippen en plakken. Visual Studio 2005 .NET ondersteunt 50 standaard expansions, van properties tot classes, maar je kunt ook eigen expansions toevoegen of bestaande expansions aanpassen. Alle expansions zijn gedefinieerd in xml-bestanden in de expansions-directory en zijn in een paar minuten naar je hand te zetten. Elk xml-bestand bevat een naam, een shortcut, optioneel de definitie van een aantal velden en een data tag waarin de daadwerkelijke tekst is opgenomen met extra keywords en veldverwijzigingen. Er is vanuit een gebruikersperspectief goed nagedacht om expansions zo optimaal mogelijk in te kunnen zetten. Zo kun je bijvoorbeeld met het keyword end instellen waar de cursor moet eindigen nadat alle velden zijn ingevuld, zodat je onmiddellijk kunt doorgaan met typen. Ook zijn velden via standaardfuncties in te vullen, zoals de functie ClassName(), die bij de constructor expansion gebruikt wordt om de naam van de constructor al in te vullen. Zie codevoorbeeld 1 met een constructor expansion.

```
private int myVar;

public int MyProperty
{
    get { return myVar; }
    set { myVar = value; }
}
```

Afbeelding 1. Gebruik van expansions

2. Automatisch formatteren van code

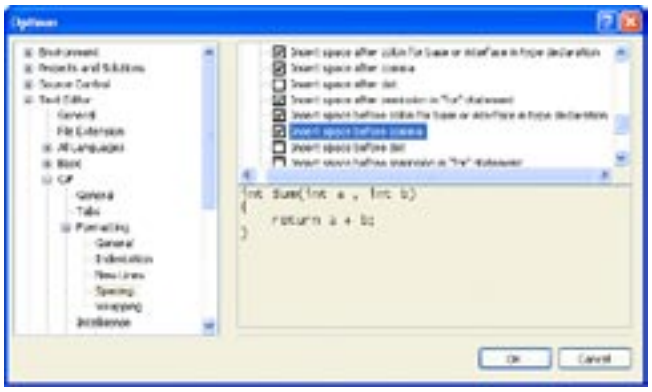
Formattering van code is sterk afhankelijk van de persoonlijke voorkeur van een developer. De één zweert bij K&R bracing, waarbij

```
<CodeSnippet Format="1.0.0">
  <Header>
    <Title>a common constructor pattern</Title>
    <Shortcut>ctor</Shortcut>
    <Description>Expansion for constructor</Description>
    <category>Expansion</category>
  </Header>
  <Snippet>
  <Declarations>
    <Literal>
      <ID>type</ID>
      <Default>int</Default>
    </Literal>
    <Literal>
      <ID>name</ID>
      <Default>field</Default>
    </Literal>
    <Literal default="true" Editable="false">
      <ID>classname</ID>
      <ToolTip>Class name</ToolTip>
      <Function>ClassName()</Function>
      <Default>ClassNamePlaceholder</Default>
    </Literal>
  </Declarations>
  <Code Language="csharp" Format="CData">

<![CDATA[
public $classname$ ($type$ $name$)
{
  this._$name$ = $name$;
  $end$
}

private readonly $type$ _$name$;
]]>
</Code>
  </Snippet>
</CodeSnippet>
```

Codevoorbeeld 1.
Constructor expansion

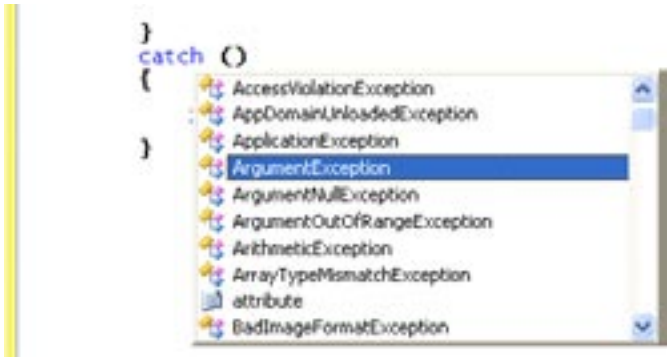


Afbeelding 2. Eigen keuze wat betreft formattering van code

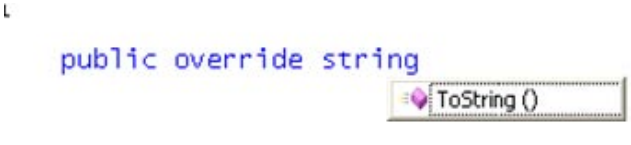
wordt geopend met een accolade aan het einde van de regel en afgesloten met een accolade aan het begin van een regel. Een ander vindt dit niet leesbaar en geeft de voorkeur aan netjes onder elkaar uitgelijnde accolades. Deze strijdbijl kunnen we nu begraven: in Visual Studio 2005 zijn de persoonlijke voorkeuren tot in detail instelbaar. Op dit moment is het al mogelijk om in Visual Studio 2003 globaal de voorkeur voor het formatteren van code in te stellen, maar dat is niet te vergelijken met de tientallen opties die Visual Studio 2005 bevat. In de ontwikkelfase van deze IDE is een grondige analyse gedaan van alle verschillende coding-standaarden en vervolgens zijn al deze variëteiten en alternatieven vertaald in de nieuwe release.
Wil je een spatie na een komma in een parameterlijst, dan is dit in te stellen. Accolades op een nieuwe regel beginnen, inspringen bij elke accolade of een spatie na een binaire operatie: het is allemaal mogelijk. Zoals afbeelding 2 toont, heeft het merendeel van de opties te maken met het gebruik van spaties en witregels.
Het formatteren vindt automatisch plaats tijdens het typen, maar is ook via het menu op een geselecteerd stuk code uit te voeren. Het komt vaak voor dat een stuk voorbeeldcode op het internet of in een helpfile niet voldoet aan iemands persoonlijke standaard. Bij het invoegen van deze code past Visual Studio 2005 automatisch de formattering aan, zodat deze meteen voldoet aan iemands bedrijfsstandaarden. Deze instellingen kunnen net als alle andere gebruikersinstellingen worden geëxporteerd en in iedere Visual Studio 2005-omgeving snel worden ingelezen. Het hele team kan zo zonder veel moeite dezelfde instellingen inlezen en gebruiken.

3. IntelliSense en completion list

Inmiddels is iedereen gewend geraakt aan IntelliSense en vertrouwt men volledig op de completion lists met variabelen en functies die te pas (en voor sommigen te onpas) verschijnen in Visual Studio. Hoe groot het voordeel van deze lijstjes ook is,



Afbeelding 3. Typen afgeleid van System.Exception



Afbeelding 5. Alleen nog keuze voor de methode ToString

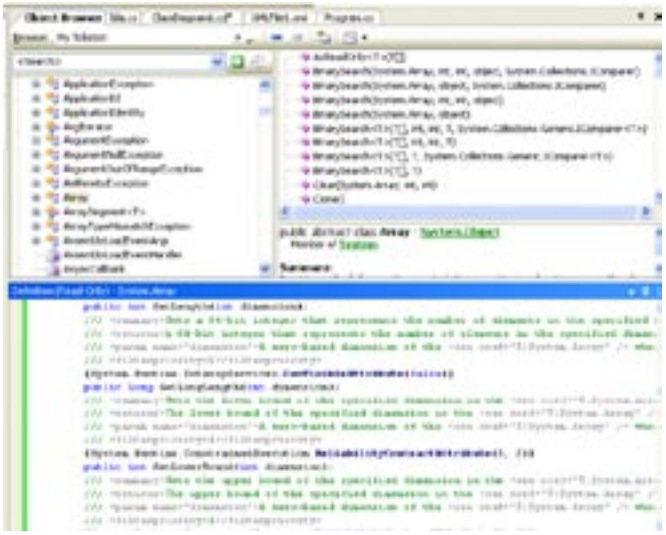
Afbeelding 4. Alleen nog interfaces in de completion list

ze bevatten altijd elementen die je niet gebruikt. Om dit lijstje effectiever te maken, worden in Visual Studio 2005 de selectiemogelijkheden gefilterd, afhankelijk van de context. Na een catch block toont Visual Studio 2005 bijvoorbeeld alleen nog typen die afgeleid zijn van Sytem.Exception; zie afbeelding 3. Deze filterfunctionaliteit gaat diep tot in de details. Als bijvoorbeeld een class afgeleid is van een base-class, dan worden bij het uitbreiden van de overervinglijst alleen nog maar interfaces getoond in de completion list; zie afbeelding 4. In Bèta 1 toont de IDE alleen bij de eerste overerving in de lijst alle classes en bij opeenvolgende alleen de interfaces, ongeacht of er verderop in de lijst al een base class genoemd staat.
Er is verder een aantal subtiele wijzigingen in de IntelliSense doorgevoerd, die ervoor zorgen dat je er veel vaker gebruik van kunt maken. Bij het toevoegen van een 'override methode' verdween in 2003 de completion list op het moment dat je het return type van de methode opgaf. In Visual Studio 2005 blijft de completion list niet alleen staan, maar deze filtert ook de methodes op basis van het return type dat je intikt. In afbeelding 5 is door het invoeren van het return type string alleen nog de methode ToString te kiezen.

4. Code-navigation

Waaraan besteden ontwikkelaars hun tijd als ze aan het programmeren zijn? Het antwoord op deze vraag is verrassend als je het onderzoek van Microsoft mag geloven. Op de Professional Developer Conference 2003 besprak Scott Wiltamuth, Microsoft C# Product Unit Manager, de uitkomsten van dit onderzoek. Door middel van interviews, maar ook door daadwerkelijk naast een ontwikkelaar te gaan zitten, probeerde men inzicht te verkrijgen in diens activiteiten. Wat bleek: lang niet

Afbeelding 6. Implementatie Swap-methode



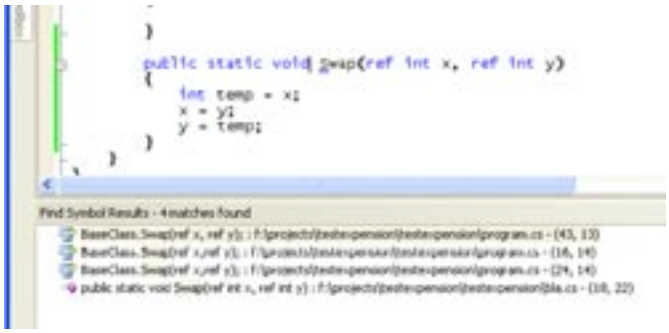
Afbeelding 7. Code Definition View Window

alle tijd wordt besteed aan het schrijven of debuggen van code. Een ontwikkelaar is significant veel tijd kwijt met het begrijpen van deze code. Hij browst en navigeert door de code heen om het overzicht van een applicatie te verkrijgen. Daarbij springt hij veelvuldig naar de definitie van een methode (drill down) en bekijkt hij welke classes een specifieke methode aanroepen. Het vervelende aan dit navigeren is, dat je continu moet wisselen tussen verschillende codevensters. Met het nieuwe Code Definition View Window hoeft dit niet meer. Dit is een klein venster onder in het scherm dat de implementatie laat zien van de methode waarop je op dat moment staat. Dit venster is read-only, maar geeft precies genoeg informatie, zodat het niet nodig is om naar de methode zelf te navigeren. In afbeelding 6 wordt de implementatie van de Swap-methode getoond die in de normale code-file wordt aangeroepen.

5. Browsen van componenten: Object Browser

Een functionaliteit die ook in lijn is met het sneller en beter begrijpen van code, is de wijziging aan de object browser. Deze browser werkt samen met het Code Definition View Window. Dit venster toont in de object browser de definities van de classes in source code, naast het standaardlijstje van methodes en fields; zie afbeelding 7.
Natuurlijk is de daadwerkelijke broncode niet meegeleverd voor framework classes, maar de complete definitie met al het bijbehorende commentaar wordt wel als broncode getoond. De gedachte hierachter is, dat je zo als developer veel beter in staat bent om code te begrijpen en te doorzien dan wanneer de omschrijving in een helpfile is opgenomen. Een tweede voordeel is dat definities van bijvoorbeeld classes eenvoudig te kopiëren zijn als je een class maakt met vergelijkbare functionaliteit. Het Code Definition View Window kun je overigens ook gebruiken in combinatie met

Afbeelding 9. Meegeven van reference parameters aan een statische functie



Afbeelding 8. Compacte lijst met juiste zoekresultaten

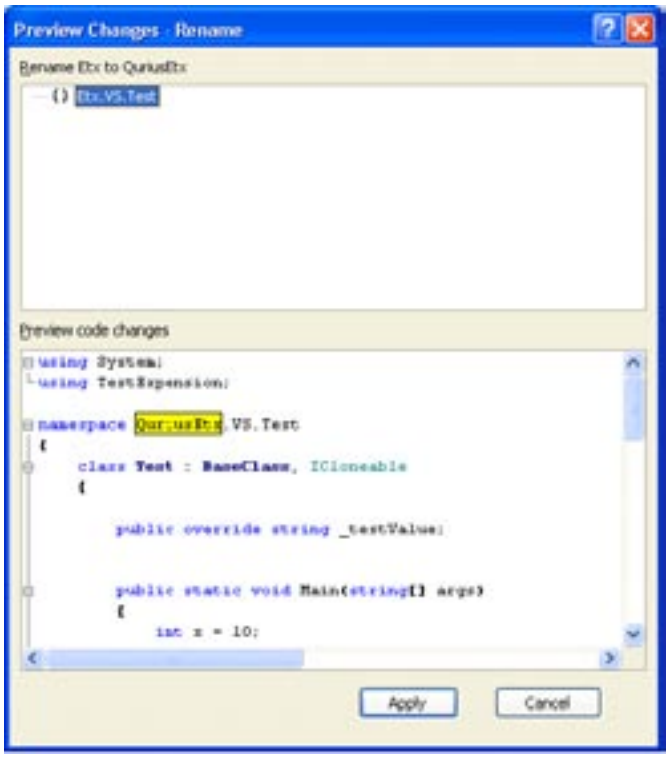
een standaard codevenster. Op het moment dat de cursor op een framework-methode staat, wordt in het Code Definition View Window ook een broncode-definitie gegeven.

6. Vinden van referenties

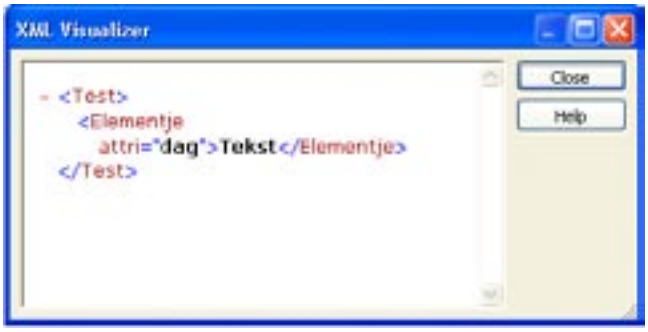
"Waar wordt deze methode nog meer gebruikt?" Een vraag die iedere developer veelvuldig zal stellen. Hoe en waar een methode gebruikt wordt, zegt veel over de structuur van de code en bepaalt hoe gemakkelijk de methode aangepast kan worden. Iedereen kent het gevoel dat, na het gebruik van 'Find in Files' er een lijst van 300 items verschijnt, waar maar tien juiste referenties in zitten. Dit behoort tot het verleden met de nieuwe 'Find All References'-functionaliteit. Deze geeft een overzicht van alleen die referenties die verwijzen naar de betreffende methode. Visual Studio 2005 herkent daarbij het verschil met methode-namen uit andere classes en namespaces. Visual Studio 2003 bevat al de functionaliteit om van de ene naar de andere referentie te springen, maar door een compacte lijst zonder verkeerde zoekresultaten kan veel sneller worden genavigeerd; zie afbeelding 8.

7. Refactoring

Visual Studio 2005 ondersteunt refactoring. Hieronder verstaan we het op een gecontroleerde manier structureren van bestaande code met behulp van een aantal kleine aanpassingen/transformaties, zonder dat daardoor het externe gedrag van de code wordt aangepast. Martin Fowler heeft hier een goed boek over geschreven: 'Refactoring: Improving the Design of Existing Code'. In dit boek



Afbeelding 10. Aanpassing namespace



Afbeelding 11. Voorbeeld xml-visualizer

staan technieken beschreven die zelfs zonder Visual Studio 2005 van toegevoegde waarde en direct toepasbaar zijn.

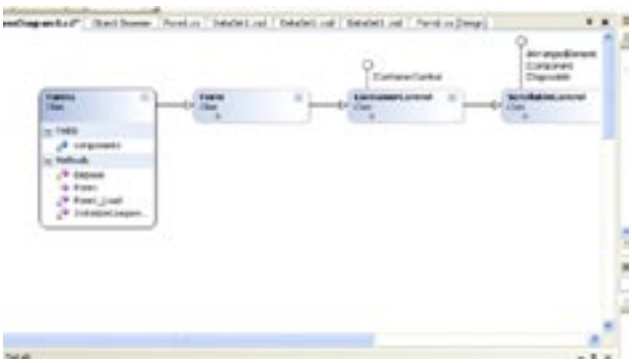
Totnogtoe worden zeven van de door Fowler gedefinieerde refactoring-technieken in de huidige bètaversie ondersteund:

- rename: aanpassen van de naam van een methode, property of field
- extract: promoten van een selectie uit een grote methode naar een zelfstandige methode
- delete parameter: verwijderen van parameters uit een methode
- promote local variable to parameter: toevoegen van een parameter aan een methode
- reorder parameters: verwisselen van twee parameters in een methode
- extract Interface: opstellen van een interface aan de hand van een aantal methodes
- encapsulate field: toevoegen van een property voor een field.

Alle refactoring-technieken zijn voor de hele solution werkzaam. Het voordeel van refactoring boven zoeken en vervangen is dat Visual Studio 2005 met behulp van de IntelliSense in staat is keuzes te maken voor bijvoorbeeld het hernoemen van een methode. Alle referenties worden geüpdate en methodes met dezelfde naam van andere classes worden netjes met rust gelaten. De refactoring-methodes zijn intelligent. Bij het extraheren van een methode wordt afhankelijk van de selectie automatisch bepaald of er parameters moeten worden doorgegeven, of die parameters 'by reference' moeten zijn, wat het return type is en of het een statische of een member methode moet zijn. In afbeelding 9 is te zien dat Visual Studio bepaalt dat er bij het extraheren van een 'Swap'-methode twee reference parameters aan een statische functie moeten worden meegegeven. Aangezien ik persoonlijk wel eens tot de conclusie kom dat mijn code door andere namen of een extra parameter beter wordt, zal deze functionaliteit veel tijd gaan besparen. Zeker indien een namespace voor een groot project wordt aangepast, zoals in afbeelding 10.

8. Debugger visualizers

Op het gebied van debugging is er veel verbeterd. Voor de VB.NET-developers is 'edit and continue' weer terug van weggeweest. De C#-developers moeten helaas nog even wachten. Een verbetering die op beide talen van toepassing is, zijn de debug visualizers. Dit zijn speciale viewers voor datatypes. Voor ingewikkelde datastructuren zoals een dataset is het moeilijk om snel en overzichtelijk de gegevens in de dataset te bekijken via het standaard watch window. Door in het watch window via een dropdown-menu een visualizer te selecteren, wordt een apart window getoond met de gegevens van de dataset. Visual Studio 2005 bevat standaard verschillende visualizers. Voor een string zijn tekst-, xml- en html-visualizers beschikbaar. Een voorbeeld van de xml-visualizer is weergegeven in afbeelding 11. Daarnaast kun je voor elk datatype ook zelf een visualizer maken, waarmee data wordt getoond op een manier die de developer het handigst vindt.



Afbeelding 12. Class diagram: extra manier om code te bekijken en aan te passen

9. Class diagrammen

Iedere developer heeft zijn eigen denk- en werkwijze. Voor iemand die altijd in code denkt, is het Code Definition View Window een welkome aanvulling. Voor wie daar weinig toegevoegde waarde in ziet, zijn de class-diagrammen in Visual Studio 2005 mogelijk een uitkomst. Visual Studio 2005 is geen case tool, maar bevat met het class-diagram nog een extra manier om code te bekijken en aan te passen; zie afbeelding 12. Het class-diagram is nog het beste te vergelijken met een DataSet editor. Aanpassingen in het class-diagram worden onmiddellijk in de code getoond en vice versa. Er wordt geen UML-notatie gebruikt, maar wel een variant die daar erg op lijkt. Bestaande classes kunnen worden toegevoegd door ze te selecteren of te slepen op het diagram. De gebruiker kan zelf beslissen of hij member variabelen wil zien als zelfstandige class met een associatie of als een field. Deze functionaliteit maakt het mogelijk om alleen belangrijke classes apart in het diagram te tonen. In het class-diagram kan de developer ook refactoring toepassen. Omdat een class-diagram een totaaloverzicht op de code biedt, zijn de gevolgen van een wijziging inzichtelijker. Het class-diagram is dan ook vooral bedoeld als een alternatieve manier om door de code te navigeren, met een beter overzicht van zowel bestaande als nieuwe code als resultaat.

10. En elf, en twaalf en meer...

Niet alle onderdelen zijn een eigen paragraaf waard, maar Visual Studio 2005 bevat nog heel veel meer kleine wijzigingen die handiger zijn of beter werken dan in vorige versies. Bij foutmeldingen en waarschuwingen na het compileren verschijnt bijvoorbeeld een error list, zodat het nu veel gemakkelijker is om alleen op errors te filteren. Of het docken van vensters: in de vorige versie toch echt een kleine ramp, die nu grotendeels verholpen is. Bij het docken wordt namelijk extra visuele ondersteuning geboden. Er verschijnen pijlen om aan te geven waar gedockt kan worden en voordat je het venster met de muis loslaat, wordt met een grijs vlak exact aangegeven waar gedockt wordt. Of het markeren van gewijzigde regels: in bronbestanden wordt nu in de kantlijn met kleuren aangegeven welke regels in de huidige sessie zijn gewijzigd (groen) en welke regels nog niet bewaard zijn (geel). Dit geeft een veel beter overzicht, zeker als het om grote bestanden gaat met veel coderegels.
Zo zou ik nog wel even kunnen doorgaan. Een upgrade naar Visual Studio 2005 is dus de moeite waard vanwege de talrijke productiviteitsverbeteringen in de IDE. Microsoft legt met dit totaalpakket aan verbeteringen weer een nieuwe standaard neer, die code beter inzichtelijk en aanpasbaar maakt en daardoor de snelheid en de kwaliteit van code verhoogt.

Nuttige internetadressen

http://lab.msdn.microsoft.com/vs2005/
http://msdn.microsoft.com/vstudio/
Rss feed: http://msdn.microsoft.com/vs2005/rss.xml

Robert van der Kleij is technisch directeur bij Qurius ETX. Hij is eindverantwoordelijk voor de door Microsoft gecertificeerde SoftwareStraat.NET. Voor vragen of opmerkingen kun je mailen naar robertk@etx.nl.