Marcel de Vries is werkzaam bij Info Support in Veenendaal

Team Development met .NET

VOORSPELBAAR KWALITATIEF HOOGWAARDIGE SOFTWARE REALISEREN BINNEN TIJD EN BUDGET

Visual Studio Team System is de nieuwe Visual Studio productsuite die Microsoft eind zomer 2005 op de markt zal brengen. Hiermee breidt Microsoft het product uit om het gezamenlijke werken in een ontwikkelteam beter te ondersteunen. Dit artikel geeft een impressie van alle nieuwe tools die in deze productsuite worden geleverd. Hierbij zal de focus liggen op de tools speciaal voor de ontwikkelaars.

De Visual Studio productsuite bestaat, naast de vertrouwde Visual Studio-omgeving, uit een aantal onderdelen te weten: Team Architect, Team Developer, Team Test en Team Foundation. Samen vormen deze onderdelen Visual Studio Team System (VSTS). VSTS helpt het ontwikkelteam bij het bouwen van kwalitatief hoogwaardige software in een zo kort mogelijk tijdbestek. Daarnaast biedt VSTS ondersteuning aan de levenscyclus van softwareontwikkeling in de vorm van procesbegeleiding binnen de tools.

In afbeelding 1 is aangegeven hoe de productsuite wordt opgedeeld en uitgeleverd in de vier verschillende producteenheden.

Laten we de afzonderlijke onderdelen eens in detail bekijken, te beginnen bij de gemeenschappelijke basis: Team Foundation. De Team Foundation bestaat uit de componenten die er voor moeten zorgen dat de communicatie en uitwisseling van gegevens in een projectteam zo eenvoudig en foutloos mogelijk verlopen. We laten de afzonderlijke producten de revue passeren en beschrijven op welk onderdeel de ontwikkelaar wordt geholpen in zijn dagelijkse werkzaamheden.

Work item tracking (Code name Currituck)

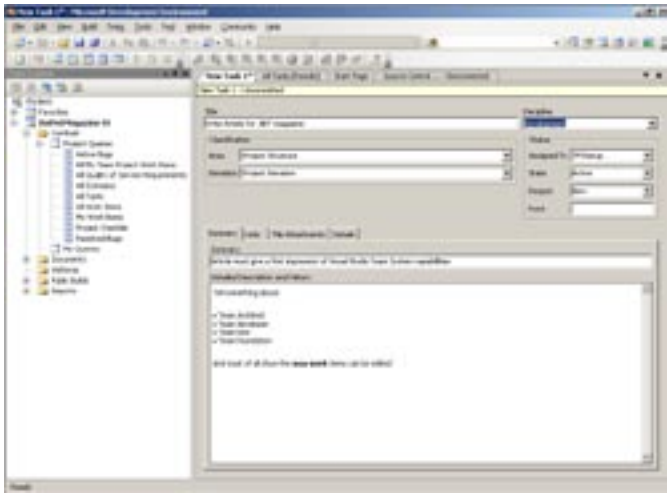
Afhankelijk van het project heeft men een manier gevonden om met elkaar tot een planning te komen en stukjes werk te verdelen in het team. Soms staat alles in een Excel-sheet, soms in een MS Project Schedule, maar in de meeste gevallen is het werk op meer

Afbeelding 1. Visual Studio Team System-productsuite

plaatsen in verschillende formaten beschreven en natuurlijk zijn al deze bestanden niet gesynchroniseerd. Iedere week heeft het team ook het gebruikelijke statusoverleg. Dat is die vergadering waarin iedereen een beurt krijgt en mag vertellen waarom hij nog niet klaar is met zijn werk. Persoonlijk vind ik deze vergaderingen erg slaapverwekkend en vind ik ze ook meestal veel te lang duren. Work item tracking (WIT) kan worden gebruikt om bovengeschetste vergaderingen en de zoektocht naar de status van de planning te vermijden. Met WIT kun je alle werkzaamheden van het projectteam vastleggen in Team Foundation en toekennen aan een projectlid of -groep. In WIT zijn de soorten werk die je wilt vastleggen voor de start van een project gedefinieerd en kan iedereen in het team (die daar de juiste rechten voor heeft) work items van een specifiek type aan de WIT-database toevoegen of aanpassen.

Alle projectleden kunnen zelf aan de hand van queries definiëren welke lijst met work items voor hen belangrijk is. Een ontwikkelaar kan de work items benaderen vanuit de Team Explorer in Visual Studio. Hij kan hier de work items inzien en deze ook invoeren of veranderen in de speciaal beschikbare work item editor; zie afbeelding 2. Een projectleider kan echter gebruikmaken van de door hem zo geliefde omgeving Excel of MS Project. Door de hechte integratie van work item tracking en de Office-tools kan de projectleider de work items niet alleen inladen, maar ook in Microsoft Project of Excel bewerken en weer terugzetten in de WIT-database. Doordat op deze manier iedereen in het team altijd een actuele kijk heeft op het onderhanden werk, kunnen de statusvergaderingen een heel ander karakter krijgen waardoor alleen nog wordt gefocust op problemen, risico's en hoe deze te verhelpen. Het rapporteren van de status is dan opgelost met behulp van WIT.

Een work item heeft naast gegevens over het onderhanden werk ook per type een workflow. Deze workflow is vastgelegd in een set met statussen en statusovergangen. Door de work items van status te veranderen en toe te kennen aan een ander projectlid komen ze bij andere teamleden terecht die er vervolgens weer mee aan de slag kunnen. WIT is zelfs zo diep geïntegreerd in Visual Studio dat, bijvoorbeeld bij het constateren van een fout bij het runnen van een unit-test, er een contextmenu is waarmee de fout kan worden vastgelegd in WIT; zie afbeelding 3. Hierbij wordt alle relevante informatie direct gelinkt aan het work item, zodat een ontwikkelaar voor het reproduceren van het probleem alle informatie direct beschikbaar heeft. Op deze manier wordt voorkomen dat de ontwikkelaar continue incidenten toegewezen krijgt waarvan hij



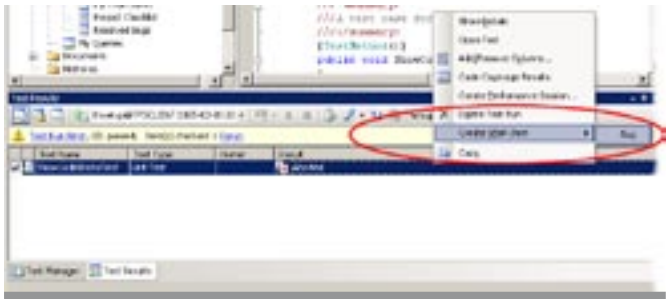
Afbeelding 2. Team Explorer en work item editor

niet kan achterhalen waardoor ze zijn opgetreden. Door de directe beschikbaarheid van alle metadata kan de ontwikkelaar veel beter het incident reproduceren. Hiermee zal het aantal niet te reproduceren incidenten aanzienlijk verminderen.

Work items kunnen ook worden gerelateerd aan additionele informatie die aanwezig is in een project. Zo kan bijvoorbeeld bij het inchecken van broncode worden aangegeven bij welke work items de code hoort. Dit maakt het mogelijk codeveranderingen te herleiden naar een requirement of scenario (traceability). Doordat in Team Foundation standaard één datawarehouse aanwezig is, dat wordt gevoed vanuit alle producten in Team System, kunnen allerlei rapportages worden gemaakt. Denk hierbij bijvoorbeeld aan overzichten van requirements die de meeste bugs hebben opgeleverd of het aantal bugs gerelateerd aan de code coverage die is behaald op een onderdeel van het product. Standaard zal een aantal van dit soort rapportages aanwezig zijn. Daarnaast kun je zelf ook rapportages samenstellen. Interessant om te melden is dat WIT is gebaseerd op een door Microsoft intern gebruikte tool dat door het overgrote deel van de ontwikkelteams wordt gebruikt. Na de komst van WIT zal ook Microsoft intern overgaan op deze nieuwe versie van het product. Sterker nog, op dit moment wordt door een deel van het VSTS-ontwikkelteam al gebruik gemaakt van WIT onder het motto: Eat your own dog food.

Software Configuration Management (codenaam Hatteras)

Samen met een team werken aan diverse bestanden kent zo zijn uitdagingen. Wie is op welk moment bezig met welk bestand? Wat is de laatste versie en waar kan ik die vinden? Als je bij elkaar op één kamer zit is dit misschien nog wel te doen door continu aan elkaar te vertellen waar je mee bezig bent. Hierdoor voorkom je dat je bestanden van elkaar overschrijft. In wat grotere teams, die werken met honderden bestanden en tegenwoordig vaak geografisch van elkaar zijn gescheiden, is dit geen werkbare oplossing meer. Een deel van deze problematiek konden we al oplossen door gebruik te maken van Visual SourceSafe. Dit stelt ons in staat om gezamenlijk met dezelfde bestanden te werken en deze te vergrendelen, zodat niemand anders tegelijkertijd aan hetzelfde bestand kan werken. Visual SourceSafe is echter een Source Control-systeem dat prima kan werken voor kleine teams op een netwerk met voldoende bandbreedte. SourceSafe kan echter snel uit zijn jasje groeien en loopt, door de gekozen architectuur, het risico dat de repository corrupt raakt. Deze problematiek treedt voornamelijk op wanneer de omvang van de repository boven de 2 GB komt. Om voor de enterprise-ontwikkelaar een oplossing te bieden en daarbij ook grote ontwikkelteams te ondersteunen die op geografisch gescheiden locaties werken, is een nieuw systeem gebouwd, het Software Configuration Management (SCM) met de codenaam Hatteras.



Afbeelding 3. Work item invoeren op basis van het testresultaat

Software Configuration Management is een belangrijk onderdeel van het Integrated Change Managementsysteem in Team Foundation. Integrated change management is de samenwerking tussen de onderdelen WIT, WIT-workflow en SCM. Hatteras is volledig opnieuw geschreven en gebaseerd op SQL Server 2005. Hatteras dient niet verward te worden met de nieuwe versie van Visual SourceSafe. Visual SourceSafe zal ook in een verbeterde vorm beschikbaar zijn voor teams van kleinere omvang. Hatteras is gepositioneerd als een enterprise class configuration management-systeem.
Met de komst van SCM wordt een aantal concepten geïntroduceerd die voor Visual Studio-ontwikkelaars relatief nieuw zullen zijn. Voor de doorgewinterde ontwikkelaar die al meer software configuration managementsystemen heeft gebruikt, zullen deze termen redelijk bekend voorkomen. De termen die worden gebruikt zijn onder andere Workspace, Working folder, ChangeSet, Pending changes en Shelving.
Laten we beginnen bij het concept van een Workspace. Een Workspace is een lokale kopie van de bronbestanden op het lokale filesysteem. Een Workspace biedt een locatie waarop je veranderingen kunt aanbrengen op de bronbestanden zonder daarbij direct andere ontwikkelaars in de weg te zitten. Als ontwikkelaar kun je indien gewenst meer van deze Workspaces aanmaken. Als je werkt aan één enkel project zul je maar één Workspace nodig hebben. Het gebruik van meer Workspaces is zeer nuttig als je bijvoorbeeld werkt aan verscheidene releases van een softwareproduct op hetzelfde moment.
Een Workspace wordt geregistreerd in Hatteras en wordt gebruikt voor het bepalen van de veranderingen in de bestanden. Een wijziging in een bestand is altijd een wijziging ten opzichte van de versie die in de Workspace is opgehaald. Alle veranderingen op de Workspace worden bijgehouden en aangemerkt als een pending change.

Verder kennen we het begrip Working-folder, dat al bekend is uit de Visual SourceSafe-omgeving. Een Working-folder geeft de locatie aan binnen een Workspace waar bestanden naar toe worden gekopieerd vanuit de repository. Een Workspace kan meer Working-folders bevatten. De relatie tussen Working-folders en een Workspace wordt een Workspace-mapping genoemd. De gewenste bestanden kunnen in een Workspace worden geladen met het Get-commando. Het 'getten' van de bronnen kan op basis van een aantal criteria waaronder Changesets, Labels, Date, Workspace en Latest. Belangrijk is dat bij het uitvoeren van het Get-commando op een Workspace lokale veranderingen nooit worden overschreven. Wel worden onveranderde bestanden overschreven met een laatste versie uit de repository. Een best practice is om altijd, voor het uitvoeren van een check-in, eerst een Get-commando uit te voeren. Vervolgens worden na het compileren en testen de bestanden ingecheckt. Op deze manier wordt direct de integratie met andere wijzigingen in de bestanden getest.

Alle veranderingen die je maakt op bestanden in een Workspace worden gemarkeerd als een pending change. Je kunt een pending change vervolgens doorvoeren in de repository door het uitvoeren van één van de commando's: Check in, Shelve of Undo. Hierbij is de Undo het eenvoudigste commando. Undo zal geen veranderingen doorvoeren in de repository en de lokale kopiebestanden uit

Stap naam	Beschrijving
PreSync	De stap waarin wordt bepaald welke bronnen moeten worden opgehaald uit de bronrepository en welk buildnummer de build zal krijgen. In de presync wordt ook de bron gelabeld zodat deze altijd weer terug te halen is om de build opnieuw uit te voeren.
Sync	De stap waarbij de daadwerkelijke bronnen worden opgehaald naar een locatie waar een build kan worden gedaan.
PostSync	De stap waarbij nog veranderingen kunnen worden aangebracht in de bronnen voordat deze worden gecompileerd. Denk daarbij aan het aanzetten van XML-documentatie in de projectbestanden. Dit is ook het moment waarop bijvoorbeeld de code churn wordt berekend. Code churn betreft alle veranderingen die in de code zijn uitgevoerd. Dit is een lijst met verwijderde, veranderde en toegevoegde regels.
Compile	De stap waar de compilatie en statische codeanalyse wordt uitgevoerd (fxcop).
Test	De stap waarin naast het uitvoeren van de unittesten ook de code coverage wordt bepaald voor de testen die zijn uitgevoerd.
Analyze	De stap waar alle data in het datawarehouse wordt geladen en eventuele rapportages worden gepubliceerd.

Tabel 1. Team Build-stappen

de repository weer vervangen in de Workspace. Hierdoor worden alle veranderingen ongedaan gemaakt.

Check-in zorgt ervoor dat de veranderingen aan de bestanden in de Workspace worden doorgevoerd in de repository. Bij het inchecken worden bestanden geselecteerd die als één atomaire actie naar de server moeten worden doorgevoerd. Hierbij wordt ook meta-informatie over de check-in verzameld zoals Related workitems, Check in notes, Code reviewer, Datum, Tijd en Revisie. Een check-in is altijd atomair. Deze atomair ingecheckte set met bestanden en de metadata worden aangemerkt als een Changeset.

Bij het inchecken wordt een vaste workflow doorlopen waarbij een aantal controles en acties wordt uitgevoerd voordat de bestanden in de repository komen te staan. De eerste stap in het check-in-proces is controleren of de pending changes voldoen aan de voor de repository geldende check-in policies. Een policy is een in Hatteras geregistreerde .NET-assembly die controles kan uitvoeren tijdens het check-in-proces. Een voorbeeld van zo'n policy is dat er tenminste één Workitem is gerelateerd aan de code die wordt ingecheckt. Policies zijn een onderdeel van een proces-template die men kiest bij het aanmaken van een nieuw project in Team Foundation. Policies zijn ook zelf te bouwen en kunnen eenvoudig worden toegevoegd aan het project. Indien niet wordt voldaan aan een check-in-policy kan de ontwikkelaar er voor kiezen deze policy te negeren. Hij dient hierbij wel altijd een reden op te geven. De policy-violations en bijbehorende reden worden in dit geval via e-mail gerapporteerd aan de projectleider. Microsoft levert een aantal standaard policies, maar het is de verwachting dat andere softwarebedrijven en de community hier ook een bijdrage aan leveren.

Als aan de policies is voldaan, wordt vervolgens gecontroleerd of er nog andere veranderingen aan de bestanden zijn geweest in de tijd dat de bestanden in de huidige Workspace zijn aangepast. Bij het uitchecken van een bron wordt standaard het multiple check-out-principe gehanteerd, waarbij een bestand niet exclusief wordt vergrendeld. (Dit in tegenstelling tot wat we gewend zijn bij het gebruik van Visual SourceSafe). Hierdoor kan er een merge-conflict optreden waardoor een automatische of handmatige merge noodzakelijk is. Nadat eventuele merge-conflicten - automatisch of door de ontwikkelaar - zijn opgelost, is er nog één laatste stap. Deze voert de zogenaamde keyword-expansion uit. Hierbij worden markeringen in de broncode ingevuld met informatie van de check-in. Denk hierbij aan versienummers, naam van ontwikkelaar, related Workitems of naam van de code-reviewer.
Het check-in-proces is zelf aan te passen door het definiëren van eigen policies en check-in notes. Een check-in note is informatie over een check-in. Denk hierbij aan review-commentaar van de code-reviewer. De check-in notes kunnen naderhand gebruikt worden om diverse lijsten te genereren, waaronder release-notes.

Het Shelve-commando biedt de mogelijkheid om werkzaamheden, die nog niet mogen worden ingecheckt, wel op te slaan in de repository. Dit is te gebruiken voor verschillende doeleinden waaronder het beschikbaar stellen van code aan medeteamleden voor een code-review of voor het veiligstellen van werkzaamheden als er tijdelijk aan iets anders moet worden gewerkt. Een set met bestanden die wordt geshelved, kan door een unshelve-commando worden opgehaald uit de repository en in een gewenste workspace worden geplaatst. Shelving kun je prima gebruiken om er aan het eind van de dag voor te zorgen dat de bronnen veilig zijn opgeborgen op de server, terwijl de code misschien zelfs nog niet eens compileerbaar is. Op deze manier wordt voorkomen dat de ontwikkelaar werk verliest, omdat onderhanden werk niet in een back-up is meegenomen.

Team Build (codenaam Big Build)

Wat is de status van het project? Dat is een van de belangrijkste vragen waarmee het projectmanagement zich dagelijks bezighoudt. Om een antwoord te krijgen op die vraag kan de projectleider nu gebruik maken van WIT. Maar hoe staat het met het aantal features in het product? Wat is de kwaliteit van de tot nu toe gebouwde features? Hoeveel testen zijn er uitgevoerd? Om er voor te zorgen dat iedereen een consistent beeld krijgt van de status van het product is een dagelijkse build van cruciaal belang. Een dagelijkse build zorgt er voor dat alle bronnen gebouwd worden tot eindproduct en dat op dit product iedere dag een set met testen kan worden losgelaten. De resultaten van deze build en deze testen kunnen vervolgens worden geladen in een centrale datawarehouse waaraan allerlei projectgerelateerde informatie kan worden toegevoegd. Al deze informatie kan inzicht geven in de status van het eindproduct in termen van features, stabiliteit, kwaliteit, enzovoort. Op deze manier is het mogelijk voor iedereen een centrale plaats te maken waar informatie over het project beschikbaar is. Het zelf inrichten van een dagelijkse build is geen eenvoudige klus. Je dient daarbij een build te maken op basis van diverse scripts met Nant of MSBuild.

In Team System is de dagelijkse build een onlosmakelijk onderdeel geworden van een project. Zodra je een project aanmaakt in de Team Explorer, wordt een dagelijkse build voor je aangemaakt. Deze build bestaat uit een aantal vaste stappen waarvan de details zijn aan te passen. Op dit moment zijn de stappen onderkend die terug te vinden zijn in tabel 1.

Team Build is geheel gebaseerd op MSBuild. MSBuild lijkt sterk op het welbekende MAKE waarbij het script wordt geschreven in XML. In het script staat hoe de bronnen moeten worden gebouwd, getest en wat de onderlinge afhankelijkheden zijn. Team Build maakt gebruik van Hatteras Workspaces voor het uitvoeren van een build. Bij het definiëren van een build dient de Workspace te worden geselecteerd. Er kunnen ook meerdere buildtypen worden gedefinieerd die elk hun eigen Workspace gebruiken.
Doordat Team Build volledig is gebouwd op MSBuild, is het aanpassen of uitbreiden van een build relatief eenvoudig. Team Build maakt voor iedere build een projectbestand aan. Dit projectbestand maakt op zijn beurt gebruik van twee additionele bestanden, te weten *BigBuild.Target* en *CustomActions.Target*. Het BigBuild.Target-bestand bevat alle stappen die standaard worden uitgevoerd door de build. In het CustomActions.Target-bestand kun je zelf taken definiëren

die moeten worden uitgevoerd gedurende de build. Op dit moment heeft men hiervoor de volgende mogelijkheden beschikbaar: CustomPreSync, CustomSync, CustomPostSync, CustomCompile, CustomTest en CustomAnalysis. De PreSync, Sync en PostSync zijn bedoeld om in te kunnen grijpen op het samenstellen van de build.

Project Portal

Als op basis van een dagelijkse build zoveel informatie beschikbaar komt, waar kan ik dan alle informatie in een oogopslag terugvinden? Om die vraag te beantwoorden is Team System voorzien van een Project Portal die wordt gecreëerd zodra je een project aanmaakt in de Team Explorer. Deze portal toont een aantal rapportages die uit het datawarehouse komen. Deze geven het hele team een indicatie van de status van hun project. Voor ieder lid van het team zijn er speciale document-libraries aangemaakt waar iedereen zijn documenten kan delen met het team. De portal is volledig gebaseerd op Windows Sharepoint services. Dit is één van de redenen waarom Team System op een Windows 2003 server moet worden geïnstalleerd.

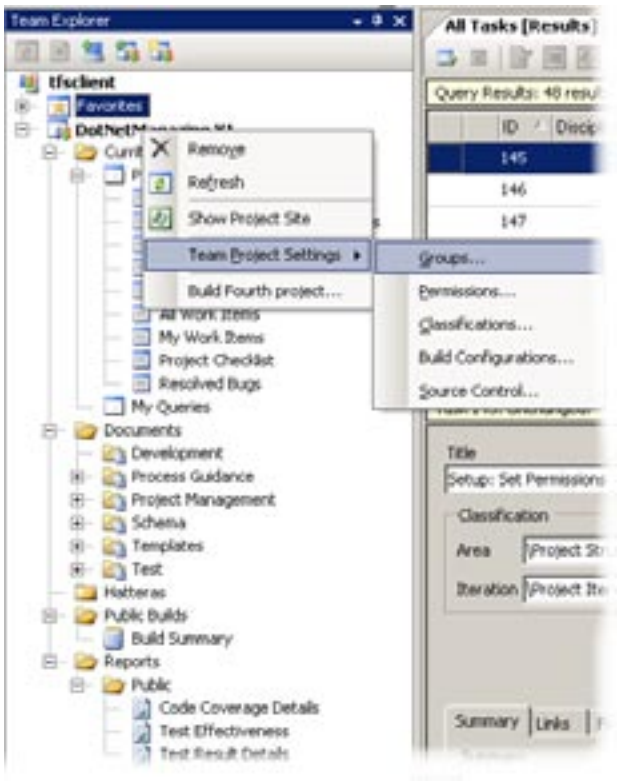
Team Explorer

Alle onderdelen van Team Foundation zijn in de Visual Studio IDE te benaderen via de Team Explorer; zie afbeelding 4.

Proces-guidance

Bij het aanmaken van een nieuw project in de Team Explorer kiest men een proces-template voor het project. Deze template bevat de definitie van een aantal zaken waaronder Work item-types, Work item workflows, check-in policies, gewenste rapportages en welke proces-guidance-website beschikbaar komt voor het team. In de huidige preview is alleen MSF Agile beschikbaar. Als je gebruik maakt van MSF Agile, dan zijn er standaard de Work item-types Task, Bug, Requirement, Scenario en Quality of service in de Workitem tracking-database beschikbaar. Verder is op dit moment één check-in-policy actief die verplicht dat een code-reviewer wordt opgegeven bij het inchecken. Ook is er een proces-guidance website beschikbaar die MSF Agile beschrijft.

Het is mogelijk zelf een proces te definiëren door de templates te definiëren of aan te passen naar eigen behoefte. Standaard zal Team



Afbeelding 4. Team Explorer

System worden voorzien van de processen MSF 4.0.Agile en Formal. De verwachting is dat kort na de lancering van VSTS andere partijen met hun procesbeschrijving zullen komen die volledig integreert met het product.

Team Architect

Naast het servergedeelte bevat Team System ook een set tools die onder de Visual Studio-omgeving valt. Dit zijn de tools die specifiek voor een doelgroep zijn ontwikkeld. Hierbij wordt, zoals eerder opgemerkt, een opdeling gemaakt in Architect, Developer en Tester. De architect krijgt een aantal designers geleverd waarmee hij een eerste ontwerp kan maken van een applicatie. Deze designers worden de Distributed System Designer en de Logical Datacenter Designer genoemd. In de Distributed System Designer kan een architect diagrammen maken en aangeven uit welke onderdelen de applicatie moet gaan bestaan en hoe deze onderling met elkaar communiceren. Met de Logical Datacenter Designer kan de architect een diagram maken dat beschrijft hoe het datacenter is ingericht. Met de twee diagrammen kan vervolgens een simulatie worden uitgevoerd of de applicatie, zoals bedacht, ook daadwerkelijk in het datacenter te installeren is. Het uitvoeren van een simulatie wordt een test-deployment genoemd. Verder kunnen vanuit de diagrammen de basisstructuren voor de solution en de bijbehorende projecten worden gegenereerd. Op deze manier kan de architect de structuur al volledig voordefiniëren en kan de ontwikkelaar vervolgens zijn werk richten op het implementeren van features en componenten. Er wordt ook een class designer meegeleverd. Deze designer is te gebruiken als alternatieve view op de broncode en zal altijd gesynchroniseerd zijn met de bronnen. De designers zijn een eerste implementatie die onderdeel uit maken van Microsoft's visie op software-factories. Hierbij wordt gebruik gemaakt van Domain Specific Languages (DSL) voor de specificatie van een product. Hiervoor komt ook een DSL-designer beschikbaar die iedereen in staat stelt zijn eigen designer te maken die integreert met Visual Studio en naast visualisatie ook codegeneratie tot doel heeft.

Team Developer

Als ontwikkelaar dien je de broncode te laten voldoen aan standaarden en richtlijnen die in het bedrijf of de afdeling gelden. Daarnaast is het van belang dat je op de juiste manier gebruik maakt van het .NET Framework. Het zou erg prettig zijn als je van tevoren op de hoogte wordt gesteld van mogelijke fouten in het programma, voordat dit aan een test wordt onderworpen. Om de ontwikkelaar hierbij te helpen biedt Team Developer twee additionele tools, te weten statische en dynamische code-analysetools. Deze tools zijn erop gericht de ontwikkelaar vanaf het eerste moment te helpen bij het schrijven van foutloze code die voldoet aan geldende richtlijnen. Daarnaast wordt geanalyseerd of het .NET Framework op de juiste manier wordt toepast zodat bekende problemen of veel gemaakte fouten vroegtijdig kunnen worden opgelost.
De statische code-analysetools zijn waarschijnlijk bij de meeste ontwikkelaars al wel bekend onder een andere naam, namelijk fxcop.exe. Fxcop is in Team System volledig geïntegreerd met de Visual Studio IDE. Als je regels wilt toevoegen of verwijderen, is dit mogelijk in de project-properties. Ook kun je aangeven of je bij iedere compilatie automatisch Fxcop-controle wilt uitvoeren. Ook dit is in de project-properties in te stellen. De FxCop-controles worden ook uitgevoerd als een project wordt gecompileerd vanuit Team Build.

De dynamische code-analyse betreft de 'profiling' van een applicatie. Met profiling kunnen performancebottlenecks in de applicatie worden opgespoord. Profiling is beschikbaar in twee varianten, te weten Instrumentation en Sampling. In het geval van instrumentation bewerkt een preprocessor de bestaande applicatie en worden in de MSIL van de applicatie probes geïnjecteerd. Een probe is een meetpunt dat bijhoudt hoe vaak en hoe snel een deel van de code is doorlopen. Nadat een test heeft gelopen, worden alle probewaarden uitgevraagd en kan het profiel worden getoond. De twee-

de manier van profiling werkt op basis van sampling. Hierbij wordt op een vast interval gekeken naar welke functie op dat moment op de callstack staat. Dit wordt per sample vastgelegd, zodat na het eindigen van de test de resultaten kunnen worden getoond. De sampling-methode heeft een veel lagere performance-impact op een applicatie die wordt getest dan het gebruik van instrumentation. De sampling-methode geeft echter een minder nauwkeurig beeld en er kunnen methodes worden gemist. Microsoft biedt beide mogelijkheden aan, omdat het zelf deze tools intern inzet vanwege hun unieke eigenschappen. De instrumentation-profiler is een commerciële versie van een interne tool die door het SQL- en Windows-team wordt gebruikt. De sampling-profiler zet Microsoft intern in bij de ontwikkeling van de Xbox.

Test Driven Development

Tijdens het bouwen van het softwareproduct ben je als ontwikkelaar continu bezig functionaliteit te bouwen die wordt aangedragen door de business. Hoe wordt deze gewenste functionaliteit vastgelegd in het team en hoe kun je valideren of aan de specificaties wordt voldaan? In de development-community is al enkele jaren het concept Test Driven Development in zwang. Hierbij worden de functionele eisen van de programmatuur vastgelegd in een set testen die je op de geschreven code kunt loslaten. Met deze testen bewerkstellig je dat je zelf eerst helder krijgt wat de eisen zijn, waarna het makkelijker wordt om te voldoen aan de gestelde eisen uit het testscenario. In plaats van de standaard testforms met een grote knop voor het aftrappen van een stuk code waar je doorheen moet debuggen, beschik je nu over een testomgeving die je daarbij volledig ondersteunt.

Een belangrijk concept dat wordt gebruikt bij Test Driven Development is het schrijven van een UnitTest. Een unittest is een stuk code dat andere code test. Hierbij is de definitie van een unit bijna altijd een class. Bij het schrijven van unittesten wordt gebruik gemaakt van testdrivers (de class die de andere class test) en van een testtool die deze testdrivers kan uitvoeren. Op dit moment is een veelgebruikte unittest-omgeving NUnit. Dit is een open source-product dat beschikbaar is op http://nunit.org. Een van de auteurs van NUnit werkt nu in het Visual Studio-team aan de testtools die in Team System beschikbaar komen.

VSTS biedt naast het genereren van de testdrivers vanuit het contextmenu ook de mogelijkheid deze testen te ordenen en te groeperen met de Testmanager. Bij de aanpak volgens Test Driven Development is het zeer gebruikelijk testen te schrijven voor code die nog niet bestaat. Hiervoor wordt een zogenaamde stub-implementatie gemaakt waarna de testen kunnen worden gecompileerd. Het genereren van deze stubs is tevens mogelijk vanuit het contextmenu. Hierbij worden de naam van de functie, de argumenten en de types van de argumenten overgenomen en wordt een implementatie gegenereerd die een exception gooit van het type NotImplementedException.

Als een test loopt, worden de resultaten van de testen opgeslagen in testmetabestanden. Deze bestanden bevatten alle gegevens over de test inclusief de resultaten en de test-coverage. Wordt met een test een fout aangetoond in de programmatuur, dan kan de metadata samen met een nieuw Work item worden opgeslagen in de WIT-database; zie afbeelding 3. De programmeur kan vervolgens na het openen van het Work item de opgeslagen testresultaten bekijken en de test lokaal nog eens uitvoeren. Dankzij de volledige integratie van testen in de VSTS-omgeving wordt de communicatie tussen de tester en de ontwikkelaar sterk verbeterd. Zo zijn er voor het inchecken van de broncode in Hatteras standaard policies beschikbaar, die je kunt activeren en die je dwingen eerst een code-coverage-percentage van x% te hebben voordat mag worden ingecheckt.

Team Test

Als je een willekeurige ontwikkelaar of projectleider vraagt naar het testen in het team, dan zijn ze het altijd eens met het feit dat testen zeer belangrijk is. Als je vervolgens vraagt hoeveel er wordt getest dan is het antwoord helaas vaak bedroevend. Testen wordt zeer vaak als sluitpost op de begroting gezien, terwijl het zo cruciaal is voor het bouwen van kwalitatief goede software. Testen kan op een aantal fronten een complexe materie zijn. Neem nu bijvoorbeeld het testen van non-functional requirements voor een website. Vaak zijn in het ontwerp wel responsetijden genoemd, maar hoe toon je aan dat je daaraan voldoet? Sterker nog, hoe zorg je er voor dat je vanaf dag één voldoet aan die eis en hoe kun je bewaken dat aan die eis wordt voldaan bij oplevering? Als je een testomgeving hebt die iedere dag herhaaldelijk de performancetesten kan uitvoeren en rapporteren, wordt het ook een natuurlijke strategie om de besproken tools in te zetten en daarmee de kwaliteit van het eindproduct continu te meten. Met Team Test in combinatie met Team Build en het datawarehouse is dit nu binnen handbereik gekomen.

Voor de webomgeving wordt een record/playback testtool geleverd die een HTTP-request herhaaldelijk kan afspelen en de response kan valideren. Op die manier is het mogelijk te valideren of schermen de juiste teksten bevatten en de juiste schermen na elkaar volgen bij het uitvoeren van de HTTP-verzoeken. Deze webtest biedt ook de mogelijkheid om een load- en stress-test uit te voeren. Hierbij kun je onder andere aangeven wat de browsermix is die moet worden gebruikt (bijvoorbeeld 10% Netscape en 90% IE), welke denktijden moeten worden gebruikt of hoeveel clients moeten worden gesimuleerd. Een bijzonder waardevolle toevoeging aan deze testomgeving is dat Microsoft een set belangrijke performance-counters heeft gedefinieerd. Bij deze performance-counterset hebben zij ook zogenaamde threshold-waarden meegeleverd voor specifieke serverscenario's. Zodra deze waarden worden overschreden bij het testen, kan dit worden gerapporteerd. De resultaten in de rapportage verschaffen direct inzicht in de problemen die een applicatie eventueel bevat. Standaard zijn deze performance-countersets beschikbaar voor onder andere ASP.Net en SQL Server.

Testen wordt zeer vaak als sluitpost op de begroting gezien, terwijl het zo cruciaal is voor het bouwen van kwalitatief goede software.

Goede software

Met Visual Studio Team System biedt Microsoft ontwikkelaars alle mogelijkheden kwalitatief goede software te schrijven in de .NET-omgeving. Hierbij is ook de nodige aandacht besteed aan het managen van een project en daarbij aan de communicatie in een ontwikkelteam.

Marcel de Vries is werkzaam bij Info Support in Veenendaal. Hier is hij als .NET-architect mede verantwoordelijk voor innovatie in het Professional Development Center, waar de software ontwikkelstraat Endeavour wordt ontwikkeld. Vanuit zijn rol in het PDC is hij ook de trekker van het Whidbey Technical Adoption Program (TAP), waarvoor Info Support is geselecteerd als één van de 40 bedrijven wereldwijd. Vanuit het TAP-programma is Marcel al sinds juni 2004 actief aan het werk met Visual Studio Team System.

Nuttige internetadressen

- http://lab.msdn.microsoft.com/vs2005/default.aspx
- http://lab.msdn.microsoft.com/vs2005/teamsystem/
- http://lab.msdn.microsoft.com/vs2005/teamsystem/blogs/default.aspx
- http://lab.msdn.microsoft.com/vs2005/teamsystem/workshop/dsltools/
- http://msdn.microsoft.com/library/en-us/dnvsent/html/vsts-pm.asp
- http://www.nunit.org