Marcel de Vries

Visual Studio Team System "Rosario"

Het zal de meeste ontwikkelaars niet zijn ontgaan: Visual Studio 2008 is gereleased. De Team System omgeving is in Visual Studio 2008 op een groot aantal punten verbeterd en je ziet dat het met deze omgeving steeds beter mogelijk wordt de softwarelifecycle onder controle te krijgen. Misschien ben je net aan de slag gegaan met Visual Studio 2008, of werk je nog gewoon met de Visual Studio 2005-omgeving, toch wil ik in dit artikel alvast een blik werpen op de toekomstige ontwikkelingen op het gebied van Visual Studio Team System (VSTS) en Team Foundation Server(TFS). Zelf ben ik van namelijk mening dat vooruitkijken erg belangrijk is voor het maken van de juiste keuzes vandaag. Door middel van dit artikel hoop ik je inzicht te geven in wat we kunnen verwachten van Microsoft als het gaat om het verder versterken van de tools die we nodig hebben om de steeds maar complexere applicaties goed te ondersteunen.

De volgende versie van VSTS en TFS worden momenteel ontwikkeld onder de codenaam "Rosario". Voor "Rosario" heeft Microsoft een thema gedefinieerd waaraan alle nieuwe features worden opgehangen. Het thema is: "Bouw de juiste software op de juiste manier". Onder deze noemer zul je een aantal toevoegingen gaan zien aan de huidige toolset waarvan momenteel al een subset publiek beschikbaar is in de vorm van specificaties en een Community Technical Preview(CTP). In dit artikel baseer ik mij op de features die men al publiek heeft gemaakt en terug te vinden zijn in de april CTP.

Bouw de juiste software op de juiste manier

Visual Studio Team Architect Edition

Het kan misschien als een schok komen, maar zelf ben ik zeer blij met het feit dat Microsoft er voor heeft gekozen in Team Architect UML 2.0 diagrammen te gaan ondersteunen. Daarbij moet wel direct worden opgemerkt dat het niet het doel is van Microsoft volledig UML te gaan ondersteunen. Men heeft de keuze gemaakt modellen te kunnen gebruiken als ondersteuning voor het bouwen van software en daarbij gebruik te maken van de kennis die men al heeft op dit gebied. Om die reden heeft men er ook voor gekozen alleen de meest gebruikte diagrammen te ondersteunen. De diagrammen die je in ieder geval terug zult vinden zijn: het (Conceptual) Class diagram, het (Conceptual) Sequence Diagram, het Component Diagram en het Use Case Model. In de april CTP kun je al een eerste indruk krijgen van een aantal van de diagrammen. Daarbij moet ik opmerken dat het Sequence Diagram er het meest compleet uitziet. Wat erg krachtig is aan de Sequence Diagrammen is dat men daarbij ook goed heeft nagedacht over het snel vol raken van een diagram waardoor je het overzicht dreigt te verliezen. Doordat je eenvoudig delen van de sequence kunt in- en uitklappen kun je heel snel overzicht krijgen. Tevens heeft men een optie toegevoegd om delen van een sequence te verplaatsen naar een ander diagram dat dan via een "click through" eenvoudig benaderbaar blijft. Op die manier is het prima mogelijk voor een architect zijn intenties duidelijk te maken, waarna de detailinvulling naderhand bijvoorbeeld wordt gecompleteerd door een ontwikkelaar. Er missen in de huidige implementatie van de diagrammen nog een aantal belangrijke features op zowel het UML-vlak als op het vlak van usabillity, maar het geeft al wel een aardig idee waar het uiteindelijk naartoe gaat.
In de manier van gebruik van diagrammen binnen een project zijn er duidelijk twee stromingen te onderscheiden. De top down benadering waarbij de architect/ontwikkelaar begint met het maken van een ontwerp en de bottom up benadering waarbij er reeds sprake is van een bestaande codebase en men van daaruit diagrammen wil kunnen zien.

Team Architect in Rosario ondersteunt UML 2.0 diagrammen

Om die reden heeft Microsoft het ook mogelijk gemaakt vanuit de bestaande code diagrammen te genereren die een afspiegeling zijn van het gemaakte systeem. Hiervoor heeft men een zogenaamde Architecture Explorer gemaakt. Met deze Explorer kun je bij een bestaand product makkelijker inzicht verkrijgen in o.a. de structuur van het product. Je kunt er b.v. eenvoudig de eerder beschreven sequence- of class-diagrammen reverse engineeren. In figuur 1 is een screenshot te zien van de Architecture Explorer. Hierbij zie je in één oogopslag de explorer (onderkant) en het sequence diagram dat is gegenereerd op basis van de broncode in de Team Foundation Server.

Fig. 1: Architecture Explorer en sequence diagram

Visual Studio Team Developer Edition

Binnen de developer edition heeft men zich voornamelijk bezig gehouden met het probleem dat beter bekend is als het "Non Repro" probleem. Dit is het probleem waarbij een tester of eindgebruiker een fout constateert in het product maar dit vervolgens niet kan reproduceren in een ontwikkelomgeving. Het kost vervolgens zeer veel tijd en geld om het probleem boven tafel te krijgen. In sommige gevallen blijkt het zelfs echt onmogelijk om het probleem te reproduceren. Vaak is dit gerelateerd aan andere verschillen tussen productie- en ontwikkelomgeving en in steeds grotere mate gerelateerd aan concurrency problemen. Zeker met de komst van MultiCore-processoren zal dit probleem in de toekomst alleen maar groter worden. Om dit probleem het hoofd te gaan bieden introduceert Microsoft in de developer edition van Team System zogenaamde Historical Debugging. Historical debugging kun je het meest eenvoudig vergelijken met de Flight Data Recorder die in een vliegtuig zit. Deze recorder houdt de belangrijkste gegevens bij tijdens een vlucht en zodra er sprake is van een probleem (in het ergste geval zelfs een crash) dan zal alle benodigde data beschikbaar zijn voor analyse achteraf. Historical debugging doet feitelijk hetzelfde. Tijdens het testen van een applicatie kan men deze "flight recorder" op de achtergrond laten meedraaien. Gedurende het testen (of eventueel zelfs in productie) kan dan de tester bij het rapporteren van een bug de historische execution log aan het Bug-workitem koppelen. Met deze log kan vervolgens de ontwikkelaar direct zien welke codepaden zijn doorlopen. De integratie wordt zelfs zover doorgevoerd dat de debugger te switchen is tussen live mode en historical mode, waarbij in historical mode de ontwikkelaar door de code kan steppen alsof de applicatie live aan het werk is. Hierbij kan de ontwikkelaar alle bekende debug informatie zichtbaar maken, waaronder de waarden van variabelen die gebruikt zijn gedurende de uitvoering van het programma. Zaken als Exceptions, File IO, Registry Access, etc. worden ook allemaal vastgelegd. Je kunt je voorstellen dat dit natuurlijk een zeer grote vooruitgang is om het "No Repro" probleem het hoofd te bieden en dat je dan als ontwikkelaar niet eerst veel energie hoeft te steken in het reproduceren van het probleem voordat je het kunt opsporen. Je kunt dus bij het openen van een Bug direct de log laden, er door heen stappen en zien wat er mis is gegaan in de applicatie.

Historical Debugging werkt als een Flight Recorder

Historical debugging is al een aantal jaren in ontwikkeling bij Microsoft onder de Code Naam "Protheus". Op dit moment heeft men het in de planning zitten de Protheus runtime, die op de achtergrond de traces bijhoudt, als losse executable beschikbaar te stellen zodat je deze als onderdeel van je applicatie kunt meeleveren. Daarmee kun je dan zelfs in een applicatie een optie inbouwen om tijdelijk de Protheus logging aan te zetten. Dan kan zeer waardevol zijn indien de klant een lastig te reproduceren probleem constateert. Door dan de logs te laten opsturen kunnen die lastig te vinden productieproblemen eenvoudiger worden opgelost.

Naast historical debuggen heeft men ook gekeken naar het concept van Impact Analysis. Daarbij heeft men zich momenteel beperkt tot Impact Analysis op het vlak van uit te voeren unittesten ten gevolge van het aanpassen van code. Hierbij gaat het erom dat tijdens het aanbrengen van aanpassingen in de code de omgeving kan bepalen wat de minimaal uit te voeren set unittesten is om te valideren of een aanpassing regressie tot gevolg heeft gehad. Het idee daarbij is dat men vaak enkele tientallen en soms zelfs ettelijke honderden unittesten beschikbaar heeft, maar geen idee heeft welke testen zouden kunnen aantonen of een aanpassing mogelijk regressie tot gevolg heeft gehad. Met behulp van Test Impact Analysis wordt op basis van de code-coverage-data uit de unit testen, die is opgeslagen op de server gedurende de builds, bepaald of een unittest uitgevoerd zou moeten worden. Zodra wordt geconstateerd dat een Unittest coverage heeft op de regels code die aangepast zijn, zal deze unit test als "aanbevolen" test worden aangemerkt. Je kunt dan vervolgens na afloop van het doorvoeren van je verandering deze aanbevolen lijst uitvoeren om te controleren of een van deze testen faalt.

Deze feature is overigens niet alleen erg handig in de IDE, maar ook tijdens het uitvoeren van een Build, zeker in het geval je zogenaamde "Continious Integration" (CI) build hebt aangezet op een project. Daarbij is namelijk het doel zo snel mogelijk een build te maken die aantoont of een change mogelijk problemen veroorzaakt in de totale codebase. Door nu in een CI build alleen de "aanbevolen" testen uit te voeren kan een minimale set unit testen worden gebruikt voor het aantonen van regressie in de build. Dit stelt ons in staat om een CI build zeer efficiënt in te richten.

Visual Studio Team Test Edition

Een van de meest in het oog springende nieuwe features is, denk ik, de nieuwe test tools die beschikbaar komen. In de huidige (2008) Visual Studio Team Test editie is heel duidelijk de "technische" tester de doelgroep. In Rosario heeft men ook heel duidelijk de functioneel tester in het vizier. Dit maakt men mogelijk door de introductie van een nieuwe toolset, met als doelgroep functioneel tester, die buiten de Visual Studio IDE wordt gebruikt. Deze applicatie heeft de naam "Camano". "Camano" is een applicatie die ondersteunt in het opzetten, beheren, plannen en uitvoeren van testen. Hierbij introduceert Microsoft een aantal termen die overal in de tooling terug komen, te weten: Test Case, Test Suite, Test Configuration, Test Plan en Test Pass.

De Functioneel Tester wordt niet langer vergeten

Een test case bevat de stappen om een specifiek onderdeel van een applicatie te testen. Een test case is een workitem dat o.a. de stappen bevat die in een test moeten worden uitgevoerd. Meerdere testcases worden samengevoegd in een test suite, voordat er een test kan worden uitgevoerd. Een test suite kan je in "Camano" samenstellen uit een aantal te selecteren test cases of door middel van een query. Dit laatste noemt men dan een dynamic test suite. Test configurations zijn aanduidingen van omgevingen waar een test wordt uitgevoerd, b.v. Windows Vista SP1 met FireFox browser, of Windows XP SP3 met IE7. Test suites kun je samen met een test configuration gaan inplannen om uit te voeren door de groep testers. Hierbij kun je specifieke testen toekennen aan testers en kun je het test plan activeren zodat de benodigde workitems op de naam van een betreffende tester worden gezet. Het uitvoeren van een test wordt een test pass genoemd. Op een test pass wordt gerapporteerd wat de resultaten van de test waren. De resultaten worden door de tester tijdens het uitvoeren van de test aangegeven in de test tool "MSRun". Deze test tool dient ter ondersteuning tijdens de testuitvoer en biedt een aantal opties (zie figuur 2).

Fig. 2: MSRun test applicatie

De MSRun test tool kent een aantal opties, waaronder het op de achtergrond opnemen van de acties die zijn uitgevoerd, het vastleggen van de machine instellingen zoals CapsLock, NumLock, service packs, etc., het opnemen van een zogenaamde automation strip, etc. Al deze opties resulteren in losse logs die bij het vastleggen van een bug ook de omstandigheden duidelijk maken waaronder de fout is opgetreden. Er wordt zelfs een video-opname gemaakt van wat de tester allemaal heeft uitgevoerd. Bij het rapporteren van een bug worden al deze gegevens opgeslagen bij het workitem, zodat de ontwikkelaar direct kan zien waar het probleem zich heeft voorgedaan. In de toekomst komt daar ook de Protheus log bij, zodat de ontwikkelaar ook meteen door de historische data van de testrun kan stappen om zo het probleem zo snel mogelijk boven tafel te krijgen.

Fig.3: Camano test pass details

Gedurende het uitvoeren van de testen is één van de opties het opnemen van een zogenaamde automation strip. Deze automation strip kan voor twee doeleinden worden ingezet: ten eerste voor de ondersteuning van een hertest en ten tweede voor het automatiseren van de test als een Coded UI Test. Voor de ondersteuning van de handmatige testen kan de automation strip worden ingezet voor het afspelen van stappen die in een voorgaande run ook al eens zijn uitgevoerd. Stel je bijvoorbeeld eens voor dat een testcase 25 stappen bevat en je alleen de laatste stap opnieuw moet controleren. Je kunt dan de test starten, de opgenomen test automation strip uit de vorige test pass gebruiken om automatisch de stappen tot en met stap 24 voor je te laten uitvoeren. Dit wordt gedaan door in het MSRun tool te kiezen voor de playback van de automation strip tot een aangegeven punt. Vervolgens hoeft de tester alleen de laatste stap nog zelf uit te voeren. Deze manier van werken zorgt er voor dat voor regressie testen de scenario's op exact dezelfde manier worden doorlopen en zorgt daarnaast voor een stukje werkgemak voor de tester.

Zoals aangegeven, kun je de automation strip ook inzetten voor het verder automatiseren van test cases. Hiervoor kun je dan een test toevoegen aan een test project van het type "CodedUItest". Dit lijkt erg op een unit test, echter is dit specifiek bedoeld voor UI test automation. In deze test kun je de automation strip gebruiken om code te laten genereren die de stappen volledig automatisch voor je uitvoert. Je kunt dan tevens een zogenaamd verification point toevoegen, die bijvoorbeeld de waarden uit een control uitleest en kan vergelijken met verwachte waarden. Op deze manier kun je relatief eenvoudig de UI testing volledig automatiseren en de kosten van het regressietesten significant reduceren.

De test tools zijn primair bedoeld voor de functioneel testers. Dit is ook direct herkenbaar doordat de tester niet Visual Studio hoeft te gebruiken voor het maken van de test cases, suites, configurations en plans. Ook de rapportage over een test pass wordt volledig in Camano gedaan en geeft een grafisch overzicht van de resultaten tot op dat moment voor een betreffend test plan. Camano zelf is volledig in WPF geschreven en geeft door middel van de Team Foundation Server de mogelijkheid op een juiste manier in het team samen te werken. De test cases, plans en results worden allemaal opgeslagen in Workitem Tracking en een aparte Team Test Database. Op die manier worden ook de resultaten t.a.v de historie keurig bijgehouden. In figuur 3 is een screenshot van Camano weergegeven, waarbij de resultaten van een test pass worden getoond.

Team Foundation Server

Als laatste wil ik nog heel kort kijken naar de Team Foundation Server. Ook aan die kant worden de nodige verbeteringen doorgevoerd. In bijvoorbeeld Version Control wordt er hard gewerkt om er voor te zorgen dat de historie van een change set beter inzichtelijk te maken is. Momenteel komt het nogal vaak voor dat je niet duidelijk kunt zien of een bepaalde change nu wel of niet onderdeel is geworden van een specifieke branch. Daarom maakt men een optie voor het visualiseren van de history van een change set. Op die manier wordt dan inzichtelijk in welke branch de change set op een gegeven moment in de tijd aanwezig is. Dit is zeer nuttig om bijvoorbeeld inzichtelijk te maken in welke versie van een product een bugfix terug te vinden is. In figuur 4 is weergegeven hoe deze visualisatie er uit zal komen te zien:

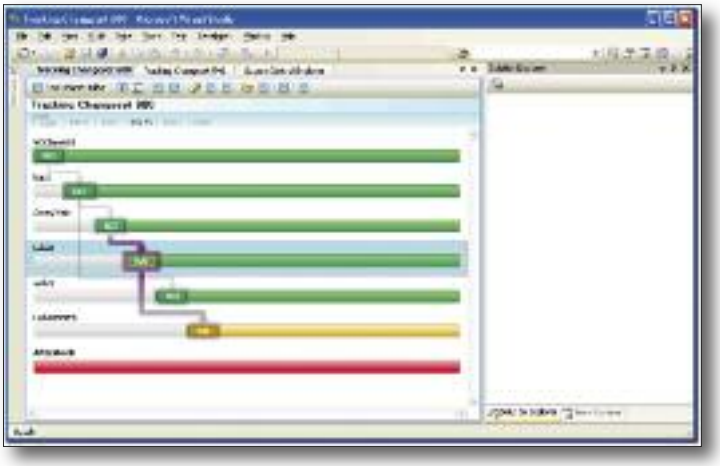

Fig. 4: Change set visualization

Verder werkt men ook hard om zgn. conflict resolving beter voor ons op te lossen. Hierbij is het doel om zoveel mogelijk scenario's te automatiseren, zodat de tooling de juiste keuzes maakt en de ontwikkelaar dit alleen nog hoeft te controleren.

Beter inzicht in change set history …

Als laatste wil ik ingaan op veranderingen die worden gemaakt t.a.v. de Team Build omgeving. In de 2008 versie is er al een aardige aanpassing geweest aan de architectuur van Team Build. Toch heeft men er voor gekozen om ook in "Rosario" een significante verandering in de aanpak te kiezen. Een van de veelgevraagde mogelijkheden is namelijk de optie om niet een eenmalig script te genereren voor de build, maar dat de build definitie continue aanpasbaar blijft vanuit de IDE. Daarbij wil men ook graag dat de build-stappen gevisualiseerd worden. Om die reden heeft het team ervoor gekozen de toekomstige team-build-omgeving wederom volledig opnieuw te bouwen. Deze keer doet men dit op basis van Windows Workflow Foundation (WF).

Door middel van WF kan men namelijk naast de eenvoudige build-visualisatie (Workflow designers) ook een veel robuustere runtime-build-omgeving realiseren. Deze heeft dan standaard ondersteuning voor zaken als parallel uitvoeren van stappen en het distribueren van stappen over machines heen. Last but not least, geeft dit ook de oplossing voor het aanpasbaar houden van de buildstappen door middel van een editor.
In Team Build maakt men het mogelijk een build uit te laten voeren door een op dat moment vrije buildmachine uit een pool aan machines. Hierbij wordt gekeken of een buildserver uit de pool voldoet aan gestelde randvoorwaarden alvorens deze wordt gebruikt om de build te queuen. Het selecteren van een machine uit de pool wordt op basis van zogenaamde Tags gedaan. Door Tags aan een server te hangen en aan een build-definitie kan dan een beschikbare server gekozen worden uit de pool met machines.

Conclusie

Al met al kunnen we stellen dat Team System "Rosario" een significant aantal nieuwe features kent. De focus bij al deze features ligt in het feit dat ze ondersteuning moeten bieden aan het eerder genoemde thema. Door middel van het toevoegen van architectuurtools als UML en een architectuur explorer maakt men het mogelijk ook in een Microsoft omgeving eerst de concepten op te schrijven in diagrammen en vervolgens pas te starten met het implementeren. Met behulp van de nieuwe Team Developer features maakt men het mogelijk sneller fouten op te sporen, wat ons productiever maakt. Door middel van de nieuwe test tools kan beter worden gecontroleerd of alles wat we hebben gemaakt, in overeenstemming is met de afspraken die we hebben gemaakt. Door vervolgens al deze features naadloos op elkaar aan te sluiten maakt Microsoft het mogelijk applicaties steeds efficiënter te ontwikkelen en de kosten voor realisatie en onderhoud in de toekomst verder te reduceren.

Voor mijzelf is "Rosario" zeker een versie waar ik nu al naar uitkijk. Zodra de officiële bèta's beschikbaar komen weten we beter waar het product daadwerkelijk naartoe gaat en of de features die we tot nu toe hebben gezien, ook daadwerkelijk in het eindproduct terug te vinden zullen zijn ●

Marcel de Vries

Marcel de Vries is IT-Architect bij Info Support voor de Business unit Finance en MVP. Marcel heeft na vele jaren ervaring opgedaan met het .NET platform bij het bouwen van Enterprise administratieve applicaties voor grote bedrijven in Nederland.
Naast het schrijven van artikelen voor SDN en Microsoft.NET magazine is hij een veel gevraagde spreker op seminars en conferenties waaronder Microfsoft Tech-Ed, Developer Days en SDC. Naast zijn werkzaamheden bij diverse klanten geeft Marcel trainingen bij Info Support in het .NET curriculum. Verder is hij bij Info Support architect binnen het innovatie team van Professional Development Center waar de Info Support software ontwikkelstraat wordt gemaakt en beheerd. Vanuit deze rol is hij mede verantwoordelijk voor het opdoen van kennis over de nieuwste technologieën zodat deze kunnen worden toegepast in de ontwikkelstraat en verder kan worden uitgedragen binnen en buiten het bedrijf.