Bruggen bouwen met Visual Studio 2010

Hoe testers en ontwikkelaars beter samenwerken

Marcel de Vries

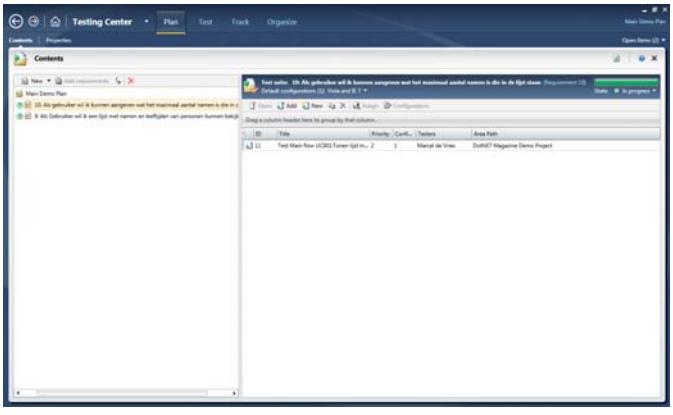
Unit testen is vandaag de dag een volledig ingeburgerd principe als het gaat om de kwaliteit en onderhoudbaarheid van software te verbeteren. Wat vaak onderbelicht blijft is echter het herhaalbaar en gestructureerd functioneel testen of de software wel voldoet aan de requirements die vooraf of gedurende het project zijn opgesteld in samenwerking met de klant.

Meestal is in een projectplan wel voorzien dat de eindgebruikers de applicatie uiteindelijk moeten accepteren door middel van een acceptatietest, maar helaas is dit maar al te vaak een sluitpost van de begroting. Laat staan dat er wordt geïnvesteerd in het herhaalbaar vastleggen van dit soort testen op een dusdanige manier dat deze direct weer herhaald kunnen worden zodra men in een later stadium nog een aanpassing maakt aan het systeem. Eigenlijk is dit is ook wel te verklaren, want als je dit goed wilt inregelen moet je eigenlijk een forse investering plegen in een testspecialist en bijbehorende tooling.
Deze testspecialist moet je vanaf dag één laten meelopen in je project en betrekken bij het opschrijven en afspreken van de specificaties. Hierdoor worden de specificaties ook goed getest en zijn de specificaties veel beter te testen. In de dagelijkse praktijk kom ik echter maar zeer weinig een testspecialist tegen die al vanaf de allereerste dag betrokken is bij het ontwikkeltraject. Een gevleugelde uitspraak die je vaak hoort is: "We hebben wel testers, namelijk de eindgebruikers nadat we hebben opgeleverd". Verder zie je ook dat projectleiders vaak worstelen aan het eind van het project om goed inzicht te krijgen wat nu de kwaliteit is van het systeem. Natuurlijk is dit herleidbaar op het weinig gestructureerd testen en het niet meten van de kwaliteit gedurende de realisatie van het project en dit wel achteraf willen weten.

Als een projectleider wel vanaf de allereerste dag gestructureerd testen wil oppakken dan wordt die vervolgens helaas maar al te vaak geconfronteerd met het feit dat de tooling vandaag de dag nog maar weinig ondersteuning biedt voor testen. Alleen bij de realisatie van zeer grote of bedrijfskritische systemen zie je dat er nog wel wordt geïnvesteerd in de aanschaf van tooling ter ondersteuning van de testers, maar voor de kleinere softwareprojecten is dit toch meestal niet weggelegd
Zou het niet mooi zijn als het mogelijk zou worden als de requirements op papier op een dusdanige alternatieve manier konden worden vastgelegd, dat we gedurende het project continue inzicht houden in de voortgang en kwaliteit die wordt gerealiseerd in het ontwikkelteam. Hierbij zou je dan willen dat de testspecialist en de ontwikkelaar op een efficiënte en effectieve manier kunnen samenwerken om naast snel bugs te vinden in requirements en de software, deze ook snel te kunnen reproduceren en op te lossen.
In dit artikel zal ik ingaan op welke manier Microsoft met de nieuwe Visual Studio Ultimate Editie juist probeert het gat tussen testers en ontwikkelaars te dichten en ervoor te zorgen dat een team op een effectieve manier aan de slag kan met functioneel testen en daarbij planmatig het testen kan inrichten.
Daarnaast zal ik laten zien op welke manier Microsoft de brug slaat tussen de niet-technische tester (ook wel de generalist tester genoemd) en de ontwikkelaars, zodat ontwikkelaars in een oogopslag kunnen zien wat de tester heeft gevonden en testers kunnen zien welke testen nodig zijn de codeaanpassingen te verifiëren. Aan de hand van een zeer eenvoudige applicatie zal ik proberen te laten zien hoe de dagelijkse praktijk van de testers en de ontwikkelaars drastisch kan verbeteren met de komst van deze nieuwe toolset.

Functioneel testen van requirements

Om op een juiste manier te kunnen testen is het allereerst van belang dat de eisen en wensen van de klant helder worden opgeschreven. Dit opschrijven kan iedereen natuurlijk doen op de ma-



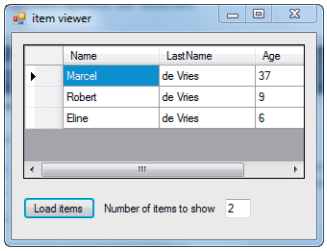
FIGUUR 1: MTLM TEST PLAN.

nier die hij zelf het prettigst vindt, maar door de requirements voortaan vast te leggen in de Team Foundation Server wordt het mogelijk deze requirements ook daadwerkelijk te gebruiken voor traceerbaarheid van bug, testen en change requests.
Dit vastleggen in de Team Foundation Server is mogelijk met behulp van work items. Work items zijn bedoeld voor het vastleggen van werk dat moet worden uitgevoerd of voor metainformatie over het te realiseren product of project. Een work item kan dus bijvoorbeeld een taak zijn voor het uitvoeren van bepaalde werkzaamheden(task work item), maar ook een user story of testcase, die respectievelijk de beschrijving bevat van een requirement of een testcase. Op zichzelf is het vastleggen van dit soort informatie niets nieuws in Visual Studio 2010, dit was al mogelijk in de eerdere versies van het product, echter niet vanuit een omgeving die specifiek voor een testspecialist is ingericht.
In Visual studio 2010 heeft men namelijk een nieuwe set aan testtools beschikbaar voor de testspecialist. Deze tools dragen de namen 'Microsoft Test en Lab Manager' (MTLM) en 'Microsoft Test Center' (MTC) en bieden een omgeving voor het plannen, beschrijven, uitvoeren en rapporteren van testen op basis van work items. In figuur 1 is een schermafdruk te zien van MTLM, waarbij het formulier open staat waarin je begint met het samenstellen van een testplan.

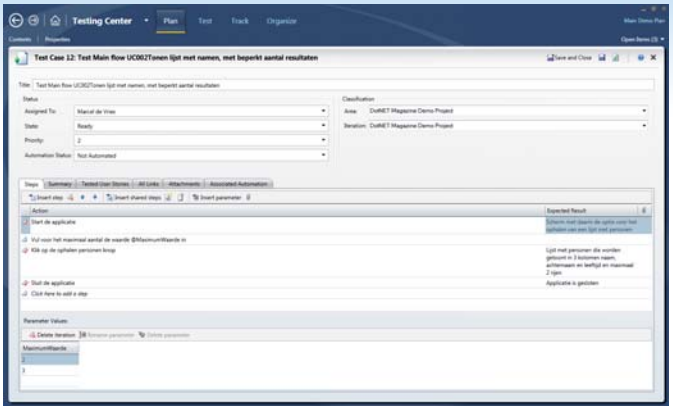
Alle requirements worden in de vorm van User Stories vastgelegd in de Team Foundation Server

In een testplan wordt feitelijk bepaald welke requirements we in een bepaalde periode willen gaan testen. Het uitgangspunt hierbij is dat alle requirements in de vorm van User Stories worden vastgelegd in de Team Foundation Server. Vervolgens worden aan requirements vervolgens testcases gekoppeld die de requirements op verschillende manieren testen. Het is ook mogelijk dit op basis van eigen work item typen te doen. In dit artikel ga ik er echter van uit dat we werken met de MSF agile work items, die standaard beschikbaar zijn in het product.

De applicatie die in dit artikel wordt gebruikt is bijzonder simpel. In figuur 2 is een schermafdruk te zien van de applicatie. De applicatie bevat een lijstje met namen van mensen die de gebruiker kan ophalen, waarbij die kan aangeven wat het maximaal aantal items is dat wordt getoond. De requirements zijn hiervoor opgeschreven in een tweetal user stories. Deze stories zijn vervolgens opgenomen in het testplan als de requirements en hieraan worden vervolgens testcases gekoppeld. In Figuur 3 is te zien dat het mogelijk is testcases aan te maken en waarin in detail wordt beschreven welke stappen moeten worden genomen bij het uitvoeren van



FIGUUR 2: VOORBEELD APPLICATIE.

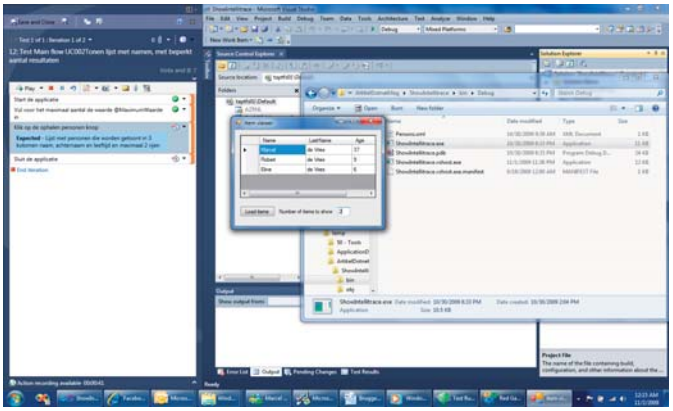


FIGUUR 3: TEST CASE BESCHRIJVING.

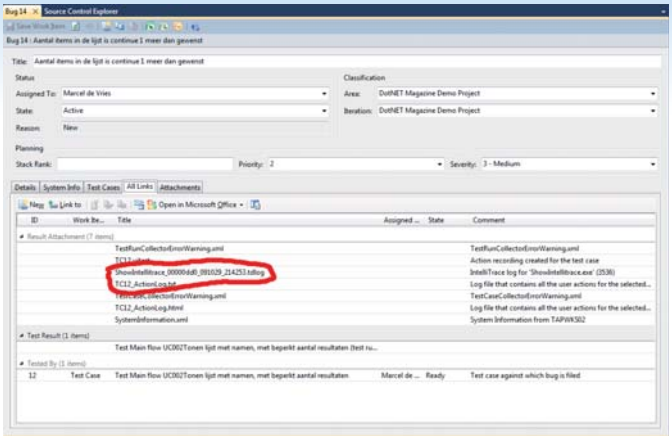
de testen. Bij iedere stap is het mogelijk een verwacht resultaat op te geven, hiermee wordt ook aangegeven dat de tester hier dan een 'ok' of 'niet ok' moet aangeven tijdens de testrun.

Bij iedere stap is het mogelijk gebruik te maken van parameters. Deze worden automatisch gedetecteerd zodra je een woord in de testaction met een @ teken voorziet. Vervolgens kun je daar waarden aan toekennen en deze zullen bij het uitvoeren van de testen resulteren in iteraties voor een betreffende stap. Ook is het mogelijk een set aan stappen op te slaan als een Shared Test Step en deze kun je dan in andere testen hergebruiken.
Als een testcase beschreven is en opgeslagen, dan kan deze in het testplan worden toegewezen aan een tester, die dan vervolgens in het MTLM testcenter, kan zien welke testen voor hem klaar staan om te worden uitgevoerd. Zodra een tester een test wil uitvoeren dan kiest hij voor de optie 'Run test' en vervolgens zal de MTLM omgeving zichzelf minimaliseren en wordt het test center actief in de linker hoek van het scherm (zie Figuur 4).

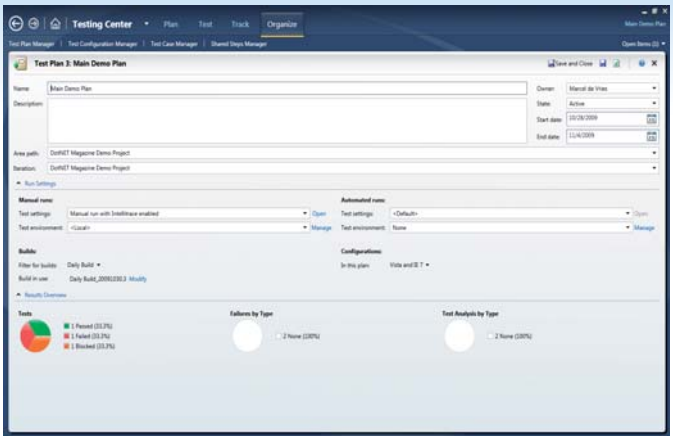
Vervolgens wordt de vraag gesteld of je ook wilt dat er een 'Action recording' wordt gemaakt van deze test. Als je hier het vinkje aanzet, dan zal van alle acties die worden uitgevoerd op het systeem worden vastgelegd in een action log, die vervolgens kan worden gebruikt om in een volgende testrun weer te gebruiken om de test geautomatiseerd te laten lopen. Ook is een action log te gebruiken voor het maken van een codedUItest, waar ik later op terug kom. Nadat de teststart worden alle stappen getoond die waren beschreven in de testcase. Vervolgens voer je iedere stap uit en geef je aan of deze stap een pass of fail was. In het geval dat de stap niet het verwachte resultaat oplevert, kan dit worden aangemerkt in de testrunner en ook worden vastgelegd door middel van een



FIGUUR 4: MTC IN ACTIE.



FIGUUR 5: STANDAARD ATTACHEMENTS BIJ AANMAKEN BUG.



FIGUUR 6: INZAGE IN DE STATUS VAN HET TESTPLAN.

work item. Hiervoor wordt het work item type Bug gebruikt. Met name op het punt van het vastleggen van een goed reproduceerbare Bug in de testomgeving is door Microsoft veel energie gestopt. Want iedere ontwikkelaar maakt het geregeld mee dat er een Bug is ingelegd in work item tracking en dat het uren duurt om het probleem te reproduceren. In heel veel gevallen is dit zelfs zo ingewikkeld, dat het niet mogelijk is de fout opnieuw aan het licht te brengen. Al met al kunnen dit soort Test, Analyseer, Reproduceer en fix cycli zeer veel tijd in beslag nemen. Met name om dit probleem op te lossen heeft Microsoft een nieuwe innovatie beschikbaar onder de naam IntelliTrace.

Intellitrace is een zogenaamde trace collector die tijdens de uitvoering van een test op de achtergrond, alle informatie met betrekking tot wat de applicatie onder test uitvoert vastlegt in een logfile. Microsoft heeft dit gerealiseerd door zogenaamde IL rewriting toe te passen vlak voordat de JIT compiler de code omzet naar machinecode. Hierbij wordt ervoor gezorgd dat er bij iedere aanroep van een functie en het resultaat van de functie, extra informatie wordt weggeschreven. De intellitrace collector is in staat ook voor specifieke .Net technologieën zoals: ADO.NET, ASP.NET, WCF, etcetera events te rapporteren met specifieke informatie die voor die technologie waardevolle informatie oplevert bij het opsporen van problemen. Denk bij ADO.NET aan een query string, of by ASP.NET aan een BeginProcessRequest event. De events die standaard worden opgenomen zijn o.a. zaken als registry toegang, File IO, ADO.NET database operaties, ASP.NET page cycle events, Windows Forms gestures(button click), Handled en Unhandled exceptions enzovoorts. Naast al deze events kan intellitrace ook alle parameters die tussen methode calls inspecteren en daarvan de data wegschrijven in de logfile. Let dan wel op, omdat er daardoor een potentieel zeer grote file van enkele Gigabites groot kan onstaan. Je kunt instellen wat Intellitrace aan standaard events en methode call logging moet uitvoeren. Dit kan in het options menu van Visual Studio.
In Figuur 5 is te zien dat er zeer veel informatie wordt opgeslagen zodra een Bug in MTLM wordt aangemaakt. De Intellitrace log file (*.Ttdlog) is de uiteindelijke logfile die alle detailinformatie bevat om direct zichtbaar te maken in de Visual studio IDE. De actionlog file is het bestand dat de automation stappen heeft weggeschreven, zodat de test opnieuw geautomatiseerd kan worden uigevoerd.

De tester heeft nu zijn werk erop zitten en sluit de MTLM omgeving af. Vervolgens kan hij zien in zijn testplan of er nog meer testen klaar staan voor hem om uit te voeren. Verder kan hij ook een overzicht krijgen van de resultaten van het testplan. Dit overzicht is weergegeven in Figuur 6.
We gaan nu van de testrol over naar de ontwikkelaar en kijken hoe hij verder kan met een zojuist door de tester geconstateerde Bug. In de Team Foundation Server kunnen we de Bug openen die de tester heeft aangemaakt en daarbij kunnen we de intellitrace logfile terugvinden. Vervolgens open je de logfile vanuit het work item.

Zodra we een debug sessie starten draait intellitrace op de achtergrond mee en slaat een logfile op van alle 'events' die we hebben aangegeven.

Na het openen van de logfile, krijgen we een overzicht te zien zoals in Figuur 7. In dit overzicht is ook direct terug te zien welke stappen de tester heeft uitgevoerd en die zijn onder het kopje 'Test Data' toegevoegd.

In de bugbeschrijving is duidelijk terug te vinden dat we de bug hebben aangemaakt op stap nummer 3 in de test, die overeen komt met stap 2 in de log. Vervolgens kunnen we dubbelklikken op deze stap en zal direct het intellitrace debug venster worden geopend waarin we de events kunnen zien die intellitrace voor ons heeft opgenomen. Zie Figuur 8 en Figuur 9.

N.B.: In Beta 2, zit nog een bug waardoor dubbelklik op een stap nog niet goed werkt. Klik op een thread en dit zorgt er voor dat de intelli Trace vensters worden geactiveerd.

Nu je de intellitrace vensters open hebt staan, kun je zien welke events zijn afgegaan gedurende de testrun. In het meest gunstige geval treed de Bug op, doordat er ergens een Exception optreed, bijvoorbeeld een File IO exception. Je ziet dan direct in de intellitrace event view, dat er een unhandled exception is opgetreden. Verder kun je daar ook zien wat de gebeurtenissen waren die vooraf zijn gegaan aan de unhandled exception. Bijvoorbeeld zaken zoals File IO of Registry access. Als je nu denkt dat een bepaald event de oorzaak is van het probleem, dan kun je door te dubbelklikken op dat event, direct naar de source code springen. De juiste code file wordt vervolgens geopend in de debugger. Zie Figuur 10.

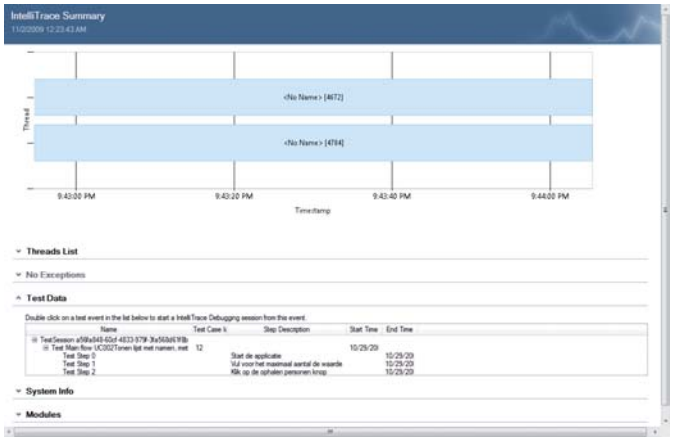
Vervolgens kun je dan in het debug window inspecteren wat de waarden van parameters waren op het moment dat de functie is aangeroepen. Intellitrace kan niet de lokale variabele waarden opslaan, maar de informatie over de functieaanroep inclusief zijn parameters, is normaal gesproken al meer dan voldoende informatie om het probleem op te sporen. Als je denkt dat niet deze functie maar een aanroepende functie het probleem is geweest, dan kun je ook via het calls view window of de dubbele pijltjes omhoog in het debugger window, verder terug in de tijd gaan naar de functies die deze functie hebben aangeroepen. Op deze manier is het dus mogelijk geworden eigenlijk als een soort videorecorder heen en weer te spoelen in de tijd en te inspecteren wat er op dat moment heeft afgespeeld in de code. Je kunt je waarschijnlijk wel voorstellen dat intellitrace zeer effectief is voor het opsporen van fouten, die door een tester zijn geconstateerd. Maar er is nog meer! Naast het werken met een logfile vanuit een bug, kan intellitrace een ontwikkelaar op nog een andere manier van dienst zijn.

We kennen allemaal wel het dagelijkse ritueel waarbij we een breakpoint plaatsen in de buurt van de plaats waar we verwachten dat de fout zit. Vervolgens starten we het programma, zorgen voor de input waarmee we verwachten de fout opnieuw te laten optreden, maar vervolgens zien we dat we net een regel te ver een breakpoint hebben gezet of dat deze op een plaats staat waarna we zelf nog een tiental keer een 'step over' uitvoeren om er vervolgens achter te komen dat net op die ene functie het een 'step into' had moeten zijn. Dit soort debug sessies kosten nodeloos veel tijd, omdat de debugger standaard alleen kan laten zien wat er in het huidige stackframe gebeurt. Met intellitrace is dit probleem ook verleden tijd. Op het moment dat we een debug sessie starten, zal intellitrace namelijk standaard direct op de achtergrond meedraaien en een logfile opslaan van alle 'events' die we hebben aangegeven interessant te zijn voor onze applicatie. Als we nu zien dat we per ongeluk een stap te ver zijn gegaan, kunnen we direct via het intellitrace window de functie selecteren die we zojuist hebben overgeslagen en direct zien wat er in die functie is gebeurd met de input en output.

Intellitrace kent op dit moment nog wel een aantal beperkingen. Zo is het niet mogelijk om een 64 bits applicatie direct in de IDE te debuggen met intellitrace. Je zult de applicatie daarvoor in de Any CPU mode moeten compileren, waarna het wel mogelijk is. Het is overigens wel mogelijk een trace log te laten aanmaken voor een 64 bits applicatie. Deze log is dan ook prima te gebruiken in Visual studio, zolang je daar maar terugstapt naar de Any Cpu configuratie. Verder zijn een aantal nieuwe features van de debugger ook nog niet opgenomen in de intellitrace. Denk hierbij aan het parallel Stacks en parallel tasks window. Deze zijn momenteel niet functioneel in intellitrace debug mode.
Op dit moment is ook nog niet bekend of intellitrace als zelfstandige distributie uit te leveren is voor het oplossen van niet reproduceerbare problemen op een productie omgeving. Een veel gevraagd scenario zal zijn dat je intellitrace kunt uitleveren samen met de eigen applicatie. Deze kun je dan activeren als er een verstoring optreed die moeilijk te reproduceren blijkt. Je kun dan tijdelijk de trace laten mee lopen en de klant de trace laten opsturen. Dit is zeker een scenario wat Microsoft in de toekomst mogelijk wil maken, echter dit is nog niet mogelijk met de huidige implementatie. Hopelijk zullen we daarover bij de lancering in maart meer horen.

Coded UI Testen

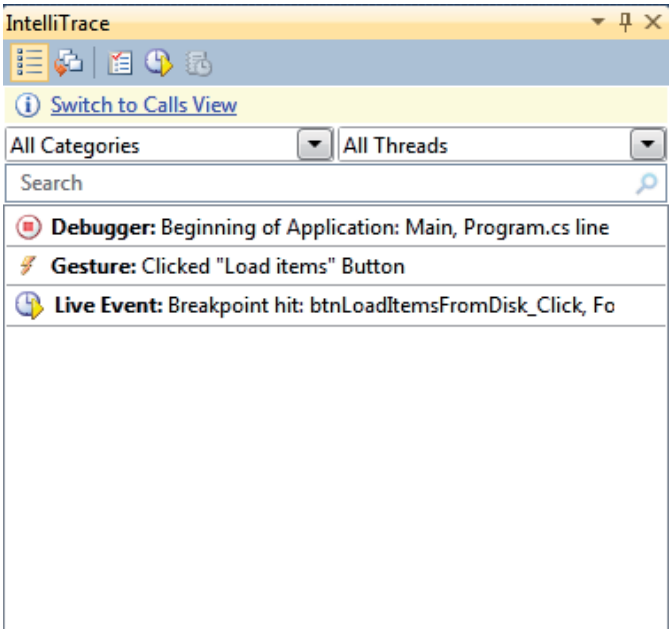
We hebben nu gezien dat we een requirement kunnen testen en dat we een gevonden bug eenvoudig kunnen reproduceren in de Visual Studio IDE. Zoals we vandaag de dag allemaal weten is het een best practice om voordat je een bug oplost, deze eerst aan te tonen met behulp van een unit test. Maar goed, dat is niet altijd



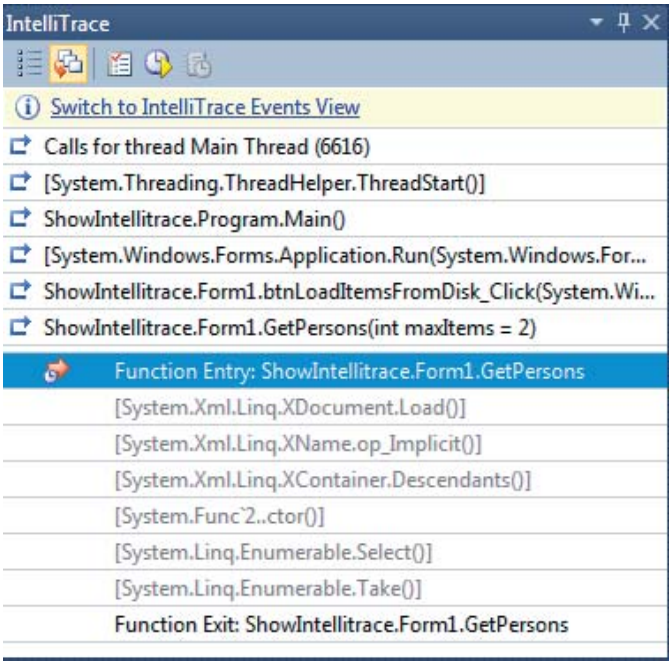
FIGUUR 7: INTELLITRACE OVERVIEW.

een even makkelijke opgave. Helaas komt het nog geregeld voor dat een ontwikkelaars toch nog een paar uur bezig is het probleem te reproduceren in een unit test, terwijl de bug fixen zelf maar een paar minuten werk zou zijn geweest.

Op dit punt kan een ander nieuw feature uit Visual Studio 2010 uitkomst bieden voor dit probleem. Visual Studio 2010 biedt namelijk vanaf de Premium editie de mogelijkheid een unit test te maken van het type CodedUItest. Een CodedUItest is een nieuw type test dat gebruik kan maken van de eerder opgenomen action log gedurende de testsessie (zie de optie voor het opnemen van de teststappen bij het uitvoeren van de test in MTLM). Zoals al te zien was in Figuur 5, is ook een action log opgenomen gedurende de test. Een dergelijke action log, kan met behulp van de 'Coded UI Test Builder' codegenerator, die onderdeel is van de Visual Studio IDE, direct een test worden gegenereerd die exact de stappen uitvoert die de tester ook heeft uitgevoerd. In de CodedUI-Test wordt een map file aangemaakt die de UI automation stappen bevat voor het naspelen van een volledige action log. Het is ook mogelijk een nieuwe recording aan te maken, dus het is niet altijd nodig dat de tester deze bestanden aanlevert.



FIGUUR 8:INTELLITRACE EVENT VIEW.



FIGUUR 9: INTELLITRACE CALLS VIEW.

Interessant punt is alleen wel dat alleen het naspelen van de acties van de tester geen informatie oplevert of de test nu wel of niet een fout aantoont. Om dit mogelijk te maken heeft men de optie toegevoegd voor het genereren van een Unit test assertion. Een assertion voor een codedUITest betreft het opzoeken van een control in de UI en vervolgens te kiezen welke conditie moet worden gecontroleerd. In Figuur 11 is te zien hoe met behulp van het 'Coded UI Test Builder' een control kan worden opgespoord en hiervoor een assertion wordt uitgegenereerd.

Vervolgens kunnen we nu een coded UI test op dezelfde manier als de standaard unit testen onderdeel laten uitmaken van de set aan testen die we iedere dag in de build willen laten meedraaien.

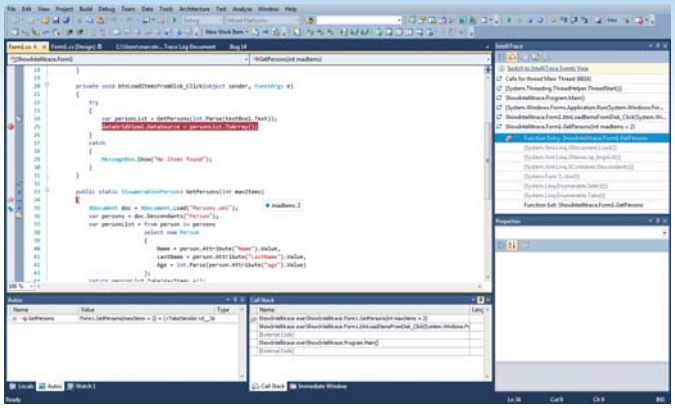
Test impact analysis

Stel je hebt uiteindelijk de applicatie van voor tot achter doorgetest en naast een groot aantal unit testen ook een groot arsenaal aan CodedUITesten en handmatige testen geschreven. Welke van deze testen moet ik dan uitvoeren om aan te tonen dat een aanpassing in code potentieel regressie bugs oplevert? Om op die vraag antwoord te krijgen heeft Microsoft een feature toegevoegd onder de naam 'Test Impact Analysis'.

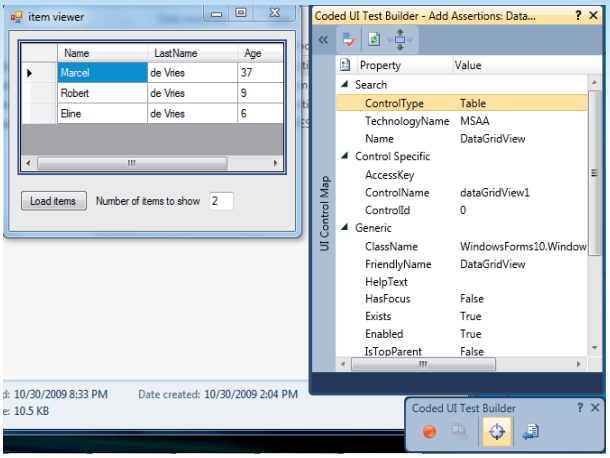
> Naspelen van de acties van de tester levert geen informatie op of de test wel of niet een fout aantoont

Test Impact Analysis maakt ook gebruik van een collector die gedurende de testen op de achtergrond informatie wegschrijft. Echter in dit geval schrijft de collector alleen informatie weg die relevant is voor het bepalen van test impact bij code aanpassingen. De weggeschreven log, dient vervolgens geregistreerd te worden bij een build die voor de applicatie is gedefinieerd. Nadat dit is gebeur, is het mogelijk om tussen twee builds van de applicatie, aan de team foundation server te vragen welke testen opnieuw moeten worden uitgevoerd, omdat ze op een of andere manier een relatie heeft met de aangepaste code.

Naast dat voor de ontwikkelaar test impact analysis kan aangeven welke (unit)testen hij opnieuw dient uit te voeren, heeft Microsoft ook een check in policy beschikbaar gemaakt, die je kunt gebruiken om de ontwikkelaar te verplichten alle aanbevolen testen tenminste opnieuw te draaien voordat de code wordt ingecheckt in versie beheer.
Ook de Generalist tester, kan in de MTLM omgeving bekijken welke testen worden aanbevolen. Hierbij is in het testplan aan te geven voor welke build de testrun wordt gepland. Vervolgens kan dan worden opgevraagd wat de lijst met aanbevolen testen is voor een betreffend plan. Op deze manier is dus niet alleen de ontwikkelaar, maar ook de tester voorzien van een minimale set aan testen die worden aanbevolen.



FIGUUR 10: INTELLITRACE GEÏNTEGREERD IN DEBUGER.



FIGUUR 11: CODED UI TEST BUILDER.

Conclusie

Op basis van de features die ik in dit artikel de revue heb laten passeren, kan wel worden gesteld dat de nieuwe versie van Visual Studio 2010 Ultimate Edition, zeker veel toegevoegde waarde zal gaan bieden bij het gestructureerd functioneel testen van onze applicaties. Microsoft is er goed in geslaagd op basis van integratie tussen de test- en ontwikkeldiscipline ervoor te zorgen dat de Bouw, test, Fix cycle die we dagelijks maken sterk kan worden geoptimaliseerd. Met behulp van Intellitrace, Test Automation en Test Impact Analysis zijn we in staat ervoor te zorgen dat testen niet meer een sluitpost van de projectbegroting zijn, maar dat we dit effectief kunnen inzetten om de kwaliteit van onze applicaties te verbeteren. Door ook de traceerbaarheid tussen de requirements en testen direct te regelen via work item tracking is men nu in staat in een project aan te tonen welke requirements ook daadwerkelijk getest zijn voordat we opleveren. Vanaf het moment dat we aan de slag gaan met Visual Studio 2010, wordt het mogelijk testen werkelijk een integraal onderdeel te maken van projecten en ervoor te zorgen dat de kwaliteit van de opgeleverde applicaties aanzienlijk verbeterd.

Marcel de Vries, is Technology Manager Microsoft bij InfoSupport. Tevens is hij Regional Director en MVP van Microsoft.