Continuous Integration helpt agile ontwikkelen

Minder rework, kosten en tijd besparen

Ewald Hofman

Steeds meer teams in de software development maken gebruik van een Agile methode bij het ontwikkelen van hun producten. Met de Agile aanpak maak je gebruik van multidisciplinaire teams en van korte cycli. Een van de mogelijkheden die je hebt om dit te ondersteunen is Continuous Integration. Dit artikel laat zien hoe je met gebruik van Team Foundation Server 2010 Continuous Integration kan inrichten.

Voor het werken in teamverband is een source control systeem een cruciaal instrument. De ontwikkelaar voert op zijn machine een wijziging uit op een kopie van de recente code base. Indien meerdere ontwikkelaars actief wijzigingen doorvoeren, is de kans groot dat één van je collega ontwikkelaars dezelfde source file heeft gewijzigd. Je lokale kopie is dus niet langer de recente revisie.

In het verleden gebruikte men hiervoor het exclusieve model, door het locken van een source file bij het uitchecken. Indien je collega actief is op een specifieke file, moet je dus wachten totdat de ander gereed is. Dit model is dus niet schaalbaar en is in populariteit vervangen door het parallelle model, waar iedereen gelijktijdig wijzigingen kan doorvoeren. Hoe langer een wijziging lokaal bij de ontwikkelaar staat, des te meer wijzigingen je collega's in de tussentijd hebben kunnen doorvoeren. Voordat een ontwikkelaar zijn wijzigingen kan inchecken zal hij er dus voor moeten zorgen dat alle recente wijzigingen die zijn ingecheckt na zijn snapshot moment ook zijn geïntegreerd in zijn lokale versie. Hoe langer een ontwikkelaar dus lokaal aan iets heeft gewerkt, des te meer integratiewerk hij kan verwachten. Het is zeker niet ongewoon dat de benodigde tijd om dit te doen vaak groter is dan de tijd die nodig was om zijn eigen change te maken. Het kan ook voorkomen dat een ontwikkelaar zijn werk volledig kan weggooien, omdat de structuur van de oplossing geheel anders is geworden.

Voor- en nadelen

Continuous integration is een practice gericht op het voorkomen van bovenstaande scenario's. Het doel is rework te reduceren en daarmee dus de kosten en tijd. Om Continuous Integration goed te kunnen inzetten, moet je een aantal uitgangspunten volgen:

- Alle ontwikkelaars integreren elke dag hun wijzigingen.
- Elke integratie start automatisch een build.
- De build moet snel zijn.
- De build voert een snelle validatie van de applicatie uit.

Als je een Continuous Integration opzet, dan heb je de volgende voordelen:

- Vroegtijdige waarschuwing over compilatie fouten en conflicterende wijzigingen.
- Alle unit tests worden meteen uitgevoerd.
- Als een unit test faalt of de build een bug bevat, kunnen ontwikkelaars terug naar een foutvrije versie van de source control zonder veel tijd kwijt te zijn met debugging.
- Doordat ontwikkelaars continu integratieproblemen ontdekken en oplossen, voorkomt het team een chaos aan het eind van de release.
- Er is altijd een werkende versie van de applicatie beschikbaar voor een test, een demo of een release.
- Ontwikkelaars krijgen direct feedback over de kwaliteit van de code die ze hebben geschreven.
- Het regelmatig inchecken van code dwingt het team modulaire, minder complexe code te maken.
- De metrieken over de code en de testen (code coverage, code complexity en aantal features gereed) zorgen ervoor dat ontwikkelaars zich concentreren op het ontwikkelen van functionele, kwalitatieve code.

Elke oplossing kent een andere kant van de medaille, en zo ook de Continuous Integration:

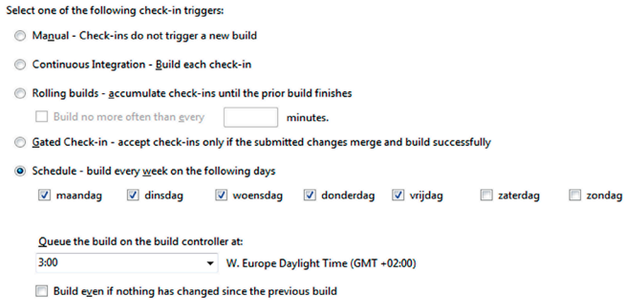
- Het opzetten van de Continuous Integration kost initieel tijd.
- Je hebt een goede test suite nodig voor het automatisch testen van de applicatie.
- Op grote schaal de code refactoren kan lastig zijn door de continu wijzigende code base.
- Hardware kosten voor de build omgeving kunnen significant zijn.

Veel teams die Continuous Integration hebben geadopteerd geven aan dat de voordelen tegen de nadelen opwegen. Het project bespaart zowel geld als tijd door de integratie bugs vroeg in de lifecycle te vinden en op te lossen.

Om Continuous Integration in een project goed te kunnen invoeren is software die een automatische build mogelijk maakt cruciaal. Sinds TFS 2008 is Continuous Integration een feature die out of the box wordt ondersteund door TFS.

Triggers

Een van de opties die je hebt bij het inrichten van de build is hoe een build wordt gestart. Figuur 1 toont de mogelijkheden die je hebt.



FIGUUR 1: DE TRIGGERS VAN EEN BUILD DEFINITIE.

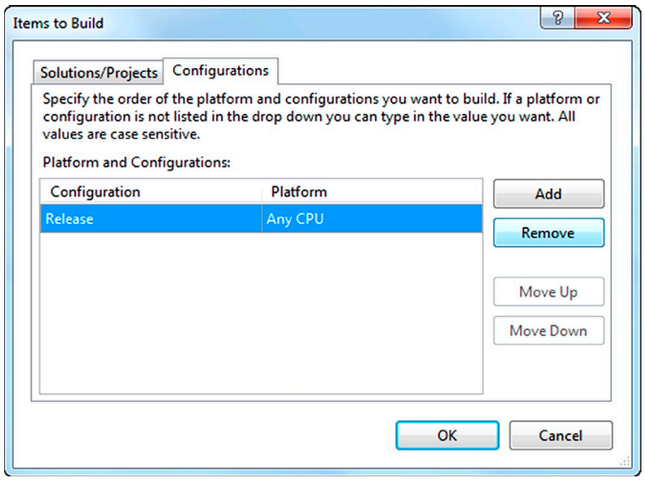
De optie Manual voert de build nooit automatisch uit. Je moet de build handmatig opstarten. Deze optie gebruik je voor builds die een deployment naar een omgeving uitvoeren, een zware automatische regressietest uitvoeren of een performance test uitvoeren.
De optie Schedule start de build op de gewenste dagen. Als je een groot aantal unit testen en/of geautomatiseerde test scenario's hebt, dan is het aan te raden deze grote set aan testen met deze trigger (bij voorkeur's nachts) uit te voeren. De andere opties zorgen er voor dat de build start als iemand een bestand incheckt. Dit type build moet niet te lang duren, zodat de build omgeving voldoende capaciteit heeft alle builds uit te voeren. Hierdoor krijgt de ontwikkelaar snel te horen of zijn check-in een geslaagde integratie is geweest.
Bij de Continuous Integration opties heb je drie mogelijkheden. De standaard Continuous Integration optie start bij elke check-in een build. Als een team meer check-ins uitvoert dan de build omgeving aan kan, waardoor de wachtrij van uit te voeren builds groeit, dan heb je de mogelijkheid van de Rolling Build. Deze trigger spaart de check-ins op totdat de vorige uitvoering van de build is afgerond. Deze trigger is goed te gebruiken als er sprake is van een grote hoeveelheid check-ins. Zo heb ik in de praktijk een team van 70 ontwikkelaars gezien die gezamenlijk meer dan 10 check-ins per uur verzorgden. Bij een build van 10 minuten kun je je voorstellen dat een Continuous Integration een enorme wachtrij veroorzaakt, welke je heel eenvoudig met een Rolling Build kunt oplossen. Het nadeel van een Rolling Build is dat een gefaalde build niet meer tot één enkele check-in is te herleiden.
De laatste trigger is de Gated Check-in. Bij deze trigger komen de wijzigingen van de check-in niet direct in Source Control te staan, maar in een shelveset. Pas als de build geslaagd is worden de wijzigingen van de shelveset in Source Control doorgevoerd. Het voordeel van deze trigger is dat de Source Control altijd code bevat die gevalideerd is door de build, en dus te compileren is en een smoke test heeft ondergaan.
Indien iemand een integratiefout veroorzaakt, wordt de rest van het team niet met deze integratiefouten geconfronteerd. Het nadeel van deze trigger is dat de wijzigingen niet direct in Source Control terecht komen en dus voor een andere ervaring voor de ontwikkelaar zorgen.
Indien je de Gated Check-in wilt introduceren in je team, dan is het raadzaam het team te informeren en te trainen.

Inrichten van de build

Als je een build aanmaakt, definieer je bij Process welke stappen de build moet uitvoeren. De build in TFS 2010 is gebaseerd op Windows Workflow. Microsoft levert bij de installatie van TFS drie Workflow templates waaruit je kunt kiezen. De eerste template is de DefaultTemplate welke de applicatie compileert en de testen uitvoert. Daarnaast bevat het de UpgradeTemplate waarmee je de TFS 2008 build scripts (TFSBuild.proj) kunt uitvoeren. De laatste template (LabDefaultTemplate) gebruik je als de build geautomatiseerde test scripts op een Lab Management omgeving moet uitvoeren. De LabDefaultTemplate gebruik je alleen als je tijdens de build testen wilt uitvoeren die een omgeving nodig hebben. De UpgradeTemplate gebruik je als je de oude TFS 2008 build scripts wilt hergebruiken. In alle andere gevallen gebruik je de DefaultTemplate. De DefaultTemplate geeft je standaard de mogelijkheid om Code Analysis uit te voeren, de integratie met Symbol Server te verzorgen, de testen uit te voeren en de testimpact te bepalen.

Configuraties

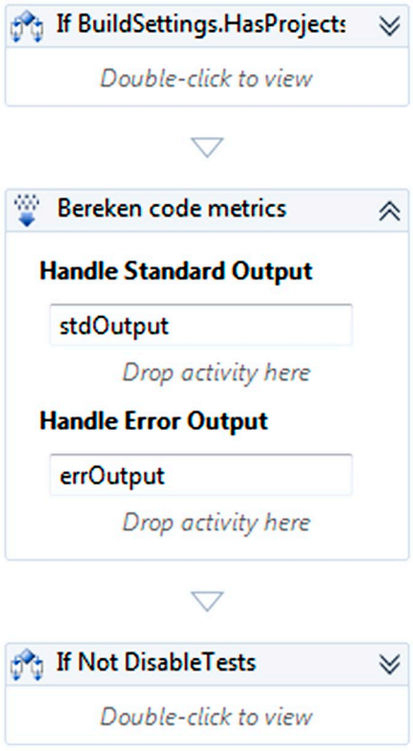
Als je het proces inricht geef je op welke solutions en/of projecten de build moet compileren. Je kunt daarnaast ook opgeven in welke configuraties je applicatie wordt gecompileerd. Standaard is deze lijst leeg, wat betekent dat de applicatie in de standaard configuratie wordt gebouwd, waarbij de configuratie die je actief hebt in Visual Studio de standaard configuratie is. Deze actieve configuratie is vaak de Debug configuratie, terwijl het niet wenselijk is de applicatie in de Debug configuratie uit te leveren. Voeg dus altijd de Release configuratie toe aan deze lijst zodat je gebruik maakt van de optimalisaties van de Release configuratie.



FIGUUR 2: DE CONFIGURATIES WAARIN JE APPLICATIE WORDT GECOMPILEERD.

Code Analysis

Code Analysis controleert de kwaliteit van de applicatie door de code te scannen op mogelijke problemen. Er zijn regels op het gebied van Performance, Security, Design, Maintainability en andere. In Visual Studio 2010 definieer je welke Code Analysis regels het moet uitvoeren in zogenaamde Rule Sets. In de build definitie geef je aan of de Code Analysis moet worden uitgevoerd, waarbij je kunt kiezen tussen Never, Always of As Configured. De waardes Never en Always spreken voor zich. Het nadeel van de Always



FIGUUR 3: CODE METRICS TOEVOEGEN AAN DE BUILD.

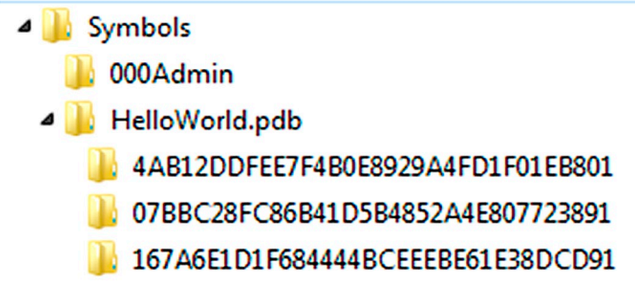
optie is dat de Code Analysis ook wordt bepaald van de test projecten, terwijl je hier helemaal niet in geïnteresseerd bent. Bij de waarde As Configured gebruikt de build de instelling van het project. Het advies is om deze optie te gebruiken.
In de vorige paragraaf heb je gezien dat de build in de Release configuratie wordt gecompileerd. De Code Analysis is configuratie afhankelijk, zodat je kunt instellen dat de Debug configuratie de Code Analysis niet uitvoert en de Release configuratie wel. Hiermee bereik je dat de ontwikkelaars lokaal geen performance impact hebben van de Code Analysis, terwijl de build server alleen de projecten controleert die je daadwerkelijk uitlevert.

Code Metrics

Sinds Visual Studio 2008 is de mogelijkheid aanwezig de code metrics te berekenen. Deze optie was altijd alleen beschikbaar in Visual Studio zelf en niet te bepalen op de build server. Sinds kort heeft Microsoft een command line tool vrijgegeven (zie referenties) waarmee de code metrics kunnen worden bepaald. Het is heel goed mogelijk deze activiteit aan je build toe te voegen. Als je dit wilt doen, dan moet je de Build Process Template aanpassen. Elk team project heeft in Source Control een map BuildProcessTemplates waarin de Build Process Templates staan. Open de DefaultTemplate.xaml in de editor door er op te dubbelklikken en scroll naar beneden tot je ziet "If Not DisableTests". Sleep boven deze activiteit een InvokeProcess activiteit, welke je kunt vinden in de toolbox. Selecteer de activiteit en vul bij de FileName "%PROGRAMFILES(x86)%\Microsoft Visual Studio 10.0\Team Tools\Static Analysis Tools\FxCop\Metrics.exe" in. Bij de Arguments geef je op welke assembly wilt analyseren en waar het resultaat naar toe moet: "/f:" + IO.Path.Combine(BinariesRoot, "MyAssembly.dll") + " /o:" + IO.Path.Combine(BinariesRoot, "MyAssemblyMetrics.xml")

Symbol Server

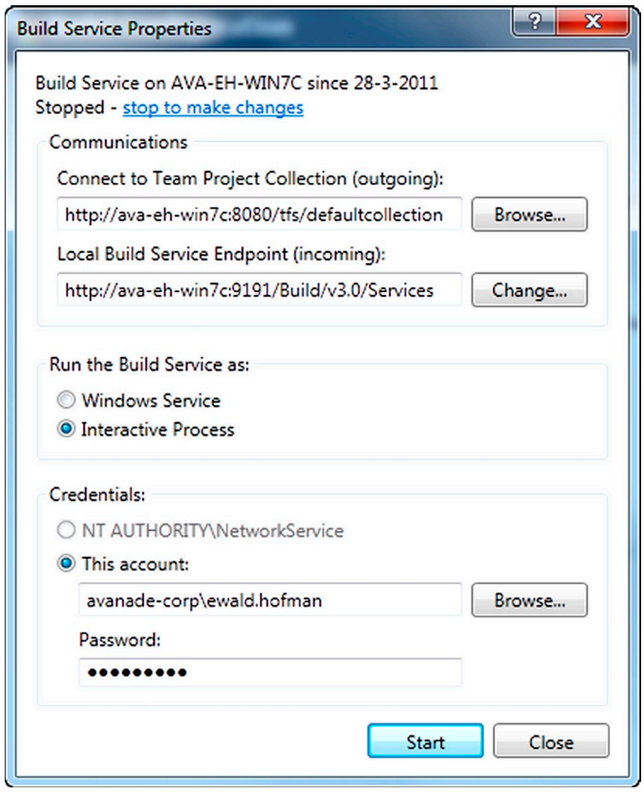
De derde standaard activiteit is het ondersteunen van de Symbol Server. Een Symbol Server is een file share waar de pdb's van alle gebouwde bestanden worden opgeslagen. Een pdb bevat alle debug informatie, welke nodig is voor het debuggen in Visual Studio aan je applicatie, het uitlezen van een dump file en het uitlezen van een IntelliTrace log bestand. Elke gecompileerde dll heeft een unieke Id. De Symbol Server slaat de pdb op in de map waarvan de naam gelijk is aan deze id. Op deze manier kan Visual Studio de juiste pdb vinden bij de assembly die je wilt debuggen, zonder dat je de pdb uitlevert. Het inrichten van de Symbol Server is niets anders dan het invullen van de file share in de build definitie. Het geeft je de mogelijkheid je applicatie zonder pdb uit te leveren en toch te kunnen debuggen als de klant een Dump of een IntelliTrace log aanlevert.



FIGUUR 4: DE INHOUD VAN DE SYMBOL SERVER FILE SHARE.

Testen

Het volgende wat je kunt toevoegen aan de build zijn de testen. Als je een nieuwe build definitie maakt, is al standaard ingesteld dat de testen worden uitgevoerd. Er zijn twee type testen die je kunt uitvoeren: de unit test en de CodedUI test. Een unit test va-



FIGUUR 5: DE BUILD SERVICE ALS EEN INTERACTIEF PROCES CONFIGUREREN.

lideert een stuk functionaliteit van de applicatie en dit is prima in de build uit te voeren. Echter een CodedUI test voert een scenario uit op de User Interface. Omdat de het proces wat de build uitvoert een Windows Service is en deze niet met de User Interface kan interacteren, moet je extra stappen ondernemen om dit te ondersteunen. Je hebt hier twee opties. De eerste optie is om van de build een Interactive Process te maken. Je kunt deze optie vinden via de Team Foundation Administration Console. Als je dit doet, dan wordt de TFSBuildServiceHost als een console applicatie gestart. Het nadeel van deze oplossing is dat je op de build server moet zijn ingelogd en dat je heel eenvoudig met het sluiten van de console applicatie de build service stopt.
De tweede optie is gebruik te maken van de LabDefaultTemplate

Team Build 2010 biedt agile teams goede mogelijkheden om validaties van integraties uit te voeren.

die we eerder hebben besproken. In deze template geef je op welke omgeving uit Lab Management de build gebruikt bij de uitvoeren van de CodedUI tests. Als je geïnteresseerd bent in meer informatie over dit onderwerp, dan verwijs ik je naar het artikel over Lab Management elders in het .NET magazine.

Advies

Bij het inrichten van een Continuous Integration zijn er veel keuzes te maken:

- Welke build type(s) moet ik gebruiken?
- Welke Code Analysis set moet ik uitvoeren in de build?
- Welke test suite moet de build uitvoeren, en tijdens welke build type?
- Welke build types voeren de CodedUI tests uit?
- Hoe snel moet een build zijn om effectief feedback te leveren aan een ontwikkelaar?

Als ik een Continuous Integration build inricht, dan maak ik twee builds: een Continuous Integration build en een Nightly build. Welke trigger ik gebruik voor de Continuous Integration ligt aan de situatie. Ik gebruik bij voorkeur de Gated Checkin trigger, zodat de code base altijd gevalideerde source code bevat. Deze build bevat het compileren van de code en het uitvoeren van een de meest belangrijke unit tests.
Als er een omgeving beschikbaar is waarop de build de applicatie kan installeren, dan laat ik de build ook de meest belangrijke gebruiker interacties testen met een CodedUI test. Omdat een Gated Checkin build interactie met de ontwikkelaar vraagt, moet het niet te lang duren voor de ontwikkelaar zijn feedback krijgt. De doorlooptijd van een build is opgebouwd uit de wachttijd voordat een build aan de beurt is en de uitvoertijd van de build zelf. Is de doorlooptijd van een Gated Checkin build meer dan tien minuten dan overweeg ik samen met het team een aantal opties. Als er een te lange wachtrij is, dan stel ik voor om de capaciteit van de build omgeving te vergroten door meerdere agents te configureren. Als de uitvoertijd een probleem is, dan probeer ik de build versnellen door onder andere een Incremental Build te overwegen waarbij alleen de bestanden overgehaald die gewijzigd zijn en hoeft de build alleen die projecten te compileren die gewijzigd zijn. Indien dit niet haalbaar is, is de Gated Checkin niet de beste optie als trigger voor een Continuous Integration build. Ik pas dan de trigger aan naar de Rolling Build (lange wachtrij) of de Continuous Integration (lange uitvoertijd).

Als een team niet met een Gated Checkin kan werken, dan biedt TFS 2010 de mogelijkheid een Private Build (ook wel een Buddy Build genoemd) uit te voeren. Deze build gebruikt, net zoals de Gated Checkin, een shelveset bij het uitvoeren, maar de ontwikkelaar start deze build zelf. Met een Private Build valideert een ontwikkelaar van zijn gemaakte wijzigingen voordat hij/zij de code incheckt. Dit build type is heel goed te gebruiken als een ontwikkelaar grote wijzigingen heeft uitgevoerd of de code flinke heeft geherstructureerd.
De tweede build schedule ik voor de nacht. Omdat de build 's nachts uit wordt gevoerd is de wachtrij en de uitvoertijd niet van belang. In deze build stop ik alles wat het team uitgevoerd wilt hebben, zoals:

- Het compileren van alle projecten.
- Het uitvoeren van Code Analysis.
- Het bepalen van de Code Metrics.
- Het uitvoeren van alle unit testen.
- Het packagen van de applicatie zodat het eenvoudig gedistribueerd kan worden.
- Een automatische installatie in de test omgeving.
- Het uitvoeren van alle geautomatiseerde testgevallen (CodedUI).

In de praktijk heb ik al menigmaal meegemaakt dat deze nachtelijke build meer dan één uur duurt. Bij Microsoft duurt de nachtelijke build voor het Visual Studio product zelfs meer dan twaalf uur!

Conclusie

Agile teams hebben behoefte aan vele integratie momenten en de validatie van deze integraties. Team Build biedt deze teams uitstekende mogelijkheden deze validaties uit te voeren. Doordat Team Build 2010 werkt met Windows Workflow is het mogelijk één Build Process Template te definiëren voor de verschillende teams in je organisatie. Deze template is geheel naar eigen wens in te vullen. Dit artikel heeft zich niet verdiept in hoe je de template kunt aanpassen met extra maatwerk activiteiten. Hiervoor verwijs ik door naar mijn blog die je kunt vinden bij de referenties.

Referenties

- Serie posts over het aanpassen van de build: http://www.ewaldhofman.nl/?tag=/build+2010+customization
- Codeplex project met build activities: http://tfsbuildextensions.codeplex.com/
- Code metrics command line tool: http://www.microsoft.com/downloads/en/details.aspx?FamilyID=edd1dfb0-b9fe-4e90-b6a6-5ed6f6f6e615
- Private build: http://msdn.microsoft.com/en-us/library/ms181722.aspx#queue_private

Ewald Hofman, is Solution Developer bij Avanade. Hij is tevens MVP op het gebied van Visual Studio ALM. Hij blogt regelmatig over Team Foundatuion Server op http://www.ewaldhofman.nl