Robert van der Kleij
is technisch directeur bij ETX. Hij is de lead-designer van de BuildMonkey.NET, die een onderdeel vormt van de ETX SoftwareStraat.Net. Voor vragen of opmerkingen kun je mailen naar robertk@etx.nl

Automatisch buildproces: keuzes en alternatieven

SNEL EN EFFECTIEF EEN BUILDPROCES OPZETTEN MET VISUAL STUDIO.NET

Dit artikel bespreekt hoe je snel en concreet een buildproces met Microsoft Visual Studio.NET kunt opzetten. Daarbij komen vragen aan bod als hoe je een buildproces kunt inrichten voor complexere projecten en welke uitdagingen moeten worden overwonnen bij de invoering van een buildproces. Het artikel gaat ervan uit dat de basisbegrippen van Microsoft Visual SourceSafe en Visual Studio.NET 2003 bekend zijn. De nachtelijke automatische build is het eerste onderdeel dat hierbij uitvoerig uit de doeken wordt gedaan.

Het ontwikkelen van software in teamverband is geen eenvoudige taak. Hoe kun je garanderen dat de software goed samenwerkt en dat bij de integratie, die meestal aan het eind van een traject plaatsvindt, geen grote fouten meer naar boven komen. Een goed buildproces zorgt ervoor dat dit soort problemen eerder worden gesignaleerd. Hiermee wordt de overstap van de ontwikkelfase naar een integratietest en de uiteindelijke oplevering kleiner. Bovendien kan met een automatische build op een herhaalbare manier software worden gebouwd en biedt een automatische build een kapstok om tests en controles op de code uit te voeren.

's Nachts bouwen

Een automatische build van software vindt meestal 's nachts plaats. Overdag zouden ontwikkelaars last kunnen hebben van het bouwen, omdat bijvoorbeeld SourceSafe intensief wordt gebruikt tijdens een build. Een veel belangrijker argument dan eventuele overlast voor de ontwikkelaar, is de kwaliteit van de build zelf. Op het moment van bouwen moet bij voorkeur een stabiele versie uit SourceSafe kunnen worden opgehaald. 's Nachts is zo'n versie beschikbaar, indien iedere ontwikkelaar aan het eind van de dag standaard de compileerbare code incheckt. De kans op een succesvolle build wordt daarmee aanzienlijk vergroot. Om de build 's nachts te laten uitvoeren is een script of applicatie nodig die het bouwproces coördineert. Het kernonderdeel daarbij is het compileren van de projecten. De eenvoudigste methode om projecten te compileren is via de commandline-interface van Visual Studio. Door de naam van een solution-file en een configuratie op te geven wordt hetzelfde bouwproces opgestart als via de IDE-interface. Het resultaat van de build kan via de out-parameter worden opgeslagen in een tekstbestand; zie codevoorbeeld 1.

```
devenv.exe    TestProject.sln    /rebuild Debug    /out Build.txt
```

Codevoorbeeld 1

Voor eenvoudige projecten is dit een goede en efficiënte oplossing. Helaas is de invloed die via de commandline op het compileerproces kan worden uitgeoefend niet zo groot. Deze aanpak kan tot problemen leiden indien er bijvoorbeeld binnen het team verschillende solution-files worden gebruikt, of indien de configuratie van de buildserver afwijkt van die op de ontwikkelmachines. Een postbuild-actie die een bestand kopieert kan ideaal zijn op een ontwikkelmachine, maar er juist op een buildserver voor zorgen dat de build mislukt. Een totaal andere aanpak om de code te bouwen is het gebruik van CodeProvider-classes. Voor talen als C# en VB.NET worden in het .NET Framework compiler-classes meegeleverd die te integreren zijn in een eigen buildapplicatie. Deze aanpak brengt met zich mee dat je Visual Studio niet meer hoeft, maar ook niet meer kunt gebruiken als compiler. In codevoorbeeld 2 wordt getoond hoe met deze compilerOclasses binnen het .NET Framework een eenvoudig C#-project kan worden gecompileerd.

```
public void Build()
{
   CodeDomProvider codeProvider = new CSharpCodeProvider();
   ICodeCompiler compiler = codeProvider.CreateCompiler();
   CompilerParameters params = new CompilerParameters();

   params.GenerateExecutable = true;
   params.OutputAssembly = @"c:\Projects\test.exe";

   params.CompilerOptions = "/target:winexe /doc:Test.xml";
   params.IncludeDebugInformation = true;
   params.ReferencedAssemblies.Add("System.dll");

   string [] files =
new string [2] {"test.cs", "AssemblyInfo.cs"};

CompilerResults results =
compiler.CompileAssemblyFromFileBatch(params, files);

   if (results.Errors.Count > 0)
   {
       // Catch Errors
   }
}
```

Codevoorbeeld 2

```
string localPath = @"C:\Projects\";

SourceSafeTypeLib.VSSDatabase vssDatabase
      = new SourceSafeTypeLib.VSSDatabase();

// Geen username en password nodig
vssDatabase.Open(@"\\System\SourceSafe\srcsafe.ini","", "" );
SourceSafeTypeLib.VSSItem aItem =
   m_VSSDatabase.get_VSSItem("$/Projects", false);

// Alle files recursief ophalen, vervangen
aItem.Get ( ref localPath,
SourceSafeTypeLib.VSSFlags.VSSFLAG_RECURSYES +
  SourceSafeTypeLib.VSSFlags.VSSFLAG_CMPFAIL +
SourceSafeTypeLib.VSSFlags.VSSFLAG_GETYES +
SourceSafeTypeLib.VSSFlags.VSSFLAG_DELYES +
SourceSafeTypeLib.VSSFlags.VSSFLAG_REPREPLACE);
```

Codevoorbeeld 3

Met de compiler-classes is het mogelijk elke stap van het compileerproces te beïnvloeden. De prijs die je voor deze flexibiliteit betaalt, is de hoeveelheid werk die je moet uitvoeren voor een goed werkend buildproces. Project- en solution-files zijn immers onderdelen van Visual Studio.NET en door geen solution-files te gebruiken moet je veel meer zelf doen. Je moet dan zelf Je moet zelf bepalen welke files moeten worden meegecompileerd in projecten, in welke compileervolgorde en met welke compileersettings. Iedere developer weet dat dit ontzettend veel werk met zich meebrengt. Gebruik van dit soort classes is dan ook alleen efficiënt als de buildapplicatie gebruikt gaat worden voor alle software in het bedrijf; of in gespecialiseerde buildapplicaties die bijvoorbeeld in staat moeten zijn om zowel .NET-code als VB6-code te kunnen bouwen en integreren.

Visual SourceSafe gebruiken

Om te zorgen dat de build altijd uitgaat van de laatste actuele code, is integratie met de gebruikte versiebeheertool zoals Visual SourceSafe, een belangrijke tweede stap. Voor het daadwerkelijke compileerproces begint, moet de laatste code worden opgehaald. Indien de folderstructuur van SourceSafe overeenkomt met de opdeling van de solutions, dan kan per solution een recursieve 'get' op een hoofdfolder in SourceSafe voldoende zijn. Visual SourceSafe biedt een commandline-tool en een COM-interface om, buiten de SourceSafe Explorer om, acties uit te voeren op bestanden in SourceSafe. In codevoorbeeld 3 wordt getoond hoe COM-interfacebestanden uit SourceSafe kunnen worden opgehaald. Om te zorgen dat de gebruikte versie van de build later eenvoudig is terug te halen uit SourceSafe, kun je voor het 'getten' een label plaatsen op de folder in SourceSafe. Door het label te gebruiken bij de 'get' weet je zeker welke afzonderlijke file-versies worden gebruikt; hoewel je er meestal niet bang voor hoeft te zijn dat iemand 's nachts files aan het inchecken is.

Bij het maken van een nieuwe build, wordt de build van de nacht ervoor steeds overschreven. Het bewaren van oude builds is dan ook zeer verstandig, zeker omdat het aan de start van een nieuwe build nog niet duidelijk is of deze gaat lukken. Indien builds zorgvuldig worden bewaard, is het altijd mogelijk de laatst werkende software te installeren of om een specifieke build in te zetten voor testdoeleinden. Bij het ophalen uit SourceSafe mogen oude resultaten dan ook niet zomaar worden weggegooid. Dit is te realiseren door de resultaten aan het einde van een build te kopiëren naar een directory met een specifiek buildnummer in de naam. Bij het ophalen van de bestanden uit SourceSafe kan het beste dezelfde directorystructuur worden gebruikt als op de ontwikkelpc's. Als

een projectfile bijvoorbeeld een assemblyfile-referentie bevat naar "C:\projects\Test.dll", dan moet het nachtelijke buildproces dezelfde directorystructuur op de buildserver gebruiken.
De omgekeerde aanpak is ook mogelijk: de code wordt dan opgehaald uit SourceSafe, overgezet naar een specifieke builddirectory en daar gebouwd. Dit alternatief zou ik echter niet aanraden, omdat er meestal problemen optreden. Het nachtelijk buildproces detecteert dan onmiddellijk een configuratiefout, omdat C:\Projects\Test.dll niet te vinden is. Zo'n configuratiefout kost doorgaans veel tijd om te vinden en te herstellen en gedurende het zoeken is er geen nieuwe testversie beschikbaar. Je kunt deze nadelen overigens wel omzeilen door absolute verwijzingen in solution- en projectfiles te vermijden.

Ophogen buildnummer

Als het bouwen van code bedoeld is als een herhaalbaar proces, dan moet een build ook kunnen worden geïdentificeerd. Als een assembly later op een test- of productieserver staat, moet er een duidelijke relatie bestaan tussen het versienummer en het buildnummer. In .NET is het versienummer een onderdeel van de assembly. Dit nummer staat meestal vermeld in de AssemblyInfo-file van het project en wordt door Visual Studio automatisch opgehoogd; zie codevoorbeeld 4.

Helaas is de automatische nummering van Visual Studio niet afdoende. Deze wordt namelijk per sessie van Visual Studio opgehoogd, terwijl je tijdens een nachtelijk buildproces een oplopend nummer per nacht wilt gebruiken. Dit probleem kan worden opgelost door het AssemblyVersion-attribuut in een aparte file te zetten en deze voor alle projecten in de solution te 'sharen' in SourceSafe. Als iemand dit bij een nieuw project vergeet toe te passen, heb je meteen een probleem. Het versienummer komt niet meer overeen. Beter is het een search en replace uit te voeren op alle files in de projecten met behulp van een reguliere expressie. Daarbij kan dan worden gezocht naar het AssemblyVersion-attribuut, om deze vervolgens te vervangen door het specifieke buildnummer. Door ook in de oorspronkelijke source-code de versie standaard op 1.0.0.0 te zetten, is later altijd te achterhalen of de assembly via een nachtelijke build is opgeleverd of door een ontwikkelaar op zijn eigen pc.

```
[assembly: AssemblyVersion("1.0.*")]
```

Codevoorbeeld 4

Signen van assemblies

Een nachtelijke build is een ideale gelegenheid om allerlei controle- en registratietaken uit te laten voeren, die veel tijd zouden kosten als een ontwikkelaar deze op zich zou nemen. Een goed voorbeeld is het signen van assemblies met de veilig bewaarde sign-key die de private key van het bedrijf bevat. Een sign-key wil je niet laten rondslingeren op elke ontwikkelwerkplek. De sign-key is veel beter te bewaken in een gecontroleerde omgeving op één centrale buildserver. In combinatie met het versienummer kan men aan de hand van een assembly altijd vaststellen of deze assembly officieel afkomstig is van de producerende organisatie. Bovendien kan men ook nog bepalen uit welke specifieke build het bestand komt. Voor het signen kan gebruik worden gemaakt van delayed signing bij het ontwikkelen. Na het compileren kan het buildscript de assemblies signen met de tool sn.exe. Een alternatief voor delayed signing is eenzelfde search- en replace-strategie te gebruiken als bij de versienummers. Het voordeel is dat de assemblies op de ontwikkelwerkplekken ook kunnen worden gesigned met een tijdelijke key, zodat de ontwikkelaar ook security- of GAC-constructies kan testen.

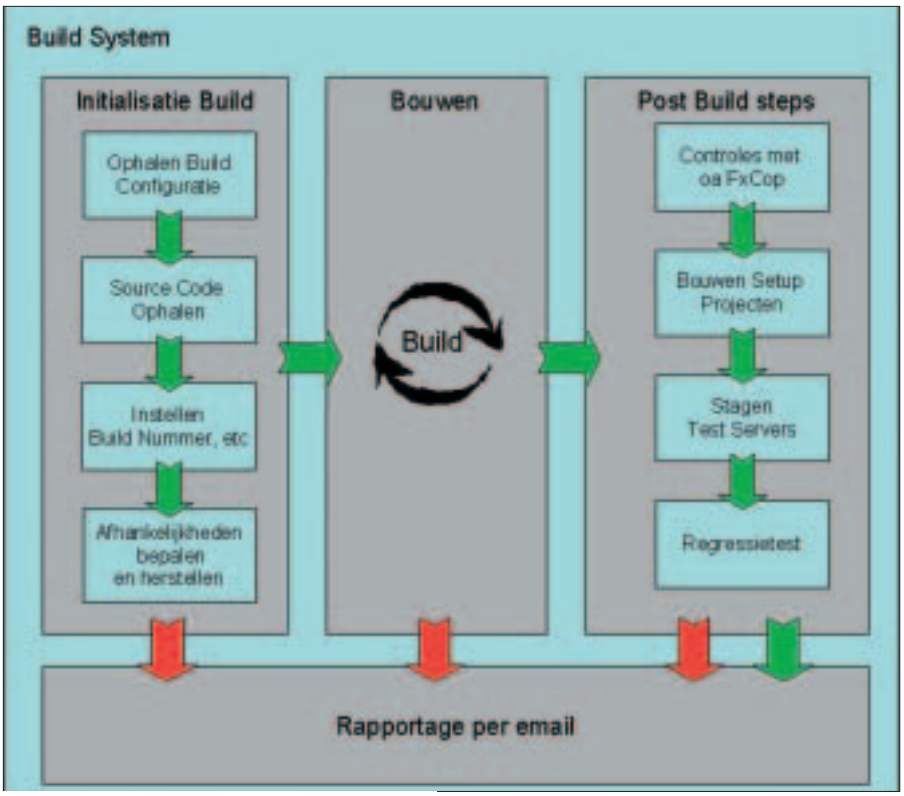
Detecteren en herstellen

Of en in hoeverre een buildproces streng moet controleren op afgesproken regels is een belangrijke keuze bij de inrichting van zo'n proces. Moet een build bijvoorbeeld stoppen, als blijkt dat een van de projecten een andere directorystructuur gebruikt dan volgens de standaard is afgesproken? Met een streng controleproces kun je problemen vermijden, bijvoorbeeld met configuratiefouten, maar je beschikt niet over software om mee te testen. Bij een vergevend proces is die software er wel, maar deze bevat mogelijk nog steeds configuratiefouten die pas op een later, en dus vervelend tijdstip naar boven komen drijven. Naar mijn mening is het primaire doel van een buildproces om sneller en op een herhaalbare manier tot betere software te komen. Configuratiefouten of onjuiste verwijzingen in een projectfile zouden wel moeten worden gedetecteerd, maar een goed buildproces moet daar mijns inziens mee kunnen omgaan en de fouten kunnen herstellen. Als een Visual Studio-project bijvoorbeeld verwijst naar een assembly 'Test.dll' met versienummer 1.2.3.4, maar er is alleen een versie 1.2.3.5 met een andere public key, dan zou een ideaal buildproces dat moeten detecteren en herstellen voordat er gecompileerd gaat worden. Dit uitgangspunt brengt echter veel werk met zich mee en wijst in de richting van het scenario met de compiler-classes, waarbij projectfiles zelf uitgelezen moeten worden. Toch is detecteren en herstellen een goede stelling om als uitgangspunt te nemen. Als op een eenvoudige manier kan worden gezorgd dat er meer succesvolle builds zijn, kun je sneller fouten in de software zelf detecteren en leidt dat uiteindelijk tot een hogere kwaliteit.

Inrichten buildscript

Tot nu toe zijn er voorbeelden uitgewerkt van onderdelen die tijdens een build worden uitgevoerd. Elke automatische build volgt een soortgelijk patroon. Afbeelding 1 toont welke stappen kunnen worden uitgevoerd in een buildscript. Elk script bestaat uit drie fases:

1. De eerste fase bereidt de compilatie voor, zorgt dat de juiste bestanden uit SourceSafe worden opgehaald, controleert afhankelijkheden, bepaalt de compilatievolgorde en stelt configuratiewaarden in zoals de build- en versienummers.



Afbeelding 1. Stappen in een buildscript.

2. De tweede fase is de compilatie zelf. Daarbij is het belangrijk om zoveel mogelijk te bouwen. Neem dus geen tussenresultaten als uitgangspunt. Indien een WSDL-file is aangeleverd als webservice-interface, dan is het genereren van een proxy een betere test dan het gebruiken van een al bestaande proxy in SourceSafe.

3. Na een succesvolle compilatie kunnen er extra taken, de zogenaamde 'post build steps', worden uitgevoerd. Goede voorbeelden daarvan zijn: de controle op standaards en richtlijnen via de tool FxCop (www.gotdotnet.com) (www.gotdotnet.com/team/fxcop), het installeren van de build op testmachines via de Microsoft Installer en het uitvoeren van unit tests met bijvoorbeeld NUnit.

Implementeren buildproces

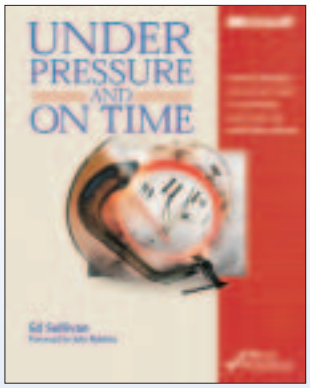
Het maken van een script en het aanwijzen van een compilatieserver is helaas niet genoeg. De nachtelijke build moet ook een onderdeel worden van het dagelijkse ontwikkelproces. Het team moet de moeite nemen het buildproces te configureren en aan te passen gedurende de realisatie, en iets doen met de testresultaten of opgetreden problemen. Daarbij is het belangrijk dat de nachtelijke build op een gemakkelijke manier genoeg uitvoer oplevert om problemen te kunnen oplossen. Het mailen van resultaten naar teamleden is een goede eerste stap. Het aanwijzen van een Build Manager in het team, die verantwoordelijk is voor het buildresultaat, is een prima vervolg. Voor teams die zich herkennen in het feit dat er tijdens het ontwikkelen weinig tijd overblijft voor iets anders dan programmeren, kan ik de volgende stok achter de deur aanbevelen. Zorg dat de opdrachtgever directe toegang heeft tot de testserver waar de buildresultaten staan, en geef hem inzicht in het buildproces. Beloof hem elke dag een nieuwe testversie. Hiervoor moet je natuurlijk zeker weten dat het buildscript goed werkt. Door de frequentie van de testserverinstallaties in het begin op één keer week te zetten, kun je het jezelf wat gemakkelijker maken.

Alle begin is moeilijk

Een buildproces functioneert niet zonder iemand die het resultaat van de build controleert en eventuele problemen oplost. Houd daarom bij de inrichting van een buildproces rekening met een opstartfase waarin je waarschijnlijk meer tijd kwijt bent aan het verwijderen van fouten uit de scripts, dan dat je voordeel hebt bij de nachtelijke build. Maar geduld wordt beloond. Als je eenmaal over een stabiel script beschikt, heb je een uitstekend middel in handen om de kwaliteit en voortgang van het ontwikkelproject te volgen.

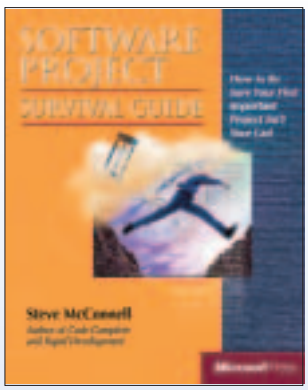
Uitgebreidere source-code en meer informatie over het buildproces zijn te vinden op www.etx.nl/Publicaties/Build

Microsoft Press



Titel: Under Pressure and On Time
ISBN: 0-7356-1184-X
Auteur: Author Ed Sullivan

Microsoft Press



Titel: Software Project Survival Guide
ISBN: 1-57231-621-7
Auteur: Steve McConnell

Nuttige internetadressen

http://www.gotdotnet.com
http://www.gotdotnet.com/team/fxcop