Gedurende het ontwikkelproces wordt de software op een bepaald moment naar de productieomgeving gebracht. In het klassieke proces, bijvoorbeeld het waterval proces, zorgt dit vaak voor de nodige stress. Werkt de software op de productieomgeving? Zijn we niets vergeten tijdens de uitrol? Zijn alle bugs wel verholpen? Hoe gaan we om met patches op de software vanuit de beheerorganisatie? Zijn de specialisten bij de uitrol betrokken of is er maar één developer die de build voor deze uitrol kan uitvoeren? Door het gebruik van continuous delivery hopen we deze problemen (deels) op te lossen.

Continuous Delivery

Altijd kunnen uitrollen op de productieomgeving

Bedrijven ontwikkelen zich steeds sneller. Ze willen snel inspelen op de behoefte van de markt. In de huidige tijd wordt het verschil tussen overleven of niet mede bepaald door de slagkracht van een (IT-)organisatie. De software van een bedrijf krijgt hier een steeds groter aandeel in. Het is van belang dat deze snel kan worden aangepast aan de behoeften van het bedrijf en dat deze snel met het bedrijf mee kan ontwikkelen. De Agile stroming die we in de afgelopen decennia hebben zien ontstaan vindt hier deels zijn oorsprong.

Veel initiatieven

We zien veel goede initiatieven in ons vakgebied: het nadenken over slagvaardige architecturen (o.a. SOA's), het leveren van kwaliteit van software (unit testing, continuous integration), alignment tussen gebruiker en ontwikkelaar middels agile processen (Scrum, eXtreme Programming), het verminderen en verwijderen van overbodige stappen in processen, vaak gebaseerd op lean principes.

Ondanks al deze verbeteringen blijft de in productiename van de nieuwe software nog steeds een spannende en vaak tijdrovende bezigheid. Het is geen uitzondering dat weken wordt gewerkt aan de voorbereiding van die inproductiename. Vaak doordat er eerst maanden wordt ontwikkeld, vervolgens wordt getest en een paar weken bugfixes worden uitgevoerd. Sterker nog, het is niet uitzonderlijk dat de software zich gedurende het softwareontwikkelproces gedurende lange tijd niet in een staat bevindt dat deze uit te rollen is op een (productie-)omgeving.

Continuous Delivery

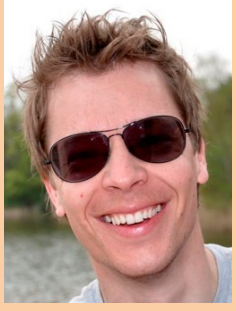
Het gereed maken van software voor een productieomgeving is een desinvestering; in lean termen zouden we dit 'waste' noemen. Immers, wat is de toegevoegde waarde in het ontwikkelproces en voor de bedrijfsvoering wanneer er kapitalen worden verspijkerd aan het gereed maken van de software voor de omgeving.

Een van de principes van de lean beweging is het integer houden van het proces (zie *Lean Software Development: An Agile Toolkit van Poppendieck*). Ik interpreteer dat als het zorgen dat de software te allen tijde een vorm heeft die het mogelijk maakt om deze uit te rollen en beschikbaar te stellen aan de gebruiker. Natuurlijk kan het voorkomen dat dit niet het geval is, bijvoorbeeld doordat een bug in de code een onvoorzien gedrag heeft opgeleverd. In dat het geval is het zaak om de software zo snel mogelijk weer in een goede staat terug te brengen.

Amplify learning

Ook hiervoor geldt dat hoe sneller het probleem opgelost wordt, hoe minder tijd (en dus geld) het kost om de problemen op te lossen. Ook hier zien we weer een lean principe terugkomen (amplify learning) in de vorm van een snelle feedback aan de gebruiker.

Deze principes worden toegepast in continuous delivery. Een van de doelen van continuous delivery is om ervoor te zorgen dat de software zich altijd in een staat bevindt die het mogelijk maakt om de software middels 'een-druk-op-de-knop' uit te rollen naar de productie omgeving.
De software is altijd 'production-ready' en iedere check-in levert een potentiële kandidaat voor productie op.



Sven Vintges
is werkzaam bij Atos Origin.

Deployment pipeline

Continuous Delivery, of automated delivery, kan helpen om de processen te ondersteunen en de IT-organisatie te helpen om slagvaardiger te worden. Om continuous delivery te implementeren is het nodig om een zogenaamde deployment pipeline op te zetten. Een deployment pipeline bestaat uit een aantal 'stages' die moeten worden doorlopen om de software live te brengen.

Een overzicht van de deployment pipeline wordt weergegeven in figuur 1 (afkomstig uit Continuous Delivery van Jez Humble en David Farjey.)

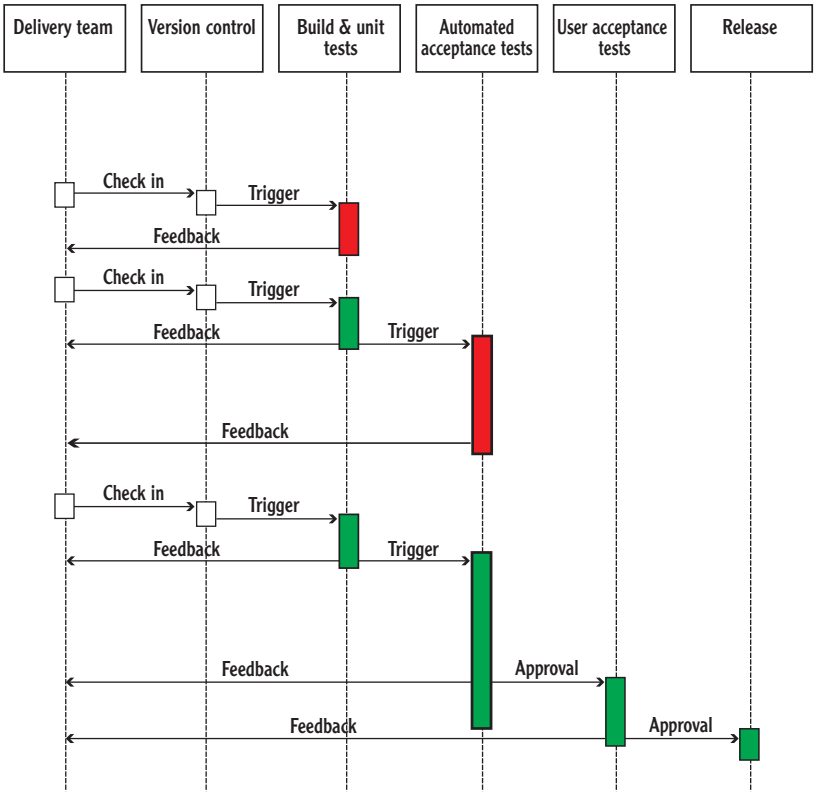
Deze figuur is de basis van continuous delivery. Het proces begint op het moment dat de ontwikkelaar een commit uitvoert van de code die is aangepast. Dit levert een trigger op naar het buildproces dat deze zal builden en de unit test zal uitvoeren. Uitgangspunt is dat deze build zo kort mogelijk moet zijn, liever minuten dan uren. De resultaten van deze build willen we zo snel mogelijk terugkoppelen aan de ontwikkelaars.

Na (eventuele) wijzigingen om de build te fixen slaagt de build en wordt de build gepromoveerd. Er worden geautomatiseerde acceptatie- en integratietesten uitgevoerd.
Hierbij kan gedacht worden aan het uitvoeren van webunit (*http://webunit.sourceforge.net/*), selenium (*http://seleniumhq.org/*), een set van SoapUI (*http://www.soapui.org*) testen, maar dit kan natuurlijk ook een eigen set van testen zijn. Op het moment dat één van de testen faalt, stopt de pipeline en wordt dit teruggekoppeld aan het ontwikkelteam dat de problemen moet oplossen.

De volgende stap is om de software op de user acceptance omgeving uit te rollen. Dit kan door de testengineer worden geinitieerd, maar dit kan natuurlijk ook geautomatiseerd door het systeem worden uitgevoerd. Wanneer de testengineer, na het uitvoeren van user acceptance en exploratory testing (*http://en.wikipedia.org/wiki/Exploratory_testing*) akkoord is, wordt de release candidate vrijgegeven en kan deze op de productieomgeving worden uitgerold.

Deployment vs Delivery

Veelal worden de termen continuous deployment en continuous delivery door elkaar gebruikt. Continuous deployment kan als een superset van continuous delivery worden gezien. In het geval van continuous deployment wordt iedere goede build direct doorgezet naar het productie ysteem. Bij continuous delivery is dat niet per definitie het geval. In de filosofie van continuous delivery moet het mogelijk zijn om iedere (succesvolle) build op de productieomgeving uit te rollen.



Figuur 1: Overzicht van de deployment pipeline.

Hierbij is het ultieme scenario dat de eindgebruiker (of een power user) zelf in charge is. Hij kan zelf bepalen wanneer welke versie van de software op welke omgeving wordt uitgerold. Het grote voordeel hiervan is dat de gebruiker zelf in control blijft en niet de IT-organisatie.
Een goed en uitgebreid stuk hierover is te vinden op *http://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/*.

Wanneer continuous delivery?

Continuous Delivery is, zoals je ondertussen waarschijnlijk hebt kunnen zien, absoluut geen goedkope oplossing. Het is ook geen oplossing die men in alle situaties moet toepassen of adviseren. Zoals bij alles wat we doen geldt dat de behoefte van de gebruikers centraal staat en dat wij 'slechts' faciliterend zijn. Wat we doen is immers geen bezigheidstherapie, maar moet ook een werkelijke bijdrage leveren.

Dit hulpmiddel, want meer dan dat is het niet, is dan ook erg geschikt voor mission-critical systemen waarbij een snelle slagkracht noodzakelijk is.
Zo kun je je voorstellen dat dit proces niet erg interessant is voor de administratie van een personeelsbestand, maar wel voor een bedrijf als Amazon dat meerdere farms heeft waarbij ze met regelmaat nieuwe features en fixes op hun software uitvoeren: een complexe delivery omgeving waarbij een snelle uitrol van nieuwe features van groot belang is.
Kort gezegd: je past continuous delivery toe voor strategische software, niet voor utility services (zie

Het is ondoenlijk (tenzij je een grote pot geld of een green field situatie hebt) om continuous delivery ineens te implementeren.

voor meer informatie *http://martinfowler.com/bliki/UtilityVsStrategicDichotomy.html* en *http://continuousdelivery.com/2011/01/strategic-vs-utility-services/#more-274*)

Aanpak implementatie

Inmiddels is duidelijk wat de aanpak van continuous delivery ongeveer inhoudt. Om dit volledig te implementeren zijn er veel zaken nodig. Het is dan ook ondoenlijk (tenzij je een grote pot geld of een green field situatie hebt) om dit ineens te implementeren.Vanuit ervaring kan ik dan ook enkel adviseren om met een evolutionair model te werken. Immers, zelfs bij een volledig nieuwbouwsysteem en deploymentomgeving is het nagenoeg niet mogelijk een systeem als dit in een keer te implementeren.

Je hebt een aantal componenten nodig om continuous delivery te implementeren. Ik zal de belangrijkste punten stapgewijs doornemen en daarna een voorstel doen voor de fasering om een deployment straat in te richten.

Versie- en configuratiebeheer

De software en haar afhankelijkheden dienen in een versiecontrolesysteem opgenomen te zijn. Dit betreft niet alleen de sources maar alle artefacten die nodig zijn om een representatie van het systeem te maken. Je kunt zelfs zover gaan dat je hierin het OS of de virtual machines opneemt, uitgangspunt is dat het te allen tijde mogelijk is om een werkend systeem te produceren.

Voor het beheer van je broncode kun je een product als SVN of GIT gebruiken. Deze systemen maken het mogelijk om de historie van je broncode te bekijken. Wie heeft bepaalde wijzigingen doorgevoerd, welke versie van de code werd gebruikt voor een bepaalde build (door het gebruik van tags)?
Hieronder valt ook het managemen van dependencies met producten als maven (*maven.apache.org*), Ivy (*ivy.apache.org*) en gradle (http://www.gradle.org/). En eventueel daarbij horende repositories als nexus (*http://nexus.sonatype.org/*)

Voor dit component geldt dat dit gaandeweg kan worden opgebouwd. Zo zou je ervoor kunnen kiezen om telkens wanneer je een component uirolt, je delen van de overige componenten opneemt in versiebeheer (bijvoorbeeld die enige configuratie file die je telkens in je JAR aan moet passen om de binary aan de omgeving aan te passen) etc.

Continuous Integration

Een van de fundamenten van de continuous delivery is continuous integration (CI). Met CI wordt bij iedere commit een integration build en test gedaan waarbij wordt gecontroleerd of een bepaalde commit de software niet kapot maakt (*http://en.wikipedia.org/wiki/Continuous_integration*). CI gaat uit van 'integreer vaak en en zo vroeg mogelijk'. Voor een deployment pipeline is dit de allereerste sanity check en de eerste mogelijkheid tot feedback aan het ontwikkelteam.

De eerste stap om CI te implemeteren is het installeren van een product als JenskinCI (*http://jenkins-ci.org/*) de opvolger van Hudson CI (*http://hudson-ci.org/*) of Bamboo (*http://www.atlassian.com/software/bamboo/*). Met deze producten is het eenvoudig om op Maven of Ant gebaseerde jobs te realiseren die middels een check-out van SVN of GIT een build uitvoeren, onafhankelijk van de machine van een ontwikkelaar.

Automated Tests

Een volgende verdieping is het toevoegen van geautomatiseerde tests. Hierbij kan gedacht worden aan het uitvoeren van unit test middels een product als jUnit (*www.junit.org*) of ngUnit (*www.ngunit.org*). Op dit niveau kan ook codeanalyse worden gebruikt door bijvoorbeeld een product als sonar (*http://www.sonarsource.org/*) waarbij er een check plaatsvindt op bijvoorbeeld codeconventie, veelgemaakte fouten etc.
De automated tests kunnen vervolgens worden uitgebreid naar geautomatiseerde acceptatietesten met behulp van producten als Selenium, web unit, etc.

Automated Deployment

Deze techniek is noodzakelijk om de pipeline te doorlopen. Voor het uitvoeren van het grootste deel van de tests is het noodzakelijk dat het systeem op een representatieve omgeving wordt uitgerold (binnen het redelijke, uiteraard). In het ideale geval zijn alle koppelvlakken beschikbaar en is in de aangrenzende systemen een gecontrolleerde testset aanwe-

zig. Waar dit niet mogelijk is kan worden gewerkt met Mock Services.

Automated deployment wordt vaak geïmplementeerd met custom scripts. Je kunt ervoor kiezen om in Maven deployment projecten aan te maken. Deze zorgt voor de uitrol naar de verschillende omgevingen en machines. In een aantal gevallen is het voldoende om een batch of shell script te ontwikkelen dat de componenten uitrolt. In geavanceerde gevallen kan gebruik worden gemaakt van distributiesystemen zoals het debian package management of puppet (*http://www.puppetlabs.com/*)

In ieder geval zal, afhankelijk van de omgeving, een specifieke implementatie moeten worden ontwikkeld om de uitrol mogelijk te maken. Verschillende tools kunnen je voor standaard containers helpen met de uitrol, je kunt dan denken aan de Cargo plugin (*http://cargo.codehaus.org/Maven2+plugin*) voor maven, die standaard een uitrol kan doen naar de meest bekende web containers (Apache Tomcat, Glassfish, Jboss, Jetty, etc). Deze plugin maakt het zelf mogelijk om een volledige nieuwe installatie van de applicatie server neer te zetten door deze vanaf het internet (of een interne site) te downloaden. Hiermee kun je je productielijn zover doorvoeren dat je zelfs een nieuwe machine met 1 druk op de knop van een versie van je software kunt voorzien.

Fasering implementatie

Om tot een implementatie te komen van continuous delivery kan de volgende hantering worden gebruikt.

1. Implementatie versiebeheer. Wanneer deze er nog niet is, is dit de eerste vereiste.
2. CI Server inrichten, de eenvoudigste stap is het downloaden van Jenkins CI
3. Opbouwen van unit tests;
4. Automated Deployment;
5. Maken van automated acceptance tests.

Conclusie

Door het toepassen van het principe uit CI, en Continuous Delivery kan er voor worden gezorgd dat een productieversie van de software gereed is voor uitrol. Hiermee voorkomen we dat er lange bug-fix cycli zijn, maar zorgen we er ook voor dat eenvoudig een nieuwe versie van de software naar de productieomgeving kan worden uitgerold. «

In het meest ideale geval is de gebruiker in control en niet de IT organisatie.