

WAT BETEKENT BUSINESS AGILITY VOOR UW ONTWIKKELSTRAAT?

SAMENVATTING

Voor het bereiken van business agility is meer nodig dan een juiste architectuur en is een iteratieve aanpak essentieel. Daarvoor is het nodig de ontwikkelstraat dusdanig te automatiseren dat de cost of change constant blijft. Belangrijke onderdelen van een ontwikkelstraat zijn een versiebeheerrepository, bug- en taakdatabase, buildscript, continuous integration server en automatische QA-controles.

ITERATIEVE AANPAK

Bedrijven moeten snel kunnen inspelen op veranderingen en verwachten hetzelfde van IT-projecten. De auteur zet uiteen hoe ontwikkelstraten kunnen worden ingericht om beter aan deze behoefte aan business agility te voldoen.

Andrej Koelewijn, IT-eye

Business Agility is op dit moment een hot topic in de IT. Bedrijven zijn het zat dat IT projecten jaren duren en hierdoor altijd achter de feiten aan lopen. Om te overleven moeten bedrijven snel kunnen reageren op veranderingen. Van de automatisering wordt het zelfde verwacht. Nieuwe kansen in de markt moeten snel gegrepen kunnen worden. IT mag hierbij geen opstakel zijn. Hoe kan een software development ontwikkelstraat de wens naar Business Agility ondersteunen?

ONTWIKKELSTRAAT

Voordat we een ontwikkelstraat kunnen definiëren is het goed te kijken naar het doel hiervan. Een bedrijf bouwt software zodat het daarmee zijn doelen beter kan bereiken. Het traject van bepalen van doelen tot bruikbare software ziet er in grote lijnen als volgt uit:



Het is goed je hierbij te realiseren dat doelen kunnen veranderen. En de manier waarop je je doelen bereikt zal waarschijnlijk in de loop van tijd ook veranderen. Bedrijven willen snel kunnen reageren op veranderde markt-omstandigheden, veranderde inzichten, nieuwe wetgeving, nieuwe kansen, nieuwe plannen, etc. Hier komt vaak de uitdrukking Business Agility om de hoek kijken: de mate waarin IT snel kan inspelen op de veranderende behoefte van de Business.

BUSINESS AGILITY

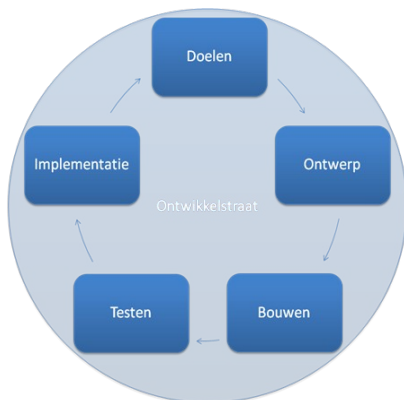
Wat betekent Business Agility voor een software ontwikkelstraat? Een ontwikkelstraat is de manier waarop je van bedrijfsdoel naar software implementatie komt: de verzameling van methodieken, procedures en tools die je binnen dit traject gebruikt. Om in te spelen op de behoefte aan Business Agility moet het traject van doel naar implementatie zo kort mogelijk worden. Hoe sneller software gerealiseerd kan worden die voldoet aan de behoefte van de business, des te effectiever kan de business acteren. Een ontwikkelstraat moet dus zo ingericht worden dat je zo snel mogelijk van behoefte naar implementatie komt. Dit kun je helemaal aan een software-ontwikkelaar overlaten, maar effectiever wordt het als je binnen een ontwikkelstraat gebruik maakt van procedures en tooling.

Vaak wordt gesteld dat veranderingen kostbaar zijn. Hoe later in het ontwikkelproces een wijziging aangebracht wordt, hoe duurder. Veel werk dat al gedaan is moet aangepast worden of soms zelfs helemaal overnieuw. Kan dit anders? Voor een optimale Business Agility zou het wenselijk zijn als de 'cost of change' constant zou blijven. Of een wijziging nu over een week, over een maand of over een half jaar wordt aangevraagd, als het belangrijk is voor de business zou het eenvoudig en goedkoop moeten zijn deze wijziging uit te voeren. Hoe kan een ontwikkelstraat bijdragen aan een constante 'cost of change'?

Wat betekent business agility voor uw ontwikkelstraat?

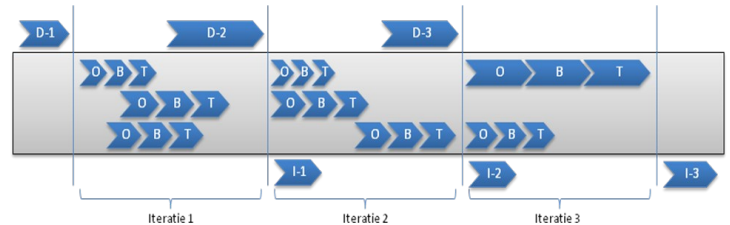
PROCEDURES

Wat betreft procedures kunnen we stellen dat iteratieve ontwikkelmethodieken meer bijdragen tot Business Agility dan een waterval aanpak. Binnen een iteratieve aanpak wordt een beperkte set van behoeften in een korte periode gerealiseerd. Idealiter kan de gerealiseerde software na afloop van een iteratie direct ingezet worden. Ook kunnen de behoeften en doelen voor iedere iteratie opnieuw bepaald worden. De business kan dus iedere iteratie bepalen wat de prioriteiten zijn. Een waterval aanpak daarentegen duurt meestal veel langer. Dit betekent dat indien de behoeften van de business halverwege veranderen, er veel overbodig werk weggegooid moet worden, of dat nieuwe wensen pas veel later gerealiseerd kunnen worden. Uit oogpunt van Business Agility zou het eerder getoonde diagram er dus als volgt uit moeten zien:



Een iteratieve aanpak heeft niet alleen als voordeel dat het de business in staat stelt om snel bij te sturen, het zorgt ook al in grote mate voor een constante 'cost of change'. Als we niet te ver vooruit werken met het uitwerken van requirements, ontwerp, implementatie en testen, dan hebben we dus nooit veel werk gedaan dat als 'waste' aangemerkt kan worden indien plannen veranderen. Pas als een feature of wens in een iteratie daadwerkelijk gerealiseerd gaat worden worden de detail requirements bepaald, het technisch ontwerp gemaakt, de software geschreven, en de code getest. Alleen dat wat echt direct wenselijk is wordt

uitgewerkt. Platgeslagen ziet bovenstaand iteratief ontwikkelproces er dan als volgt uit:



Voor iedere iteratie worden de doelen bepaald (D1,D2,D3). Tijdens een iteratie worden deze doelen ontworpen, gebouwd en getest (O,B,T). Na afloop kunnen de gerealiseerde functies in productie geplaatst worden (I1, I2, I3).

Naast een iteratieve aanpak moet er ook voor gezorgd worden dat andere procedures geen vertragende factor vormen. Zeker indien dit gaat over zaken die iedere iteratie terug komen. Goedkeuring van een ontwerp of architectuur is hiervan een mooi voorbeeld. Indien je iedere iteratie alle architectuur beslissingen moet documenteren en daarna door een architectuur-commissie moet laten goedkeuren, loop je kans ernstige vertragingen te krijgen in je iteratie. Dit soort procedures werken een goede Business Agility dus tegen.

Een oplossing hiervoor is om met cross-functional teams te werken. Een ontwikkelteam bestaat uit mensen die verstand hebben van alle disciplines die nodig zijn bij het ontwikkelen van software: ontwikkelaars, testers, documentatie schrijvers, ontwerpers, architecten, planners. Je zou denken dat je zo enorm grote team krijgt, maar dit valt in de praktijk mee, als je gebruik maakt van cross-functional ontwikkelaars. Van een ontwikkelaars wordt dus verwacht dat hij van meerdere onderwerpen verstand heeft. Niet iedereen van alles evenveel, maar wel zodanig dat het team in zijn geheel alle vlakken afdekt. Er moet dus minimaal een persoon in een team aanwezig zijn die weet welke architectuur standaarden gehanteerd dienen te worden. Hij zorgt ervoor dat de juiste ontwerp beslissingen worden genomen, en hij kan ook tijdens

de gehele iteratie controleren dat deze beslissingen op de juiste wijze worden toegepast.

Op deze manier werken heeft een aantal voordelen. De belangrijkste is de snelheid waarmee ontwikkeld kan worden. Daarnaast is er tijdens het gehele traject de juiste kennis aanwezig, en kan ervoor gezorgd worden dat deze ook goed wordt toegepast. De andere kant op helpt het ook. Een architect staat met zijn voeten in de modder en merkt ook of standaarden ook echt werken in de praktijk. Daarnaast krijg je een groot learning-on-the-job effect. Mensen in een cross-functional team leren veel meer van elkaar, dan wanneer ze alleen samenwerken met mensen met dezelfde kennis. En tenslotte wordt het risico kleiner dat het project stil komt te liggen indien een van de teamleden ontbreekt. Er zijn tenslotte meerdere mensen die dit kunnen opvangen.

TOOLING

Naast ontwikkelmethodiek is ook het gebruik van de juiste tooling belangrijk voor het realiseren van Business Agility. We zijn dus op zoek naar tooling in de ontwikkelstraat die het mogelijk maakt om de 'cost of change' zo constant mogelijk te houden en wensen zo snel mogelijk te realiseren. Bij tooling wordt vaak gedacht aan een ontwikkelomgeving, ofwel IDE (integrated development environment). Er zijn echter nog veel meer mogelijkheden om tooling te gebruiken. Hoe meer we kunnen automatiseren van wens tot implementatie, hoe meer tijd we overhouden in een iteratie voor het daadwerkelijk realiseren van de benodigde functionaliteit. Een belangrijk onderdeel is bijvoorbeeld het testen. Hoe eenvoudiger het wordt om software te testen en uit te rollen, hoe goedkoper het wordt om wijzigingen aan te brengen.

Bij de meeste ontwikkeltrajecten wordt er gebruik gemaakt van een aantal ontwikkel pc's en een zogenaamde OTAP straat. Dit staat voor Ontwikkel, Test, Acceptatie, en Productie. De OTAP straat bevat voor iedere deployment omgeving (bijvoorbeeld

applicatie server, database server, etc) een instantie. Het is zinvol om naast de OTAP onderdelen nog een aantal extra tools in te zetten. Deze worden hieronder toegelicht.

Een van de belangrijkste tools in een ontwikkelstraat is een **versie beheer repository**. Deze wordt gebruikt om iedere wijziging aan iedere file bij te houden. Een goed versie beheer systeem maakt het mogelijk dat verschillende ontwikkelaars tegelijkertijd aan dezelfde bestanden werken. Ook geeft het inzicht in de wijzigingen die gemaakt zijn per requirement. Normaal gesproken werken ontwikkelaars lokaal op een ontwikkel pc totdat de feature die ze implementeren min of meer klaar is. De wijzigingen worden dan in het centrale versie beheer systeem gezet, zodat iedereen erbij kan. Een versie beheer systeem is dus geen backup alternatief, waarin halve, niet werkende code geplaatst wordt.

Een **bug en task database** kan voor een aantal zaken gebruikt worden. Naast het bijhouden van problemen in de software kan het ook goed ingezet worden voor het documenteren van taken. Welke taken dienen er nog opgepakt te worden? Een goede bug database kan gekoppeld worden aan het versie beheer systeem. Zo is snel duidelijk welke software wijzigingen een bug of taak tot gevolg heeft. Uitgebreide bug databases kunnen ook gebruikt worden voor project management. Requirements en taken kunnen ingevoerd worden, waarbij aangegeven wordt welke prioriteit ze hebben, en voor welke iteratie ze geplanned zijn. Vervolgens is het mogelijk om de voortgang van het project te monitoren door te controleren welke taken en requirement reeds afgemeld zijn.

Een ander belangrijk onderdeel van een ontwikkelstraat is een **continuous integration server**. Deze zorgt ervoor dat software regelmatig automatisch gecompileerd, getest, gecontroleerd en geïnstalleerd wordt. Alle onderdelen van de applicatie worden zo vroeg mogelijk geïntegreerd, gecompileerd en getest. Als integratie uitgesteld wordt tot het einde van de bouwperiode kan dit vaak

Wat betekent business agility voor uw ontwikkelstraat?

leiden tot grote problemen, en is er veel tijd nodig om de issues op te lossen. Het naar voren halen van de integratie leidt ertoe dat altijd duidelijk is of de verschillende componenten goed samenwerken. Het is aan te raden om minimaal 1 keer per 24 uur, bijvoorbeeld 's nachts, een automatische build uit te voeren. Het is echter ook zinvol om na iedere wijziging in het versie beheer systeem een automatische compilatie uit te laten voeren. Hiermee wordt direct zichtbaar of de code die is ingechecked in het versie beheer systeem wel compileert. Dit voorkomt dat nieuwe aanpassingen problemen opleveren voor andere ontwikkelaars.

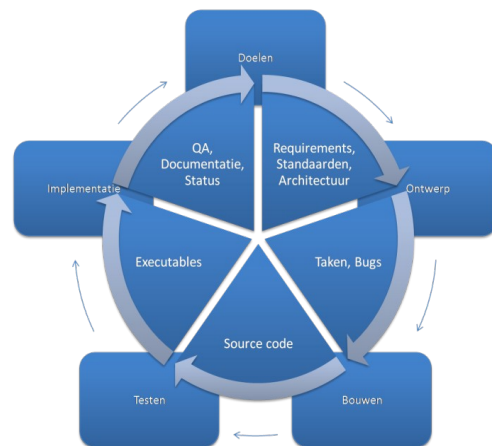
Voorwaarde voor het gebruik van een continuus integration server is het hebben van een **build script**. Een build script beschrijft exact welke stappen er nodig zijn om een software product te compileren, te testen, te packagen, en te installeren. Door dit te automatiseren mbv een script, is de inspanning minimaal om dit process regelmatig uit te voeren. Na een wijziging aan de source code door een ontwikkelaar kan binnen minuten, geheel automatisch, duidelijk zijn of de code werkt en installeert. Een belangrijke stap voor het bereiken van een contante 'cost of change'.

Een belangrijk onderdeel van een iteratieve aanpak zijn **geautomatiseerde test cases**. Deze zijn er op verschillende niveaus: unit-, integratie-, systeem-, en acceptatie-tests. Hoe meer deze geautomatiseerd zijn, hoe lager het risico van aanpassingen wordt. Na iedere aanpassing is immers direct duidelijk of de aanpassing werkt, en of bestaande onderdelen van het systeem nog correct functioneren. Meestal duren iteraties tussen de 2 en 4 weken. Er is dus geen tijd om na iedere iteratie in 1 of 2 weken het gehele systeem handmatig te testen. Zoveel mogelijk automatisch testen is dus essentieel.

Ook kunnen een groot aantal andere **QA** stappen met tools ondersteund worden. Het is bijvoorbeeld mogelijk controle op **naamgevings standaarden** te automatiseren. **Code coverage rapportage** geeft inzicht in de mate waarin de applicatie getest is mbv

geautomatiseerde tests. Andere tools kunnen veel gemaakte **fouten** in de code detecteren. Verder is het ook mogelijk om code **opmaak** te automatiseren, zodat alle code altijd op de zelfde manier uitgelijnd is. Dit verhoogt de leesbaarheid en verlaagt dus de onderhoudskosten.

Er zijn een aantal **geïntegreerde buildserver** systemen beschikbaar. Deze brengen alle hierboven genoemde tools samen, zodat eenvoudig een totaal overzicht gekregen kan worden. Zo kun je bijvoorbeeld snel zien welke wijzigingen een requirement tot gevolg heeft, of alle tests nog werken na het implementeren van de wijzigingen, welke code wijzigingen niet voldoen aan de standaarden. Op ieder moment is duidelijk wat de status is van het systeem: welke requirements zijn af, hoeveel bugs zitten er nog in de code, welke code wordt nog niet automatisch getest, voldoet de code aan de standaarden, etc. Een buildserver is dus een centraal onderdeel van de ontwikkelstraat, dat ondersteuning biedt voor alle stappen van de ontwikkelstraat.



Vaak worden 4GL ontwikkeltools en CASE generatie tools beschouwd als de beste middelen voor het verhogen van de productiviteit van ontwikkelaars. Een belangrijke vraag die echter beantwoord moet worden is of een visuele **ontwikkel omgeving** ook altijd de meest productieve ontwikkelomgeving is binnen de gehele ontwikkelstraat. Grafische tooling lijkt vaak een stap vooruit te zijn, maar indien de

gegenereerde code bijvoorbeeld niet geautomatiseerd getest of gedeployed kan worden, kan het een negatief effect hebben op de totale productiviteit. Ook is het belangrijk om te onderzoeken of tooling iteratief ontwikkelen ondersteund. Is dit niet het geval, wordt het moeilijk Business Agility te bereiken. Het is dus goed om hiermee rekening te houden bij de keuze van de ontwikkeltools.

ONTWIKKELPROCEDURES

Het werken op een iteratieve wijze met behulp van een buildserver vraagt om een bepaalde aanpak van de ontwikkelaars. Een van de grote verschillen met waterval projecten is het resultaat van de bouwfase. In traditionele omgevingen zie je vaak dat een applicatie wordt opgebouwd in een deployment container. Ontwikkelaars passen bijvoorbeeld een database zolang aan totdat het doet wat het moet doen. Ze zijn erop gericht de juiste functionaliteit in de database te realiseren. Het migreren van de ene omgeving naar de volgende, bijvoorbeeld van ontwikkel naar test, is echter een ondergeschoven kindje. Vaak wordt van de applicatie-beheerder of database-beheerder verwacht dat hij of zij dit wel even doet door wat installatie-scripts te maken.

In een iteratieve aanpak gaat dit echter te veel tijden kosten. Het installeren van een deployment omgeving zou volledig geautomatiseerd moeten gebeuren, zodat het minimaal 1 keer per nacht kan plaatsvinden. Hiervoor is het noodzakelijk dat ontwikkelaars niet gericht zijn op het inrichten van een omgeving, maar op het opzetten van een nieuwe of het upgraden van een bestaande omgeving. Zij zijn er dus voor verantwoordelijk dat er scripts komen waarmee de applicatie geïnstalleerd kan worden.

De migratie van de ene omgeving naar de volgende omgeving moet vervolgens geautomatiseerd kunnen plaatsvinden. Hierbij is het belangrijk dat er geen wijzigingen meer kunnen plaatsvinden na het compileren van de source code. Dezelfde code moet kunnen werken op de ontwikkelomgeving, de

testomgeving, en de productieomgeving. Dit garandeert een constante kwaliteit omdat datgene wat is goedgekeurd in test ook echt op de productie omgeving terecht komt. Er is geen handmatige actie meer nodig, en er kunnen dus ook geen fouten gemaakt worden bij de migratie.

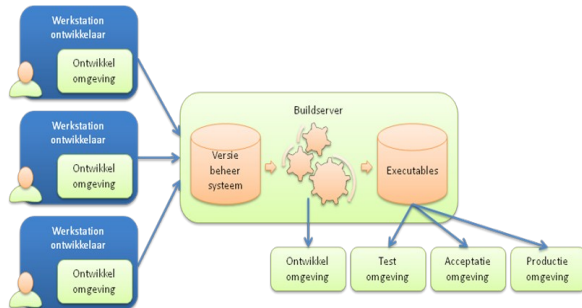
Verder is het belangrijk dat de code die op de deployment omgevingen (OTAP) geïnstalleerd wordt, niet door een ontwikkelaar handmatig gecompileerd is op een ontwikkelmachine. Dit laat namelijk een hoop ruimte voor fouten. Welke code draait er precies in productie? Is de source hiervan wel beschikbaar in het versie beheer systeem? Of was dit misschien een lokale wijziging van een ontwikkelaar en staat de code alleen op zijn pc? Alleen als een automatisch build process de executables compileert mbv van de source code uit het versie beheer systeem is precies te achterhalen welke versie van welke file op productie of test draait. Zo is goed te achterhalen welke code fouten bevat, en of de fixes ook echt in productie terecht komen.

Een handig middel om ervoor te zorgen dat ontwikkelaars eenvoudig te installeren code schrijven is door iedere ontwikkelaar zijn eigen ontwikkelomgeving te geven. Iedere ontwikkelaar heeft dus zijn eigen database, en zijn eigen applicatieserver. Indien hij de wijzigingen, ingechecked door andere ontwikkelaars, wil gebruiken zal hij zijn eigen ontwikkelomgeving moeten upgraden met de laatste stand van de code uit het versie beheer systeem. Dit werkt natuurlijk het eenvoudigst indien er een goed installatiescript voorhanden is.

Onderstaande diagram illustreert deze aanpak. Iedere ontwikkelaar heeft zijn eigen ontwikkelomgeving. Deze kan bestaan uit een database, een applicatie server en eventuele andere benodigde deployment containers. Zodra de ontwikkelaar wijzigingen incheckt in het versiebeheer systeem wordt de applicatie gecompileerd en op de centrale ontwikkel omgeving geïnstalleerd.

Wat betekent business agility voor uw ontwikkelstraat?

Vervolgens is het aan een beheerder om de executables op de overige omgevingen te installeren.



VOORBEELD TOOLING

Voor een Java SOA project heeft IT-eye gebruik gemaakt van een ontwikkelstraat op basis van de volgende opensource tools:

- **Subversion** – centraal opensource versie beheer systeem.
- **Trac** – Opensource bug database en wiki. Trac biedt ook inzicht in project-voortgang en code wijzigingen.
- **Agilo** – Scrum plugin voor Trac. Biedt een storyboard en burndown-charts voor het monitoren van project-voortgang.
- **Hudson** – Opensource continuous integration software. Zorgt voor het automatisch uitvoeren van de build scripts.
- **Nexus** – Library repository. Bevat alle benodigde afhankelijkheden, en alle opgeleverde software.
- **Maven** - Opensource build software. Zorgt voor compilatie, inpakken, uitvoeren van tests en genereren van documentatie.

Op een ander project is er gebruik gemaakt van een suite van tools van de firma Atlassian:

- **Jira** – bug database.

- **Confluence** – wiki omgeving.
- **Bamboo** – continuous integration server.
- **Greenhopper** – scrum plugin voor Jira

Deze suite is aangevuld met de eerder genoemde tools **Maven**, **Subversion** en **Nexus**.

Subversion is een centraal versie beheersysteem. Het is niet altijd mogelijk om alle medewerkers gebruik te laten maken van een centraal systeem. In sommige situaties als telewerken zonder vpn toegang tot het centrale systeem, of outsourcing van projecten naar verschillende externe leveranciers, kan een gedistribueerd versiebeheer systeem beter functioneren. IT-eye heeft in deze gevallen goede ervaringen met het versiebeheer systeem **Git**.

CONCLUSIE

In tegenstelling tot wat je tegenwoordig vaak hoort is een juiste architectuur niet voldoende voor het realiseren van Business Agility. Indien je ontwikkelaanpak zo is ingericht dat er minimaal een half jaar zit tussen wens en realisatie, dan zul je nooit echt Business Agility bereiken. Voor het bereiken van Business Agility is een iteratieve aanpak essentieel.

Dit gaat echter alleen werken indien de ontwikkelstraat dusdanig geautomatiseerd is, dat de 'cost of change' constant blijft. Je kunt het je niet permitteren om iedere iteratie handmatig te testen en te deployen. Dit kost te veel tijd, en laat veel ruimte over voor fouten. Het is dus belangrijk om je ontwikkelstraat zo veel mogelijk te automatiseren. Belangrijke onderdelen hierin zijn een versie beheer repository, bug/taak database, build script, continuous integration server, en automatische QA controles. Een ontwikkelstraat verhoogt dus de productiviteit, verbetert de kwaliteit, maakt de status van het project beter zichtbaar, en stelt IT in staat om snel te reageren op wensen van de Business.

Wat betekent business agility voor uw ontwikkelstraat?

Voor het bereiken van business agility is een iteratieve aanpak essentieel

ANDREJ KOELEWIJN

Andrej Koelewijn is IT Architect bij IT-eye. E-mail:
andrej.koelewijn@it-eye.nl.