Drs. J. van Brummelen
is manager bij KPMG Advisory N.V.
vanbrummelen.jos@kpmg.nl

Drs. J.M.A. Koedijk CISA CISM
is partner bij KPMG Advisory N.V.
koedijk.joost@kpmg.nl

Efficiënte waardecreatie met agile softwareontwikkeling

Drs. Jos van Brummelen en drs. Joost Koedijk CISA CISM

Het eerste principe van het Agile Software Manifesto is dat moet worden gewerkt aan het constant opleveren van 'waardevolle software'. In de praktijk van agile softwareontwikkeling echter wordt nog weinig aandacht besteed aan het waardeaspect en wordt onvoldoende gekeken hoeveel het realiseren van bepaalde functionaliteit/features in euro's oplevert. In dit artikel wordt uiteengezet hoe met behulp van Continuous Integration en de Theory of Constraints deze waarde kan worden bepaald en productiviteitsverbetering kan worden gerealiseerd.

Inleiding

Wie op Google de woorden 'value of software' intikt, doet al snel een bijzondere waarneming. Veel van de hits gaan niet over waarde (toevoegen), maar over kosten. Deze resultaten bevestigen dat er in de zakelijke IT-literatuur veel meer wordt gepubliceerd over de kosten van software dan over de waarde. Dit wordt veroorzaakt door het feit dat, zoals ook uit de rest van dit artikel blijkt, 'de waarde van software' een lastig begrip is. De kosten daarentegen zijn te benoemen, meetbaar of tenminste te schatten. Zoals de Google-zoekresultaten illustreren wordt daardoor vaak ook de nadruk gelegd op de kosten van software.

De kosten van software zijn in drie fasen te verdelen: initiële kosten, gebruikskosten en kosten voor het buiten gebruik stellen. De initiële kosten worden gemaakt om de software aan te schaffen, te bouwen, naar wens aan te passen, te installeren en in gebruik te nemen. De gebruikskosten omvatten onder meer licentiekosten, het in stand houden van de (hardware)omgeving, gebruikersbeheer, back-ups, het oplossen van incidenten, het beheren van de (master)data en het realiseren van aanvullende wensen rond functionaliteit en rapportages. Ten slotte, hoewel vaak vergeten, kleven er ook aan het einde van de levensduur nog kosten aan het buiten gebruik stellen van software, al was het maar om vloerruimte leeg te maken en tenminste te voorkomen dat vertrouwelijke gegevens via de (fysieke) vuilnisbak in verkeerde handen vallen.

Al deze focus op kosten levert echter direct spanning op met het eerste principe van het Agile Software Manifesto. Dat luidt immers dat moet worden gewerkt aan het constant opleveren van 'waardevolle software'. De waarde die met software in bedrijfsprocessen wordt toegevoegd is (uiteraard) ook de reden om in veel bedrijven software te gebruiken. Dit maakt de waarde van (de inzet van) software bedrijfsspecifiek en daardoor vaak lastig te bepalen, laat staan te vergelijken.

In agile softwareontwikkeling worden in het algemeen op basis van *business value* prioriteiten vastgesteld. Gezien het bovenstaande zal het geen verrassing zijn dat deze waarde, net als in businesscases die wij in de praktijk aantreffen, veelal niet in harde euro's wordt uitgedrukt. Er wordt dan overgegaan naar een relatieve waardering van kleine systeemfuncties. Deze business value van kleine systeemfuncties, die door de *product owner* (eventueel aangevuld met andere stakeholders) wordt toegekend, blijkt in de praktijk goed bruikbaar. Op basis van deze business value,

aangevuld met een kritische blik op de wijze waarop business value wordt vastgesteld, blijkt het, zoals in dit artikel naar voren komt, zelfs mogelijk de productiviteit van een softwareontwikkelorganisatie in kaart te brengen en significant te verbeteren.

In dit artikel ligt de focus op (de ontwikkeling van) maatwerksoftware. Wat betreft de gevolgen van het inzetten van standaardoplossingen en pakketsoftware hoeft alleen maar bedacht te worden dat de initiële kosten van standaardoplossingen anders tot stand komen. In navolging van Fred Brooks Jr. ([Bro095]) is de verwachting dat het delen van de standaardoplossing (over veel implementaties) leidt tot het delen van de realisatiekosten, waardoor die lager uitvallen. Over de waarde van deze oplossingen na specifieke implementaties is, net als bij maatwerksoftware, weinig bekend.

De kosten van software

We laten ons eerst leiden door de resultaten die Google geeft en de waarneming die we hebben gedaan. We komen er dan al snel achter dat er in veel publicaties (zie onder meer [Kell97] en [Jone11]) enkel en alleen wordt gekeken naar de keiharde cijfers, dus naar de manier waarop ontwikkelde (maatwerk)software wordt gewaardeerd op de balans en hoe dit zich verhoudt tot de gemaakte kosten. Zo ook in een artikel uit een vorige editie van Compact ([Gink03]). Uit deze uiteenzetting leren we onder andere dat de internationale voorschriften en verslaggevingsrichtlijnen (de IFRS, maar ook de GAAP) verplichten dat ten aanzien van het activeren van software op de balans primair wordt gekeken naar de kosten die gemaakt zijn tijdens de ontwikkeling ervan en dat niet wordt uitgegaan van de intrinsieke waarde die de software vertegenwoordigt.

De Software Improvement Group presenteerde een aanvullend model om de gemaakte kosten goed te waarderen ([Groo12]). De basis voor de berekening is de omvang van de software, objectief vastgesteld door het aantal broncoderegels te tellen, en de bijdrage die een gemiddelde softwareprogrammeur per eenheid van tijd (dag, week, maand, jaar et cetera) levert. In dit model wordt allereerst voorgesteld om bij de waardering een correctie toe te passen om te kunnen omgaan met een afwijkend aantal broncoderegels ten opzichte van industriebenchmarks. De waarde die hieruit volgt wordt als vervangingswaarde gezien en is dus in zekere zin, en met mitsen en maren of dat voor het specifieke geval toepasbaar is, gebaseerd op industriebenchmarks van de productiviteit van ontwikkelteams en gecorrigeerd voor inefficiënt ontwikkelen. Deze vervangingswaarde kan wellicht geschikt zijn voor boekhoudkundige doelstellingen; de risico's van het opnieuw ontwikkelen en implementeren van een systeem moeten bij daadwerkelijke vervanging ook wederom gewaardeerd worden.

Het model gaat verder door huidige en toekomstige kosten van het gebruik van het systeem, de beheerkosten, te relateren aan de functionele en technische kwaliteit van het systeem. Hierbij wordt getracht een financiële correctie toe te passen op de waarde van het systeem waarbij deze kosten zouden kunnen worden gezien als 'interest' op de technische schuld (of wederkerende kosten veroorzaakt door een niet optimale technische implementatie van de software; zie voor een toelichting op het begrip 'technische schuld' bijvoorbeeld [Amor13]). Zeker voor het functionele deel blijft deze financiële correctie moeilijk te kwantificeren. Allereerst zullen er altijd functionele wensen tot wijziging zijn en het is, zeker bij veranderende regelgeving die moet worden ondersteund, niet zo dat die altijd zijn terug te leiden naar het begin van de systeemontwikkeling. Bovendien is de gekozen technologie, bijvoorbeeld bij de platformkeuze, afhankelijk van de smaak en van verdere ontwikkelingen, wat op den duur ook tot discontinuïteiten in de waardering kan leiden. Maar bovenal zit in maatwerksoftware kennis van de organisatie – en van de wijze waarop zij waarde wil toevoegen – vervat die niet in het waarderingsmodel wordt meegenomen. En een werkend (waardetoevoegend) systeem blijft, ook als sprake is van een technisch matige uitvoering, gewoon veel waard!

De waarde van (de inzet van) software is bedrijfsspecifiek en daardoor vaak lastig te bepalen, laat staan te vergelijken

Er wordt nog (te) weinig aandacht besteed aan het waardeaspect: wat levert het realiseren van bepaalde functionaliteit/features daadwerkelijk op omgerekend in harde euro's?

Software als het toevoegen van waarde

De (boekhoudkundige) waardering van software in gebruik mag dan weinig aandacht krijgen, er wordt wel veel geschreven over het begroten van nieuwbouw (of aanbouw) in softwareontwikkelprojecten. De aandacht voor het vooraf goed in kaart brengen van de kosten heeft er nog niet toe geleid dat projecten ook altijd (min of meer) conform deze raming verlopen. Regelmatig worden we in projecten waar softwareontwikkeling plaatsvindt, zeker in administratieve omgevingen, geconfronteerd met uitloop en niet-toereikende budgetten. Dit leidt tot de vraag of er ondanks alle moeite niet goed begroot wordt of dat deze trajecten iets onvoorspelbaars zijn blijven houden.

De genomen maatregelen leiden er in ieder geval toe dat er vaak tot op de kleinste functie begroot wordt. Ook al omdat op die manier (a priori) beter inzichtelijk te maken is wanneer functies buiten de scope van het project vallen. Dit geeft namelijk de projectleider een handvat om te kunnen sturen naar tijdige afronding – binnen budget – van het systeemontwikkelproject. De meeste businesscases die wij (rond traditioneel ingerichte softwareontwikkelprojecten) tegenkomen kennen deze mate van detaillering van de voordelen niet. Veelal wordt uitgegaan van een geheel werkend systeem, waarna (soms wel gedetailleerd) wordt geschat wat de voordelen zijn uitgedrukt in bijvoorbeeld uitvoeringskosten (bijvoorbeeld bij minder medewerkers) of omzetgroei.

Feitelijk kan de kern van agile softwareontwikkeling ook worden samengevat als een omdraaiing van deze aanpak. In agile softwareontwikkeling wordt gedetailleerd, en zo precies mogelijk, op (klein) functieniveau bepaald wat de waarde is. Wat de kosten zijn van een dergelijke feature wordt pas inzichtelijk als de waarde zo hoog is dat het het overwegen waard is om (op korte termijn) tot implementatie over te gaan.

Naar een andere manier van kijken naar de waarde van software

Het gebruik van agile methoden is al haast gemeengoed geworden en ze worden inmiddels in organisaties ook 'at scale' en bij de grotere en meest complexe projecten toegepast. We zien daarbij in de praktijk dat er, door het gebruik van deze methode, veel beter wordt nagedacht over 'de belangrijkste minimale functionaliteit eerst'. Bij het maken van deze afweging wordt echter nog (te) weinig aandacht besteed aan het waardeaspect: wat levert het realiseren van bepaalde functionaliteit/features daadwerkelijk op omgerekend in harde euro's? Hoewel hiermee wordt gezondigd tegen belangrijke agile uitgangspunten ontbreekt juist deze informatie vaak nog op de *product backlog* (de 'lijst' nog te ontwikkelen systeemfuncties). De product owner beseft vaak niet dat dit ook onderdeel is van zijn takenpakket of heeft niet de kennis en ervaring om dit goed te kunnen invullen. Dit leidt er in veel gevallen toe dat de softwareontwikkeling vaak vanuit een beperkt perspectief, bijvoorbeeld het IT-perspectief van de ontwikkelorganisatie, wordt aangestuurd.

Uiteraard zijn het de uitzonderingen die de regel bevestigen. Organisaties die primair afhankelijk zijn van hun IT-platform voor het genereren van hun inkomsten, gaan hier vaak wel zeer goed mee om. Echter, zeker voor het ontwikkelen van softwareproducten die slechts een beperkte ondersteunende rol bieden bij het primaire proces, is deze inschatting vaak lastig te maken. Maar juist hier kan het toekennen van waarde (bijvoorbeeld door 'efficiëntie' als maatstaf te nemen) ertoe leiden dat er betere beslissingen worden genomen.

Softwarefabrieken

In de jaren negentig kwam er steeds meer aandacht voor de zogenaamde 'software factory'-technieken (zie voor een overzicht [Webe97]). In eerste instantie richtte die zich met name op toolmatige ondersteuning van softwareontwikkelaars, wat uiteindelijk heeft geleid tot de moderne grafische ondersteuning in IDE's (Integrated Development Environments) als Visual Studio en Eclipse. Al gauw kwamen er, via de Continuous Integration-trend, continu draaiende kwaliteitsmaatregelen bij die via automatisch gegenereerde dashboards ook inzichtelijk, en te monitoren, worden (zie figuur 1). En op basis van het versiebeheersysteem was het zelfs mogelijk enig inzicht te krijgen in de (omvang en kwaliteit van de) bijdrage die individuele

medewerkers aan de ontwikkeling van het systeem leverden.

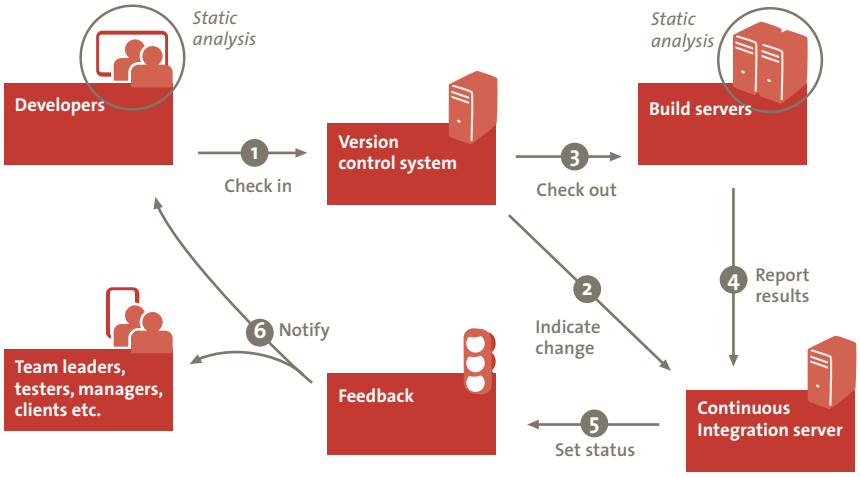
Onder softwareontwikkelaars is er weerstand tegen (een deel van) de geautomatiseerde maatregelen. Hun argument is dat de uitkomsten van dit soort tooling met verstand van zaken geïnterpreteerd dienen te worden, en daarin hebben ze gelijk. Maar als aan deze voorwaarde wordt voldaan, biedt een goed dashboard waarin kerngegevens rond kwaliteit en bijdrage worden verzameld, softwareontwikkelaars juist inzichten die hen in staat stellen hun werkwijze te verbeteren. De weerstand tegen geautomatiseerde hulpmiddelen in een primair IT-proces als softwareontwikkeling is sowieso moeilijk te begrijpen. Als de gegevens in het dashboard, in een agile softwareontwikkelproces, worden aangevuld met een backlog van kleine functionele eenheden (bijvoorbeeld de in een agile omgeving veelgebruikte *user stories*) inclusief de waarde van deze softwarecomponenten, is er feitelijk sprake van een softwarefabriek.

Dan is het ook zinvol om in de managementliteratuur op zoek te gaan naar hoe naar waarde en voortbrengingsprocessen wordt gekeken in andere takken van sport. Dan blijkt dat er veel parallellen zijn te trekken tussen softwareontwikkelingstrajecten en fabricageprocessen zoals Lean Thinking waarin agile methoden hun oorsprong vinden. Wanneer we softwareontwikkeling meer beschouwen als een fabriek waar het produceren van producten (die waarde opleveren) op een zo efficiënt mogelijke manier (optimaliseren van de kosten) plaatsvindt door de inzet van mechanisatie (automatisering), dan kunnen we hier veel van leren.

Een andere in het oog springende filosofie is de Theory of Constraints van Eliyahu Goldratt ([Goldo4]). Ook deze theorie stelt waardecreatie centraal en biedt hier nuttige handvatten voor. Het is ook deze theorie die we, zoals blijkt uit het voorbeeld in het kader 'Case: Combined Computer Solutions', hebben gebruikt als analogie voor het onderzoeken van een mogelijke productiviteitsverbetering in een softwareontwikkelorganisatie.

De Theory of Constraints

De Theory of Constraints (TOC) is een managementfilosofie afgeleid uit de natuurkunde die zich focust op het optimaliseren van de gehele waardeketen. De grondlegger van de TOC is Eliyahu Goldratt. Hij is bij het grotere publiek bekend geworden met zijn roman *The Goal* (*Het doel*) uit

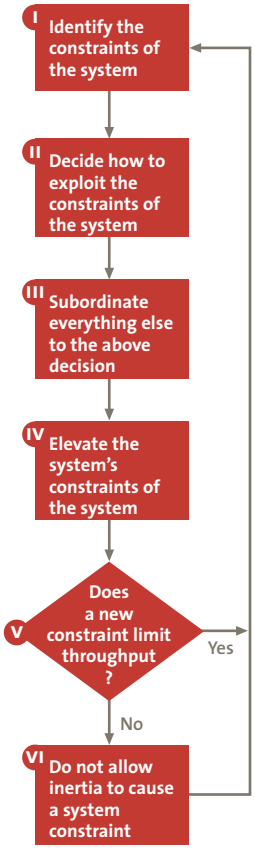


Figuur 1. **Componenten van een software factory waarbinnen Continuous Integration plaatsvindt.**

1984. Hierin zet Goldratt de TOC op een zeer toegankelijke manier uiteen. Later heeft hij nog enkele boeken geschreven over de TOC, zoals *Het Hooibergsyndroom* en de roman *Het is geen toeval*. De TOC komt voort uit het optimaliseren van productieprocessen, maar is niet primair daarvoor bedoeld. In de afgelopen jaren is wel gebleken dat deze theorie veel breder toepasbaar is.

De TOC is met name bekend als filosofie voor het optimaliseren van beperkingen (*constraints*) rondom zogenaamde knelpunten (*bottlenecks*), met als doel het systeem als één geheel te optimaliseren. Wie het boek *The Goal* goed leest, zal echter ontdekken dat de filosofie breder is dan dat. Goldratt wil in zijn boek laten zien dat het met name van belang is te zoeken naar onderliggende oorzaken van problemen en dat 'meten is weten'. Zijn filosofie is gebaseerd op drie aparte maar verbonden thema's:

1. *Prestatieverbetering*. Door het verhogen van de *throughput* en het minimaliseren van de *voorraad* kan de servicegraad (klanttevredenheid) worden verhoogd en worden de productiemiddelen efficiënt gebruikt. Volgens Goldratt is het essentieel dat een onderneming het maximaliseren van de throughput centraal stelt en pas daarna kijkt naar het minimaliseren van de voorraad (en eventueel de operationele kosten). Dit omdat in zijn visie de throughput onbegrensd verbeterd kan worden, terwijl de voorraad en kosten slechts tot nul herleid kunnen worden (maar dan is ook geen throughput meer mogelijk).
2. *Logistiek*. Dit is het bekendste gedeelte van de filosofie van de TOC. Het gaat uit van het feit dat ieder systeem ten minste één bottleneck heeft. Deze processtap dicteert het tempo. Wanneer deze zwakste schakel wordt aangepakt, zal dit tot gevolg hebben dat het systeem als geheel efficiënter wordt. Om te komen tot continue verbetering identificeert Goldratt vijf stappen (zie figuur 2):



Figuur 2. **Vijf stappen naar continue verbetering ([Gold04]).**

I. Spoor het knelpunt (de knelpunten) op.
II. Gebruik het knelpunt zo dat maximale productiviteit wordt bereikt.
III. Voer aanpassingen door in andere processen zodat de snelheid van het knelpunt richtinggevend wordt. (het zogenaamde 'drum-buffer-rope'-principe).
IV. Vergroot de capaciteit van het knelpunt (totdat het geen knelpunt meer is).
V. Is het knelpunt opgelost, begin dan weer opnieuw bij stap I.

Daarnaast stelt Goldratt dat men ervoor moet waken dat inertie de bottleneck van het proces wordt. Het belangrijkste in de TOC is niet het oplossen van lokale inefficiënties: men moet juist het systeem als geheel in ogenschouw nemen.

3. *Logisch denken.* De denkprocessen zijn belangrijk bij het identificeren van het kernprobleem, het identificeren van win-winsituaties en het opstellen en implementeren van verbeterplannen.

Goldratt stelt verder dat 'het doel' van een onderneming niet zozeer is het in dienst hebben van goede mensen, het produceren en verkopen van producten of diensten of het veroveren van de markt, maar dat het belangrijkste doel van een onderneming is om geld te genereren, nu en in de toekomst. Al het overige zijn slechts middelen om het ultieme doel te bereiken. Zo zegt hij ([Gold04]): 'De bestaansreden van een bedrijf ligt in het feit dat men het geïnvesteerde geld wil laten renderen, en dat kan alleen door het realiseren van omzet. Het produceren van voorraad is niet productief, want voorraden zijn per definitie niet verkocht.'

De TOC is geschreven met het oog op toepassing binnen productieorganisaties, maar het is gebleken dat de theorie ook zeer goed toepasbaar is binnen de dienstensector als analogie om naar processen te kijken als waardeketens binnen een organisatie.

Agile ontwikkelen en het toevoegen van waarde

De TOC kan worden 'vertaald' naar de eerder beschreven agile softwarefabriek ([Mura08], [Bail09]). Als voorbereidende stap moet dan kritisch worden gekeken naar de wijze waarop waarde in het proces wordt toegekend. We merkten al op dat product owners dat niet altijd als hun taak zien of daar niet voor zijn toegerust. Een mogelijk gevolg is dat veel inspanning, inclusief voorbereidende activiteiten (= kosten), wordt gestoken in activiteiten die leiden tot producten met weinig tot geen waarde.

Het is, zeker om te beginnen met agile ontwikkeling, niet nodig de waarde in een harde valuta uit te drukken. Een puntenwaarde die wordt gedragen binnen de organisatie voldoet om in ieder geval belangrijke kenmerken van het productieproces, het ontwikkelen van systeemfuncties, te monitoren.

Één van de belangrijke kenmerken volgens de TOC is de voorraad. Heel eenvoudig gezegd is de voorraad in agile softwareontwikkeling alle functionaliteit op de product backlog waarvan de ontwikkeling in voorbereiding is of (zeg vanaf een eerste definitie) is gestart of die is uitontwikkeld maar nog niet is terechtgekomen in een productierelease. Vanuit het perspectief van het verlagen van de voorraad en het sneller toevoegen van waarde zijn trends richting Continuous Delivery en DevOps dan ook goed te begrijpen, aangezien deze methoden primair bedoeld zijn om sneller te kunnen releasen en het mogelijk maken (veel) waarde te genereren uit de gerealiseerde functionaliteit. Zoals figuur 3 illustreert is dit hét voordeel van het gebruik van agile methoden. Het is daarbij niet verbazend dat deze trends verdere automatisering van het IT-voortbrengingsproces vereisen, waarbij zaken als geautomatiseerde testen en continu inzicht in de geleverde kwaliteit essentiële onderdelen zijn.

Een tweede belangrijk kenmerk in de TOC is de throughput, dat zich in eerste instantie vrij gemakkelijk laat vertalen naar het in Scrum al bestaande *velocity*-kenmerk: het aantal storypunten per sprint. Bij throughput komt ook de *lead time*, de tijd die nodig is om een feature uit te werken, om de hoek kijken. In Scrum is daarbij de duur van een sprint een bepalende factor. Vanuit de TOC kan dan ook direct worden gesteld dat een kortere sprintduur, vanuit het perspectief van snel waarde toevoegen, de voorkeur heeft. Dit vereist uiteraard wel dat de gehele organisatie, inclusief eventuele supportafdelingen, zodanig wordt ingericht dat dit mogelijk wordt.

Verbetering van productiviteit van agile softwareontwikkeling

Om vanuit de TOC te werken aan een daadwerkelijke productiviteitsverbetering in softwareontwikkeling kunnen de volgende detailvragen en aanbevelingen, verdeeld over zeven deelgebieden, behulpzaam zijn (vrij naar [Mura08]):

1. Elimineer verspilling

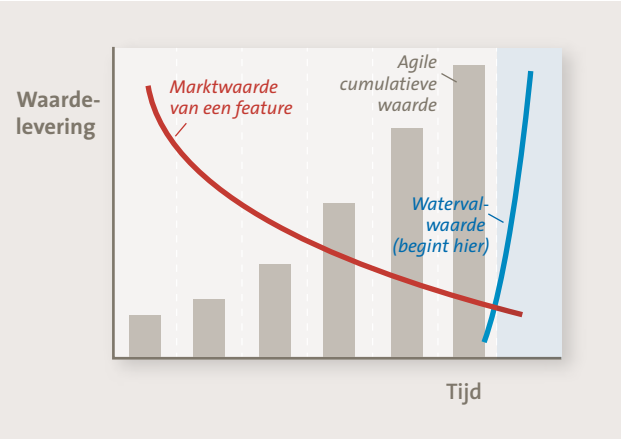
- Worden de belangrijke features gerealiseerd? Zorg voor een goede prioriteitstelling en houd je daar ook aan.
- Worden defects zo snel mogelijk opgespoord? Zodat de verspilling van (laat) herstel wordt voorkomen!
- Zijn doorlooptijden van taken klein? Zorg dat taken snel zijn afgerond, bijvoorbeeld door een architectuurdocument in delen te schrijven. De teams bij elkaar in één ruimte bevordert en versnelt noodzakelijke communicatie.
- Kunnen medewerkers zich op één taak concentreren? Het switchen tussen taken werkt verspilling (tijdverlies) in de hand.
- Is er weinig werk in de pijplijn? Release veel en snel. Features die niet nog zijn geleverd verouderen ook.
- Zijn er wachttijden bij de realisatie van softwareproducten (code, documentatie et cetera)? Waarop moet worden gewacht (op toestemming, resources, hardware et cetera)?
- Zijn er overbodige processen en managementactiviteiten? Doe die dan niet!
- Maak waarde en doorlooptijden van features inzichtelijk (in een dashboard)!

2. Bouw kwaliteit in

- Volg een 'test first'-aanpak waarbij testen (en dus juist ook de regressietest) worden geautomatiseerd.
- Stimuleer het verbeteren van de code! Hiervoor moet de kwaliteit inzichtelijk zijn (door statische analyse in buildprocessen en dashboard mee te nemen).
- Reduceer integratieproblemen. Dit kan door het systeem regelmatig samen te voegen en te testen. Met andere woorden, doe (min of meer) aan Continuous Integration.
- Werk aan discipline in het team. Stel duidelijke regels en instructies op en houd toezicht op de navolging ervan. In de praktijk blijkt dat ludieke straffen (zoals het trakteren op taart) bij belangrijke overtredingen in een softwareontwikkelteam best goed functioneren.

3. Creëer kennis

- Zorg voor snelle feedback (van de juiste personen) zodat het team zich kan aanpassen aan de gewenste (gewijzigde) omstandigheden.
- Bouw in iteraties met korte tijdsduur en voorkom dat er maar één oplevering is aan de eindklant.
- Zorg voor geleide besluitvorming waarin vooraf de opties worden uitgewerkt. Voorkom daarmee discussies waarin besluiten worden voorgesteld, verfijnd en besproken met 'iedereen' totdat consensus wordt bereikt.



Figuur 3. **Waarde toevoegen met software: agile versus waterval.**

4. Stel verplichtingen uit

- Houd opties open en neem zo laat mogelijk besluiten. Voorkom daarmee dat het team zich al vroeg op beperkingen vastlegt.
- Los problemen op vanuit de breedte van het probleem en werk dus niet eerst, in de diepte, één geval uit.

5. Lever zo snel mogelijk op

- Limiteer het werk tot de capaciteit van het team; overladen levert alleen maar extra werk op.
- Zorg voor systemen/processen die het werk voorttrekken zodat niet iedere taak door een manager moet worden toegewezen. Een voorbeeld is een backlog van features waaruit een ontwikkelaar zelf een volgende taak kan kiezen.

6. Zorg voor voldoende bevoegdheden

- Zorg ervoor dat een product owner vanuit de organisatie voldoende mandaat heeft om zelfstandig (en dus sneller) beslissingen te kunnen nemen.
- Zorg ervoor dat er een teamleider is (geen manager) die het team leidt, individuen motiveert en de richting uitzet.
- Laat individuen zelf hun taken kiezen en bepalen hoe ze die uitvoeren (binnen de randvoorwaarden van het project).
- Zorg dat het team (gezamenlijk) de bekwaamheid heeft om de taken uit te voeren.
- Zorg voor betrokkenheid bij het project die leidt tot (aanvullende) motivatie om taken binnen het project op te pakken.
- Beloon het team (nooit individuen).

7. Optimaliseer het geheel

- Voorkom 'lokale' verbeteringen die gebaseerd zijn op (lokale) deelwaarnemingen.
- Zorg voor goede samenwerking en kennisuitwisseling met partners (binnen en buiten het bedrijf) die werken aan dezelfde doelstelling.

Case: Combined Computer Solutions

'Als organisatie stonden we midden 2012 voor een flink aantal uitdagingen', aldus Pieter-Paul van Beek, CEO van Combined Computer Solutions (CCS), een gevestigde speler op het gebied van moderne verzekeringsoplossingen. Het bedrijf heeft bijna 30 jaar ervaring in het ontwikkelen en implementeren van compleet geïntegreerde oplossingen voor assurantiebedrijven op het gebied van polisadministratie, claimsafhandeling, datawarehouse, CRM en dergelijke en telde in 2012 ongeveer 180 medewerkers.

Van Beek: 'Door de snelle ontwikkelingen in de markt, met name door het afschaffen van de provisies voor tussenpersonen, waren we genoodzaakt om snel te veranderen. Op verschillende manieren hebben we daarom onze eigen organisatie kritisch onder de loep genomen. Wat betreft onze afdeling software engineering (SE) – bestaande uit ongeveer 90 medewerkers, zo'n 60 procent van het totaal – kwam binnen de directie al snel de vraag naar boven of de huidige aanpak van productontwikkeling aanknopingspunten biedt voor een sterke verhoging van de productiviteit en daarmee de waardeontwikkeling voor onze klanten. We hebben onszelf daarom een ambitieus doel gesteld: het behalen van een beoogde productiviteitswinst van 20 procent. Dat klinkt fors en dat is het ook. Daarnaast wilden we een grotere klanttevredenheid bereiken.

Van Brummelen en Koedijk adviseerden ons om de Theory of Constraints te gebruiken als analogie om naar onszelf te kijken. We waren gecharmeerd van het gebruik van deze analogie, omdat we niet wilden dat er alleen een dik rapport werd geschreven dat uiteindelijk in een la terecht zou komen, maar dat de uitkomsten van de scan tastbare bevindingen op zouden leveren waarmee we op korte termijn zelf aan de slag zouden kunnen gaan.

Op basis van een beknopt documentenonderzoek en een twintigtal interviews met mensen vanuit SE en afdelingen die veel met SE samenwerken, hebben de onderzoekers een inventarisatie gemaakt van de mogelijke bottlenecks in ons softwareontwikkelingsproces. Daaruit kwam een aantal verrassende conclusies naar voren.

Allereerst werd duidelijk dat we veel werk met weinig waarde aan het uitvoeren waren. Dat kwam doordat we veel verschillende versies van onze software bij klanten draaiden. Hierdoor moesten we kleine verbeteringen en foutherstel in al die versies doorvoeren. Door een rondje te maken langs onze klanten, en hen te overtuigen van de voordelen van een upgrade, was reeds een belangrijk deel van onze beoogde productiviteitswinst bereikt.

Een tweede belangrijke bevinding die naar voren kwam uit het onderzoek was dat op het gebied van productontwikkeling en maatwerk relatief veel 'voorraad' in ons ontwikkelproces zat. Ondanks dat de teams werkten met agile scrum bleek dat we vooraf veel tijd besteedden aan het uitwerken van specificaties en dat lang niet al deze ideeën het haalden om ontwikkeld te worden. Waarna ook nog bleek dat ontwikkelde ideeën niet aansloten bij de klant en daarom niet verkocht werden. Daarnaast was de organisatie veel tijd kwijt met het beantwoorden van vragen van klanten over features die op de planning stonden om ontwikkeld te worden. Dat leverde ook problemen op met betrekking tot het verwachtingsmanagement bij klanten: hoe langer een klant moet wachten op een bepaalde feature, des te lastiger het is om uiteindelijk te moeten uitleggen dat een bepaalde feature voorlopig nog niet ontwikkeld gaat worden. Door onze verkoopafdeling beter richting te laten geven aan SE door inzichtelijk te maken welke producten makkelijk verkocht kunnen worden, konden we nog gerichter te werk gaan, wat (mogelijk) leidt tot een verhoging van het aantal verkochte producten en daardoor zeker leidt tot een verlaging van de voorraad. Door actief te zoeken naar pilotklanten is aan deze aanbeveling effectief invulling gegeven.

Een derde bevinding die de analogie met de TOC naar voren bracht was met name gericht op het onderdeel service en onderhoud binnen SE. Een deel van dit team werkt met kanban (een agile beheermethode). Ook hier speelde een voorraadprobleem – nu doordat prioriteiten van klantproblemen niet goed werden toegekend. De oorzaak daarvan was gelegen in klantvriendelijkheid: we wilden heel erg graag leveren wat de klanten vroegen. Maar doordat het proces nog weinig inzichtelijk was, hadden klanten hier toch last van – zeker als er iets echt belangrijk was. Door het intakeproces, en de transparantie door het voorspellen van een *estimated time of arrival* (ETA), te verbeteren hebben we dit opgelost. Het gevolg was grotere klanttevredenheid, minder wachten op oplossingen en het verdwijnen van de bottleneck.

Ten slotte konden we ook op het gebied van innovatie nog het een en ander optimaliseren. De algehele bevinding was dat onze time-to-market groot was. Dit was een bevinding die we zelf ook al hadden onderkend. Ook hier was het advies om de voordelen van agile beter in te zetten en bijvoorbeeld sneller met pilotklanten aan de slag te gaan, zodat deze ook daadwerkelijk betrokken zouden raken bij het ontwikkelproces en de nieuwe producten daardoor beter aansluiting konden vinden bij deze klanten.

Omdat we door dit alles op een andere manier tegen onze organisatie aankeken, namelijk als keten van activiteiten die samen de bedoeling hebben om snel waarde toe te voegen, zijn we een aantal zaken structureel anders gaan aanpakken. Dit leidde uiteindelijk tot een productiviteitswinst van meer dan 30 procent.'

Conclusie

Hoewel in de literatuur over softwareontwikkeling weinig wordt gesproken over de waarde van software, is het bij de ontwikkeling van software van groot belang om de waarde van de features te bepalen. Alleen op basis van deze waardebepaling is zinvolle sturing binnen ontwikkelteams mogelijk.

Het gebruik van uitgebreide Continuous Integration-technieken tijdens agile softwareontwikkeling biedt handvatten om inzicht te krijgen in de kwaliteit en voortgang van de softwareontwikkeling. Deze beweging geeft, met de waardebepaling van features, tevens de mogelijkheid de softwareontwikkeling zelf in termen van fabrieksprocessen te beschrijven.

De Theory of Constraints (TOC) biedt verrassende inzichten in de wijze waarop fabrieksprocessen moeten worden ingericht. Hierbij ligt de nadruk op de throughput en op het versnellen van de waardecreatie. Een kenmerk van deze theorie is ook 'meten is weten': door het SMART meten van productiefactoren als throughput kan worden aangetoond dat maatregelen werken (of niet).

In dit artikel en de casebeschrijving wordt aangetoond dat de TOC goed werkt voor agile softwarefabrieken. Met deze methode kunnen de processen worden geoptimaliseerd. In de beschreven case komt dit duidelijk naar voren.

Literatuur

[Amor13] Dr. J.M. Amoraal, dr. G. Lanzani, drs. P. Kuiters en drs. J.M.A. Koedijk CISA CISM, *Grip op de kwaliteit van software*, Compact 2013/2.

[Bail09] D. Bailey, *The Theory of Constraints: Productivity Metrics in Software Development*, 2009.

[Bro095] F.P. Brooks Jr., *The Mythical Man-Month: Essays on Software Engineering*, 1995.

[Gink03] Drs. R.M. van Ginkel RA en drs. A.J. van de Munt RA, *Activering van zelfontwikkelde software en websites in jaarrekeningen van Nederlandse ondernemingen vanaf 2005*, Compact 2003/2.

[Gold04] E.M. Goldratt, *The Goal* (revised 3rd edition), 2004.

[Gro012] J. de Groot en J. Visser, *De waarde van softwareproducten bepalen*, de IT-Auditor nr. 1, p. 24-31, 2012.

[Jone11] C. Jones and O. Bonsignour, *The Economics of Software Quality*, 2011.

[Kell97] H.J.M. Keller RI, *Waardebepaling van software*, INTRAKT Informatica, Utrecht, 1997.

[Mura08] A. Murauskaite and V. Adomauskas, *Bottlenecks in Agile Software Development Identified Using Theory of Constraints (TOC) Principles*, IT University of Gothenburg, 2008.

[Webe97] H. Weber (ed.), *The Software Factory Challenge*, 1997.

Over de auteurs

Drs. J. van Brummelen is manager bij KPMG Advisory N.V. Hij is op dit moment projectleider in een softwareontwikkeltraject bij een grote multinational. Hij is gespecialiseerd in Agile Design & Development en houdt zich onder andere bezig met procesinnovatie door de inzet van agile methoden.

Drs. J.M.A. Koedijk CISA CISM is partner bij KPMG Advisory N.V. en is al jaren actief op het gebied van software-engineering en internetstandaarden. Hij heeft veel ervaring met review, ontwerp, ontwikkeling en implementatie van complexe IT-systemen en geeft momenteel leiding aan de Software Quality-praktijk van KPMG. Zijn expertisegebieden omvatten softwarekwaliteit, webapplicaties, gegevensverwerking, reliable messaging en Service Oriented Architecturen.