Methodology

Ondanks dat ik inmiddels al zo'n twintig jaar in dit vakgebied rondloop, blijven software development projecten uitdagingen opleveren. Zo werkte ik eerder dit jaar als coach aan een agile project waarmee een grote logistieke onderneming nieuwe producten voor hun klanten ontsloot.

Projecten beter beheersbaar houden

Serviceoriëntatie met smart use cases

Alhoewel ik vooral als coach was ingezet op het begeleiden van het proces, was het project technologisch een hele kluif. Een Java portal als front end, orchestratie van processen in SAP middleware, back office systemen in PeopleSoft, SAP en .NET die moesten worden 'verserviced', en niet te vergeten een aantal interfaces naar externe systemen, waar we niet of nauwelijks invloed op konden uitoefenen. Al met al een typisch voorbeeld van een service georiënteerd project. Dit type projecten kenmerkt zich door hun hoge complexiteit, en zijn daarom slecht in te schatten en te plannen. Alhoewel niet de oplossing voor alle uitdagingen, bestaat er een techniek die smart use cases wordt genoemd, die concreet en pragmatisch bijdraagt aan het beter inschatten, plannen, maar ook realiseren en testen van servicegeoriënteerde projecten.

Complex projecten

Iedereen die in dergelijke servicegeoriënteerde projecten heeft gewerkt, weet het inmiddels wel: service georiënteerde architecturen maken het uitvoeren van dit soort projecten er niet makkelijker op. Dit soort projecten kenmerken zich een groot aantal organisatorisch, functioneel en technisch complicerende factoren:

- Diverse stakeholders. Servicegeoriënteerde projecten automatiseren in het algemeen bedrijfsprocessen, die meestal meerdere organisatorische eenheden en verschillende systemen doorsnijden. Dientengevolge zijn er vaak diverse stakeholders betrokken bij het project.
- Flexibele front end technologieën. Voor het realiseren of uitbreiden van de front end(s) wordt vaak gebruik gemaakt van state-of-the-art, of zelfs bleeding edge technologie, waarover niet altijd evenveel kennis beschikbaar is in de organisatie.
- Heterogene back end technologien. De diverse systemen die ontsloten dienen worden – verserviced – zijn gerealiseerd in allerhande technologieën, al dan niet verouderd.
- Externe interfaces. Niet zelden wordt ook gebruik gemaakt van systemen die extern zijn aan de organisatie. Op deze systemen kan het project meestal weinig invloed uitoefenen, hooguit zijn de interfaces bekend. Helaas zijn dit soort interfaces zelden stabiel, en aan wijzigingen onderhevig.
- Diverse rollen. Opvallend aan service georiënteerde projecten is dat er een breed scala aan rollen betrokken is. Zo zorgt de diversiteit aan technologie al voor een breed palet aan developers – zo is een SAP CRM consultant geen SAP XI developer, en een .NET developer niet per se ook een software architect. Maar ook andere traditionele en minder traditionele rollen al informatieanalisten, ontwerpers, user inferface designers, enterprise architecten, business consultants en last but not least project managers, vinden hun weg in dit type projecten.

Lastig te plannen

Een recent voorbeeld is een project voor een verzekeraar voor het realiseren van een nieuw hypotheekoffertesysteem. Belangrijkste streven bij dit project was het realiseren van straight-through processing (het zonder humane tussenkomst doorvoeren van hypotheekoffertes naar de back office) en het flexibel kunnen toevoegen van nieuwe hypotheekproducten.
Naast de haast traditionele handicaps als schuivende requirements, bleek gedurende het project dat

Sander Hoogendoorn
is principal technology officer bij Capgemini.

Een servicegeoriënteerd project kent vele stakeholders.

de interfaces van externe systemen niet aansloten op de gegevens die nodig waren om de benodigde processen uit te voeren. Daarnaast bleek ook de gebruikersvriendelijkheid van de (gegenereerde) webpagina's tot hoofdbrekens te leiden. Het project wordt nu hopelijk ruim een jaar na dato opgeleverd en heeft ruim vier keer zoveel gekost als oorspronkelijk begroot.
Maar waarschijnlijk vertel ik u als lezer hiermee nog niets nieuws. Lastig is wel dat deze complexiteit voor uw project een aantal ondermijnende gevolgen kan hebben. Om u een indruk te geven laat ik er hier toch maar een paar de revue passeren:

- *Externe afhankelijkheden.* Servicegeoriënteerde projecten zijn vrijwel zonder uitzondering afhankelijk van derden, ofwel partijen binnen de eigen organisatie die andere planningen er op nahouden, ofwel partijen buiten de eigen organisatie, die lastig beïnvloedbaar zijn;
- *Moeilijk in te schatten.* Wie al eens in een offertetraject voor een dergelijk project heeft geparticipeerd; het maken van een goede schatting is cruciaal maar extreem lastig;
- *Moeilijk te plannen.* Lastige schattingen, diverse stakeholders, externe afhankelijkheden, en talrijke projectrollen maken het er niet makkelijker op voor de beoogd project manager om een goed planning voor het project af te leveren.

Een ding is zeker: een traditionele lineaire aanpak, waarin flexibiliteit tot een minimum wordt beperkt, is uit den boze. Een service georiënteerd project kent vele wendingen – steeds meer overigens naarmate het project langer duurt.
Het zal u niet verbazen dat met deze consequenties service georiënteerde projecten te vaak niet op tijd en op budget worden opgeleverd, of dat gaandeweg het project wordt besloten slechts een deel van de vereiste functionaliteit op te leveren. Tenslotte worden de beoogde uitgangspunten voor het project vaak niet gehaald. Denk daarbij aan het in service ontsluiten van legacy back end systemen, het realiseren van hergebruik ten einde toekomstige projecten sneller te kunnen uitvoeren, het opbouwen van een repository van services of het snel en flexibel kunnen toevoegen van nieuwe producten en features.

Inschatten complexiteit

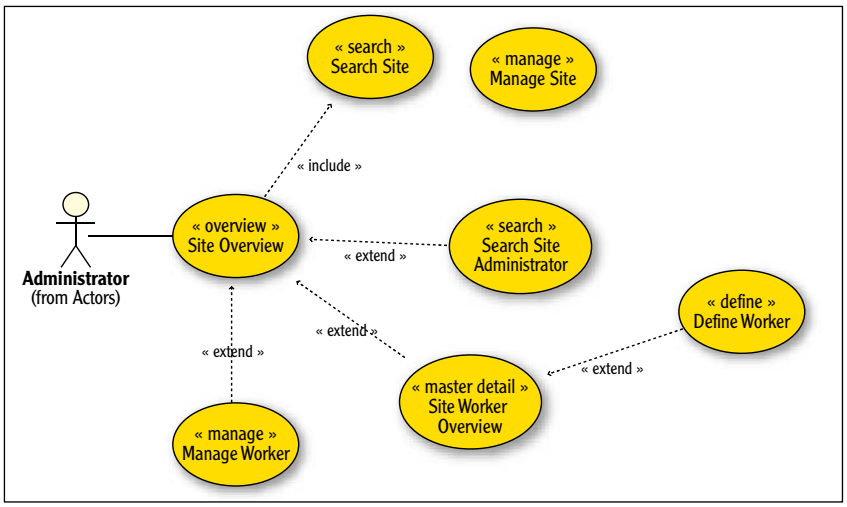
Zoals gezegd zijn servicegeoriënteerde projecten meestal lastig te schatten en derhalve nog lastiger te plannen. Een belangrijk vraag die bij aanvang van dit soort projecten dan ook dient te worden gesteld om een goede kosten-batenanalyse te kunnen maken is: hoe groot en complex is ons project nu eigenlijk?
Een belangrijk struikelblok bij het beantwoorden van deze op het oog eenvoudige vraag is dat er allerlei verschillende typen deliverables zijn te onderkennen in een service georiënteerd project. Denk hierbij bijvoorbeeld maar aan webpagina's, schermen, services, orchestratie, domeinobjecten, transacties. Dit maakt het "tellen" van de totale complexiteit van een project lastig. En zelfs wanneer al deze verschillend typen deliverables zouden kunnen worden geteld, dan nog ontbreken er de nodige ervaringscijfers, die de uiteindelijke complexiteit per type deliverable in uren kunnen helpen vertalen.

Het zou een belangrijk stap voorwaarts voor servicegeoriënteerde projecten kunnen zijn wanneer alle (functionele) deliverables op een en dezelfde schaal konden worden uitgedrukt, waarbij de complexiteit van zowel de webpagina's, schermen, orchestratie als services op eenzelfde manier wordt uitgedrukt, middels een enkele eenheid van schatten, en misschien zelfs wel een enkele eenheid van werk. Immers, zo kunnen makkelijker ervaringscijfers worden verzameld, die weer kunnen worden toegepast om volgende projecten binnen hetzelfde programma beter in te plannen. Deze eenheid van werk dient zich aan door smart use cases te introduceren in service georiënteerde projecten.

Use cases

Laat ik bij het begin beginnen. Wat zijn eigenlijk use cases? Meestal wordt een use case aangemerkt als een beschrijving van het gedrag dat software

Het is een belangrijke stap vooruit als alle deliverables op dezelfde schaal kunnen worden uitgedrukt.



Voorbeeld van een use case diagram.

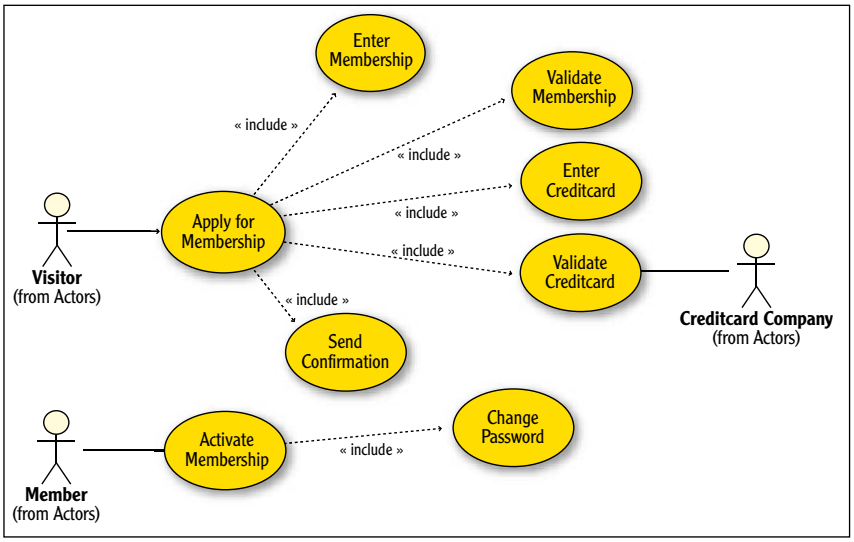
Use cases helpen niet alleen om projecten in te schatten, maar zijn ook een eenheid van werk.

vertoont als reactie op een verzoek dat stamt van buiten de software. Vrij vertaald beschrijft een use case wie met de software wat kan doen. Een use case realiseert een bepaald doel voor uitvoerder van de use case. In jargon wordt deze uitvoerder wel de actor genoemd. Zo'n actor hoeft overigens niet perse menselijk te zijn, maar kan bijvoorbeeld ook andere software zijn of zelfs een batchverwerking. Het uitvoeren van de use case vindt plaats als een opeenvolging van een aantal stappen. Deze stappen vormen tezamen een of meerdere scenario's die uitmonden of in het realiseren van het doel, of, wanneer er iets mis gaat, in een uitzonderingssituatie. Deze eindresultaten worden wel de postcondities van de use cases genoemd. Use cases worden veelal beschreven in documenten, al dan niet geformatteerd in een voorgeschreven sjabloon.

Daarnaast bestaat er een aardige, relatief informele en veel toegepast modelleertechniek, waarin actor als poppetjes worden gerepresenteerd, en use cases als ovaaltjes. Deze modelleertechniek, die het use case diagram wordt genoemd, wordt ondersteund door een breed scala aan geautomatiseerde gereedschappen.

Traditionele use cases

Traditioneel worden reguliere use cases gebruikt voor het beschrijven van de functionele requirements van een regulier project, uitgaande van het doel dat de gebruiker (of actor) wil bereiken door de use case uit te voeren. Traditioneel worden use cases beschreven in een document, dat naast het doel, het resultaat ook alle mogelijke scenario's beschrijft aan de hand waarvan de use case kan worden doorlopen.
Een interessant voorbeeld van een dergelijk use case kwam ik tegen in een service georiënteerd project bij een internationale bank, waar een use case Wijzigen Adres een document van vijfenzestig pagina's besloeg, waarin twaalf schermen voorkwamen, tientallen scenario's en waarin diverse services werden aangeroepen. Een ander gelijksoortig voorbeeld is een use case Uitvoeren Kwartaalfacturering van een pensioenfonds, waarbij circa 1.3 miljoen factuurregels werden berekend.
Alhoewel dergelijke use cases in service georiënteerde projecten meestal de te realiseren bedrijfsprocessen beschrijven, zijn ze niet bijzonder geschikt als eenheid van werk en van schatten in dit type projecten. Traditionele use cases verschillen te veel in omvang en complexiteit, om ze met een gerust hart op een lineaire schaal uit te drukken. Ter ondersteuning: het realiseren van de use case Uitvoeren Kwartaalfacturering vergde ruim anderhalf jaar, terwijl het in hetzelfde project slechts enkele dagen kostte de use case Inloggen Gebruiker in te voeren.
Daarbij komt nog het bezwaar dat deze wijze van benaderen van use cases niet bijdraagt aan het identificeren en realiseren van de gewenste services, en dat hergebruik slechts sporadisch en incidenteel tot stand komt.



In dit use case diagram zijn de use cases Apply for Membership en Activate Membership sea nieuws use cases, de overige use cases zoals Send Confirmation en Validate Creditcard zijn fish niveau.

Smart use cases

Toch hebben we in de afgelopen jaren ervaren dat use cases een heel nuttig vehikel kunnen zijn in servicegeoriënteerde projecten. En niet alleen om projecten in te kunnen schatten, maar ook als eenheid van werk, en zelfs voor het identificeren en hergebruiken van services. Om deze nobele doelstellingen te bereiken maken we intensief gebruik van wat we in de loop der jaren smart use cases zijn gaan noemen – genoemd naar de agile methodiek Smart en goed beschreven in mijn boek Pragmatisch modelleren met UML. Een korte introductie.
Alistair Cockburn, autoriteit op het gebied van use cases, beschrijft in zijn boek Writing effective use cases een model waarin use cases in vijf verschillende niveaus van granulariteit – zeg maar grootte en complexiteit – worden uitgedrukt.

Deze niveau's zijn:

- *Cloud.* Use cases op dit hoogste niveau representeren veelal groepen van samenhorende bedrijfsprocessen, zoals Verkopen Producten.
- *Kite.* Op dit tweede niveau worden in de regel individuele bedrijfsprocessen geposteerd, vaak workflow georiënteerd.
- *Sea.* Dit is het niveau waar het om draait, en waar bijvoorbeeld ook Wijzigen Adres toe behoort. De kernvraag op dit niveau zou kunnen zijn: hangt de performance van mijn organisatie af van hoeveel van deze kan ik per dag uitvoeren? Als richtlijn: deze use cases komen nagenoeg overeen met wat ook wel als elementaire bedrijfsprocessen wordt geduid.
- *Fish.* Use cases op fish niveau worden gebruikt

om op zichzelf staande, maar aan sea niveau ondersteunende functionaliteit te modelleren. Denk bijvoorbeeld aan Ophalen Abonnement of Selecteren Product.
- *Clam.* Soms modelleren projecten te ver door, en worden de use cases te klein. Hier spreekt men over clam niveau.

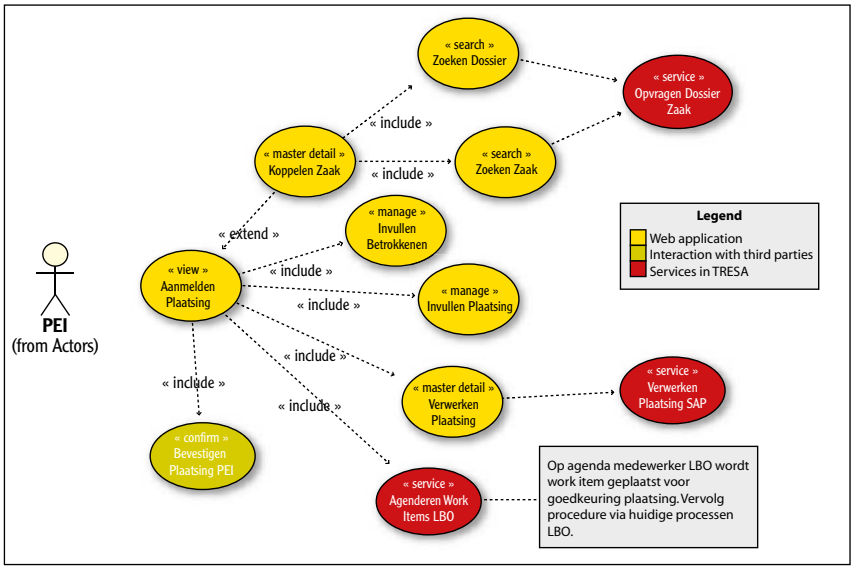
Ondanks de beknoptheid van deze introductie, moge het duidelijk zijn dat cloud en kite niveau in servicegeoriënteerde projecten met name terug te vinden zijn in de te automatiseren bedrijfsprocessen, de visie van het project en wellicht de bedrijfsarchitectuur van de organisatie.
De use cases op het sea en fish niveau worden samen smart use cases genoemd. Deze twee niveau's vormen zo een goede basis van gelijke granulariteit zowel voor het uitdrukken van functionaliteit die direct zichtbaar is voor de gebruiker, zoals Selecteren Product, maar ook voor het identificeren van services, en het aanleggen van een repository van deze services, zoals Ophalen Abonnement.

Smart use cases modelleren

Veel meer dan in traditionele use cases het geval is, gebruiken we voor het vormgeven van smart use cases de bijbehorende modelleertechniek.
Uitgaande van de lijst te realiseren bedrijfsprocessen op kite niveau, modelleren we een use case diagram per elementair bedrijfsproces, op sea niveau derhalve.

Ieder use case diagram bevat:
- Sea niveau use case. Een enkele use case die het elementair bedrijfsproces representeert, weergegeven als een ovaal.
- Actoren. Een of meerdere partijen die gebruik maken van deze use case, om daar hun doel mee te bereiken, weergegeven als poppetjes.
- Fish niveau use cases. Nul, een of meerdere use cases die ondersteunend zijn aan de sea niveau use case, eveneens weergegeven als ovalen.
- Onderstaande figuur bevat een goed voorbeeld van een dergelijke use case diagram.

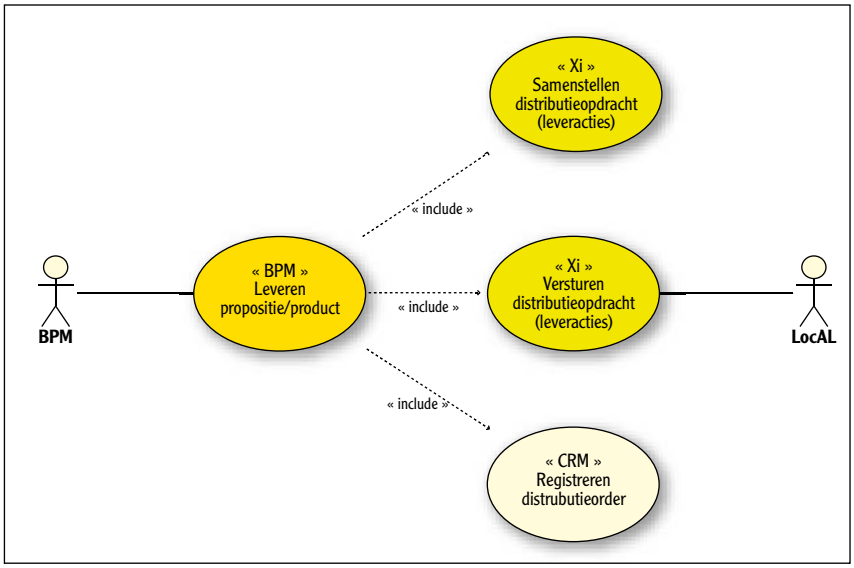
Om tot dit diagram te komen distilleren we normaliter de fish niveau use cases uit de sea niveau use case, door onszelf steeds opnieuw te vragen: wat is de volgende stap die we nemen bij het uitvoeren van deze use case? In sommige gevallen is het handig om deze stappen als individuele use cases op te nemen in het diagram, in andere gevallen weer niet. We hanteren hiervoor een set aan richtlijnen, zoals aan het identificeren van hergebruik – komen we deze use case mogelijk vaker tegen? Of denk het afhandelen van alle interactie rond een formulier of scherm, of het importeren en exporteren van bestanden.



Use case diagram dat de interactie met de gebruikers modelleert tot aan de services.

Standaardtypen smart use cases

Deze hierboven beschreven aanpak is inmiddels – met veel succes – toegepast in een diverse typen projecten, zoals reguliere webprojecten in .NET of Java, het implementeren van portals, business intelligence en zelfs pakketimplementaties.
Bij het uitvoeren van al deze projecten is vooral opgevallen dat steeds opnieuw dezelfde soorten smart use cases de revue passeren. Denk hierbij aan het selecteren, zoeken, of onderhouden van gegevens, het exporteren van een bestand, of het aggregeren van gegevens in een business intelligence situatie.
Deze standaardtypen use cases (in de wandelgangen stereotypen genoemd) helpen projecten enorm om de analyse van de requirements te versoepelen en standaardiseren. Het jargon van smart use cases wordt tijdens workshops al snel overgenomen door opdrachtgever en gebruikers. "Ik stel voor dat hier een master-detail over klanten en orders gebruiken, met een search op producten."



Use case diagram dat het uitvoeren van de service Leveren Product modelleert.

Smart use cases zijn een goede eenheid om complexiteit van een project te meten.

Belangrijk bijkomende voordelen van het toepassen van deze (of eigen) stereotypen is dat ook het schatten van de omvang van het project vergemakkelijkt, en dat ook de realisatie, inclusief testen, van dit soort standaard use cases eenvoudiger is. Immers, er is al eens eerder met hetzelfde bijltje gehakt. Een onderhouds-use case voor klanten verschilt niet wezenblijk van een onderhouds-use case voor producten.

Modelleren in service-georiënteerde projecten

Ook in servicegeoriënteerde projecten hanteren we sinds enkele jaren smart use cases. Daarbij brengen we graag een tweedeling aan, we modelleren een front end use case diagram, en een service use case diagram.
Allereerst modelleren we een smart use case diagram vanaf de sea level use case die de interactie met de gebruikers en het elementaire bedrijfsproces vertegenwoordigt, zoals Aanvragen Subsidie. Daarbij identificeren we de bijbehorende fish level use cases, maar alleen tot aan het aanroepen van de services of de orchestratie.

In dit voorbeeld zijn alle gele use cases client facing, en representeren de rode use case de interactie met de service verlenende systemen.
Daarnaast modelleren we een tweede smart use case diagram, met de desbetreffende service, en alle achterliggende services als fish level use cases. In dit diagram beschrijft de sea level use case in het algemeen de orchestratie, en representeren de achterliggende fish level use cases de individuele services.

In bovenstaand diagram wordt de orchestratie verzorgd door de use case Leveren Product (via SAP middleware), en leveren de overige use cases services die door door Leveren Product worden geconsumeerd. In feite is deze use case een samengestelde service, die (letterlijk) wordt hergebruikt in het front end use case diagram, vergelijkbaar met de rode use cases in het voorgaande voorbeeld.
Het toepassen van een dergelijke aanpak biedt zeer concrete voordelen:

- *Eenduidig.* Er is een eenduidige manier voor het beschrijven en modelleren van alle functionele onderdelen van het project, inclusief de schermen, orchestratie en services.
- *Gelijke granulariteit.* Wanner het project de hierboven beschreven richtlijnen correct toepast, hebben alle use cases een vergelijkbare granulariteit – ze zijn allemaal ongeveer even groot, of zo u wilt, even klein.
- *Technologie-onafhankelijk.* Smart use cases zijn te modelleren onafhankelijk van de platforms en technologie die wordt gebruikt. Bovenstaande voorbeelden kunnen evengoed in .NET en SAP worden gerealiseerd, als in Java en Oracle.
- *Inschatbaar.* De smart use cases, met daarbij de onderkende stereotypen, maken een goede eenheid voor het inschatten van servicegeoriënteerde projecten. Hiervoor is een eenvoudige schaal gedefinieerd die wordt uigedrukt in smart use case punten.
- *Goede eenheid van werk.* Tenslotte, smart use cases vormen een goede eenheid van werk in projecten. In de regel is iedere use case los van de andere te ontwerpen, te ontwikkelen en vooral: individueel te testen. Met name dit laatste aspect zorgt ervoor dat de acceptatie van de te ontwikkelen en hergebruiken componenten vergemakkelijkt.
- *Hergebruik.* Omdat smart use cases technologie-onafhankelijk zijn te modelleren en te beschrijven, biedt de techniek de mogelijkheid om een repository aan te leggen van gebruikte services, uitgedrukt in smart use cases.

Schatten met smart use cases

Omdat smart use cases zo eenduidig en technologie-onafhankelijk beeld geven van de functionaliteit van een project, is het een goede eenheid om de complexiteit en omvang van het project in te schatten. Hiervoor is een eenvoudige schaal gedefinieerd; de meetlat waarlangs alle smart use cases in een project wordt gelegd. Deze schaal is als volgt gedefinieerd:

- 1 - Piece of cake. Eenvoudige use cases, zoals selecties uit standaardlijstjes, of eenvoudige onderhoud, zoals Onderhouden Contracttypen.
- 2 - Moderate. Reguliere use cases, bijvoorbeeld voor selecties uit iets uitgebreidere lijstjes. Denk hierbij aan Selecteren Account Manager per Regio.
- 3 - Average. De gemiddelde complexiteit, die meestal wordt gegeven voor regulier onderhoud, zoeken, of eenvoudige rapportages. Voorbeelden zijn Zoeken Boek, of Beheren Klant. Dit is tevens de default complexiteit.
- 4 - Hard. Wordt gegeven bij lastiger onderhoud of rapportages. Ook eenvoudige services, meestal zonder het wegschrijven van gegevens, krijgen vaak deze complexiteit. Denk aan Ophalen Contract of Overzicht Orders per Klant.
- 5 - Very difficult. Use cases met deze complexiteit worden als lastig beschouwd. Meestal zijn dit uitgebreide rapportages, grafieken, al dan niet met drill-down, of complexere services waarbij gegevens worden weggeschreven. Denk aan Vastleggen Abonnement.
- 8 - Extreme, but known. Deze complexiteit wordt bewaard voor de zeer lastige use cases. Het betreft hier bijvoorbeeld complexe berekeningen of samengestelde services die vaak het uitvoeren van een proces voor hun rekening nemen. In business intelligence projecten wordt

deze complexiteit vaak gegeven aan use cases die datatransformaties uitvoeren. Denk maar aan Berekenen Prepensioen.
- 10 - Extreme and unknown. Deze hoogste complexiteit wordt uitgedeeld om twee redenen. Enerzijds voor use cases die bewezen lastig zijn, zoals het importeren en exporteren van bestanden en berichten. Anderzijds geven we deze complexiteit op het moment dat we wel weten dat de use case complex is, maar nog niet goed kunnen definiëren hoe complex dit precies is. In deze gevallen verplichten we ons deze use case nog nader onder de loep te nemen. Vaak komen hier nog additionele use cases uit naar voren.

Met behulp van deze snel toepasbare schaal, en de vele standaardtypen use cases die we in veel projecten zijn tegengekomen en die al eerder langs deze maatlat zijn gelegd, kunnen we snel en betrouwbaar de complexiteit van een project inschatten. Dit maakt smart use case schattingen herhaalbaar, en bovendien kunnen er gemakkelijk ervaringscijfers uit worden gedistilleerd voor komende projecten. Dit laatste is met name in organisaties die servicegeoriënteerd gaan werken belangrijk, omdat daar meerdere automatiseringprojecten uitvoeren binnen eenzelfde programma. “Ons huidige project beslaat 298 smart use case punten.”
Niet zelden komen dit soort schattingen tot stand tijdens workshops, waaraan alle betrokken partijen deelnemen, zoals de klant, de gebruikers, de architect, analisten, ontwikkelaars en ook de testers.

Tijdens dergelijke workshops krijgt het (toekomstige) projectteam een goed en vooral eenduidig beeld van het project en de complexiteit van de requirements. Met name voor de klant, die niet altijd een reëel beeld heeft hoe complex de realisatie van zijn of haar wensen eigenlijk is. “Oei. Dit wordt toch veel lastig en duurder dan we hadden verwacht,” is een veel gehoorde uitspraak, “misschien moesten we eens onderzoeken of er alternatieven zijn.”

Verlengen voordeeluren

Vooruit dan. Om mijn bovenstaande beweringen kracht bij te zetten een laatste anekdote. Tijdens een recent project bij een grote openbaar vervoersonderneming realiseerden we een aantal bedrijfsprocessen rondom een nieuw type abonnement. Deze processen waren uitgemodelleerd in smart use case diagrammen, zowel voor de front end als voor de services.
Ongeveer halverwege het project, terwijl we inmiddels al een deel van de processen hadden geïmplementeerd als smart use cases, kwam de opdrachtgever schoorvoetend met een verzoek. “Zouden we, als we toch ook dit nieuwe type abonnement implementeren, ook eens kunnen kijken naar het verlengen van de voordeelurenkaarten?” Bijna verschoot de opdrachtgever hierbij van kleur.

Workshop waarin de complexiteit van smart use cases door het team wordt geschat.

Vrijwel onmiddellijk organiseerde de projectleider een workshop waarin het verlengen van voordeelurenkaarten onder de loep werd genomen. Na twee uur modelleren bleek dat bijna het gehele proces was te realiseren door hergebruik van smart use cases die we al hadden. We hoefden slechts vier nieuwe smart use cases toe te voegen, met een geschat totaal van 16 smart use cases punten. Het behoeft denk ik verder weinig uitleg dat de klant snel besloot het verlengen van de voordeelurenkaarten toe te voegen aan het project. Dit is hergebruik ten top.

Inzicht en overzicht

Servicegeoriënteerde projecten kunnen, met name in grote organisaties, enorm bijdragen aan de effectiviteit en efficiëntie van de bedrijfsvoering. Een mooi voorbeeld hiervan maakte ik bij een bank mee. Wanneer een klant van deze bank het onzalige besluit nam om te verhuizen, moest dit oorspronkelijk in 36 verschillende systemen worden vastgelegd. Door steeds meer van deze systemen met behulp van services te ontsluiten, werd dit ogenschijnlijk eenvoudige bedrijfsproces in de loop van enkele jaren en projecten teruggebracht naar een enkele administratieve handeling.
Servicegeoriënteerde projecten zijn echter ook zeer complex van aard. Bovengenoemde bank had een aantal projecten nodig om de beoogde verandering te bewerkstelligen. Alhoewel smart use cases, en de bijbehorende standaardtypen en schattingstechnieken natuurlijk niet alle problematiek van dit soort projecten tot nul reduceert, dragen ze zeer concreet bij aan het inzichtelijk maken van deze complexiteit, door een eenduidige eenheid van werk te introduceren, zowel voor de front ends, als voor de achterliggende orchestratie en services. Als klapper op de vuurpijl is deze uniforme eenheid van werk ook nog eens naar behoren in te schatten, en onafhankelijk te testen. Als ik het In een notendop zou moeten samenvatten: inzicht en overzicht. Precies dat wat veel servicegeoriënteerde projecten ontberen. «

Inzicht en overzicht is wat veel projecten ontbeert.