# AGILE & REQUIREMENTS

## JUST IN TIME & JUST ENOUGH

Praktische inzichten van de werkvloer

DiVetro
analyse • sourcing • management

# COLOFON

Dit e-book is een publicatie van DiVetro, **onafhankelijk strategisch adviesbureau** voor IT-gerelateerde vraagstukken op het gebied van Analyse, Sourcing en Management. DiVetro is gevestigd in Arnhem. Wij voeren vooral veel projecten uit bij de (semi-)overheid, in de gezondheidszorg en in het onderwijs.


### Auteurs

Dennis Geluk
Marco Balvers
Richard Hilekes
Rob den Hollander
Ronald Koenis
Selmar van Egmond
Stanley Lachman



### Uitgever

DiVetro




# INHOUDSOPGAVE

# INLEIDING

Laten we maar meteen met de deur in huis vallen: agile requirements bestaan niet, agile werken wel. Agile is een denkwijze, geen voorschrift. Agile legt de nadruk op principes zoals ‘Just in Time’ en ‘Just Enough’ en wil voorkomen dat er tijdens een ontwikkelproces ‘verspilling’ ontstaat.

Wij zijn als analisten bij DiVetro elke dag met requirements en agile werken bezig. In dit e-book delen wij graag een aantal praktische inzichten om ‘Just in Time’ requirements te maken die ‘Just Enough’ informatie bevatten.

Zo vindt iedereen op het juiste moment precies de informatie die hij zoekt. En dat werkt wel zo prettig - en efficiënt.

In dit e-book beschrijven we achtereenvolgens:

- wat agile werken voor ons betekent;
- hoe u 'just in time' en 'just enough' in uw requirements kunt verwerken;
- hoe u altijd de meest geschikte requirements heeft;
- hoe u verleden, heden en toekomst in uw requirements onderscheidt;
- hoe UC2.0 u helpt om invulling te geven aan 'just in time’ en ‘just enough’.

Veel leesplezier!

DiVetro
analyse • sourcing • management

# 1. Agile requirements bestaan niet. Agile projecten wel.

Stel, u kijkt naar de code en de testcases van bedrijfskritische software die twee jaar geleden succesvol werd opgeleverd.

Zou u kunnen zien of de code en de testcases agile of niet agile ontwikkeld werden? Waarschijnlijk niet. Hetzelfde geldt wellicht voor 'agile requirements'.

Uiteraard zou u aan de diverse werkproducten kunnen zien dat de diepgang op specifieke plekken afwijkt van wat u bij een traditioneel project zou verwachten.

Denk onder meer aan extra documentatie en commentaar bij complexe codefragmenten of meer details in de requirements en de testcases. Op basis van deze variatie in diepgang zou u wellicht kunnen afleiden dat de software agile ontwikkeld werd.

Maar toch… het verschil tussen agile en meer traditionele requirements zal niet altijd duidelijk zijn. Er zijn immers diverse andere factoren die van invloed kunnen zijn op de documentatie bij code, testcases en de requirements binnen uw project. Denk hierbij bijvoorbeeld aan variatie als gevolg van offshore ontwikkeling.

Als u de schoenendoos aan stories even terzijde schuift dan kunt u vaak niet zeggen of code, testcases en requirements in een agile of een traditioneel project ontwikkeld werden.

Wat ons betreft bestaan 'agile requirements' dus niet. Agile projecten wel. En agile projecten vereisen van analisten een andere manier van denken en werken dan in meer traditionele projecten.

Agile analisten stappen af van de traditionele houding om alles vooraf uit te zoeken en vast te leggen. In plaats daarvan zorgen zij ervoor dat alle projectleden precies op het juiste moment ('Just in Time') precies de juiste ('Just Enough') informatie ter beschikking hebben. Op deze manier worden er minder tijd en resources verspild en kan iedereen binnen het project efficiënter en effectiever werken.

Agile projecten vereisen van analisten een andere manier van denken en werken dan in meer traditionele projecten.

# 2. Het nut en de noodzaak van ‘Just in Time’ en ‘Just Enough’

Agile projecten stellen het belang van uitgewerkte requirements steeds meer ter discussie. De focus ligt immers bij werkende oplossingen, meestal software. Het resultaat van het project is dus belangrijker dan de documentatie tijdens het project.

Wij zijn er bij DiVetro van overtuigd dat heldere requirements nog steeds onmisbaar zijn bij (IT) projecten. Maar er moet een goede balans zijn. Analisten willen vaak te veel zaken uitzoeken, analyseren en opschrijven. De concepten ‘Just in Time’ en ‘Just Enough’ bieden een mooie oplossing voor dit probleem.

## ‘Just In Time’: het juiste moment

‘**Just In Time**’ (JIT) is bekend door Lean Six Sigma. Het is een principe dat erop gericht is om precies op het juiste moment de juiste dingen te doen.

Het principe wordt veel toegepast in de logistiek en de industrie, maar het kan net zozeer op IT-projecten toegepast worden. Agile is in wezen niets anders dan Lean toegepast op IT-projecten.

We zien nog te vaak projecten waarbij vooruit gewerkt wordt om (valse) zekerheid te verschaffen over de volgende sprint of simpelweg om mensen aan het werk te houden. In realiteit levert 'vooruit werken' echter niet altijd de gewenste waarde op. Het maakt het project als totaal niet effectiever.

Wij zijn er bij DiVetro van overtuigd dat heldere requirements nog steeds onmisbaar zijn bij (IT) projecten. Maar er moet een goede balans zijn.

Als analisten te veel vooruit werken dan ontstaat er immers voorraadvorming. Dat is volgens Lean een vorm van **verspilling** (**muda**).

Dit heeft vervelende gevolgen in productielijnen (opslag van ongebruikte producten is duur), maar voorraadvorming kan ook overbodig en nadelig zijn voor IT-projecten. Het kost immers onnodige tijd en resources, en daar wordt niemand gelukkig van.

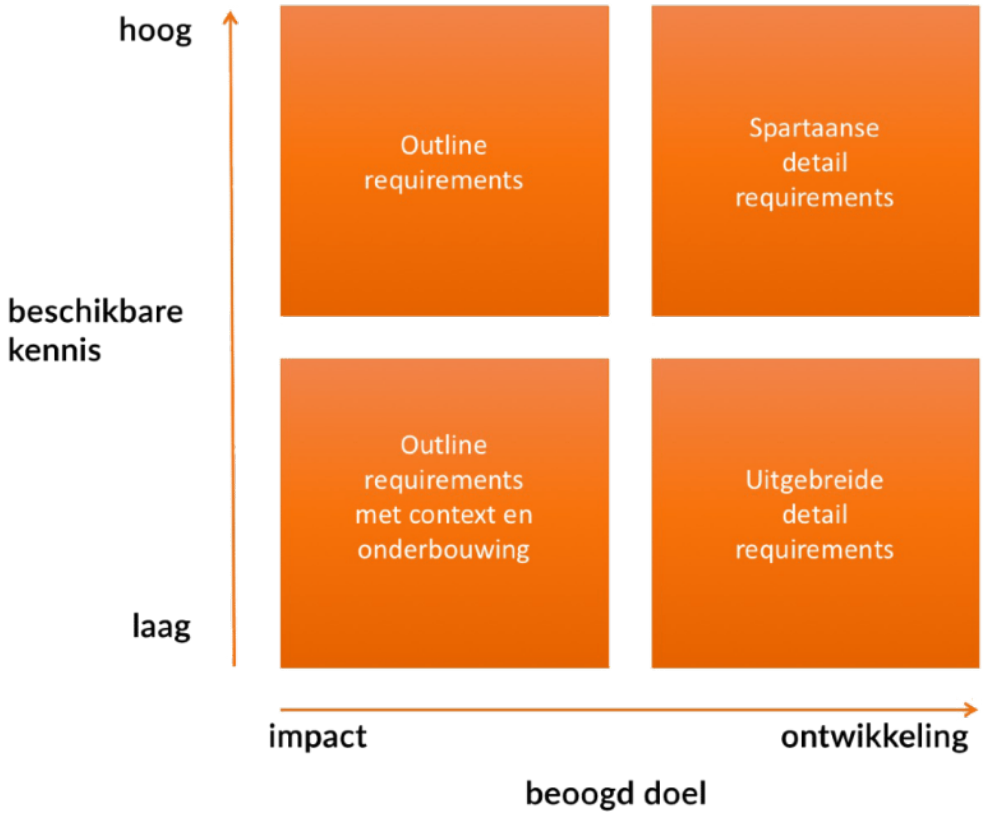
## Just Enough: de juiste diepgang

**'Just Enough'** is ook een veel gebruikt principe in Lean projecten. Wij vinden dit principe bijzonder waardevol bij het opstellen van requirements voor (IT-)projecten.

Vanuit een procesgedachte zouden requirements **precies voldoende informatie** moeten bevatten om de vervolgstappen succesvol te laten verlopen. De diepgang van de requirements is hierbij afhankelijk van de beschikbare kennis en het doel.

Als er al veel kennis binnen het ontwikkel- en beheerteam aanwezig is en als deze informatie in het hele team gedeeld is dan lijkt veel opschrijven niet nodig. Het is echter wel cruciaal dat iedereen op de hoogte is van informatie die voor hen belangrijk is. Ook externe stakeholders!

Bij een hoog gedeeld kennisniveau is ons advies om 'Spartaanse' ofwel beknopte documentatie op te leveren waarin alleen de meest belangrijke details staan (zie de figuur hiernaast). U hoeft geen details op te nemen die bekend zouden moeten zijn voor mensen die al in het domein bekend zijn.



Verder hoeft niet iedere informatiebehoefte van een stakeholder gevolgd te worden. Daarvoor is een breder draagvlak nodig. Een eventueel 'gat' in kennis van een bepaalde stakeholder kan eventueel met maatwerk worden opgelost, bijvoorbeeld door een opleiding of betere interactie.

## Requirements voor impactbepaling en prioriteitsbepaling

Zoals gezegd is de diepgang van requirements niet alleen afhankelijk van de beschikbare kennis maar ook van het doel waarvoor ze gebruikt zullen worden. Zo dienen requirements voor impact- en prioriteitsbepaling overzicht en scope te geven. Ze moeten precies voldoende details hebben, zodat een outline van de gewenste oplossing ontstaat.

Op basis van deze outline kunnen we vervolgens, bijvoorbeeld door het opstellen van user stories, inschatten hoeveel werk er te verzetten valt en welke onderdelen de meeste waarde voor de klant toevoegen. Voor het stellen van prioriteiten door een product owner is deze beperkte diepgang genoeg.

## Requirements tijdens het ontwikkelproces

Voor het ontwikkelproces is in veel gevallen **meer diepgang** in de requirements nodig. Er volgt dan een verdieping op de outline zodat er precies voldoende informatie is om te weten wat de klant wil en zodat de ontwikkelaar kan bouwen.

Meer informatie is niet nodig. Meer informatie kan zelfs ballast zijn; het kan het proces vertragen. Niemand zit nog te wachten op uitgebreide (wollige) verhalen. En toch stappen we als analisten vaak nog in dezelfde valkuil: we schrijven op wat we weten, ook als dat nu niet de prioriteit heeft.

Voor het ontwikkelproces is in veel gevallen **meer diepgang** in de requirements nodig.

Andere projectleden zullen hier om- of doorheen moeten lezen. Dit kost extra tijd en het leidt mogelijk tot fouten omdat de essentie verdoezeld wordt. Daarom is het belangrijk om zaken die nu niet relevant zijn altijd op een **backlog** te plaatsen en extra toelichtingen die u niet wilt verliezen in een apart document of een bijlage op te nemen.

## Zijn 'Just In Time' en 'Just Enough' overal even noodzakelijk?

De praktijk is natuurlijk nooit alleen zwart en wit. Iedere organisatie en ieder project zijn weer anders. Maak daarom bij het begin van elk project heldere afspraken over het niveau en de diepgang van de requirements.

Wij adviseren om als team voldoende tijd te besteden aan het doordenken van de documentatiestandaarden en het afstemmen van de behoeften en de verwachtingen van alle stakeholders.

Het kan helpen om bestaande standaarden van de organisatie als startpunt te gebruiken en op basis hiervan een voorbeeld uit te werken of richtlijnen op te stellen.

Uiteraard mag het volgen van een standaard of richtlijn geen doel op zich worden. Het moet in de specifieke projectsituatie passen. Ga dus nooit van start zonder goede afstemming met de stakeholders. De behoeften van de stakeholders moeten altijd helder zijn.

## Conclusie

Requirements bepalen in grote mate het succes van (IT) projecten. Dat wil zeggen dat analisten een cruciale rol en een grote verantwoordelijkheid hebben.

Om het project efficiënt te laten verlopen en zo weinig mogelijk tijd en resources te verspillen raden wij analisten aan om '**Just in Time**' en vooral '**Just Enough**' requirements op te leveren.

Daarbij is het ook belangrijk om tijd te investeren in de afstemming van de informatiebehoefte van alle stakeholders en het team.

Lever als analist de requirements just in time en vooral just enough op voor uw stakeholders zodat u samen succesvol kunt zijn.

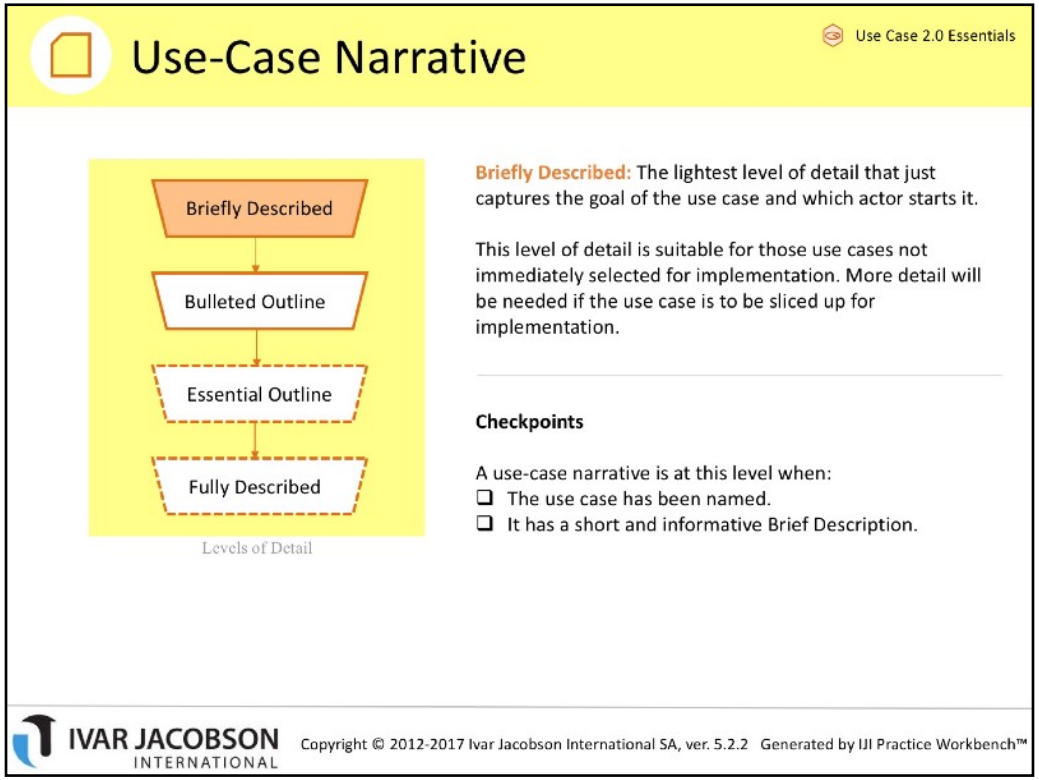
# 3. ‘Just in Time’ en ‘Just Enough’ in de praktijk

In het vorige hoofdstuk beschreven we het belang van ‘Just in Time’ en ‘Just Enough’ voor de efficiëntie van projecten. In dit hoofdstuk bekijken we hoe analisten ‘Just in Time’ en ‘Just Enough’ informatie in requirements kunnen verwerken.

Want hoe stellen we bijvoorbeeld vast wat op welk moment de juiste diepgang voor requirements is? Hoe voorkomen we dat analisten verzanden in het uitschrijven van requirements tot een niveau waarop niemand hen nog kan volgen? En hoe voorkomen we dat de toegevoegde waarde van de extra inspanningen nihil wordt en enkel bijdraagt aan het verhogen van verspilling (muda)? Kortom…

Wanneer hebben we in de praktijk ‘Just in Time’
’Just Enough’ details in requirements?

‘Just in Time’ en ‘Just Enough’ zijn principes die elkaar sterk raken. Wat op het ene moment voldoende diepgang is kan zomaar onvoldoende zijn op een ander moment. Het toepassen van de 'Just in Time' en 'Just Enough' principes is dus een voortdurende zoektocht naar de juiste balans. De relatie tussen de principes en de invulling hiervan laat zich het beste uit leggen aan de hand van een 'standaard project' waarbij de normale projectcyclus van idee tot realisatie wordt doorlopen.




## Efficiënte analyse toegepast: de start van uw project

Een project begint meestal vanuit een behoefte of een goed idee. Maar al snel ligt de budgetvraag op tafel. Om een eerste schatting van het budget te kunnen maken hebben we een scope of een stip aan de horizon nodig. Dat kan op een aantal manieren.

Nog niet zo lang geleden stelden we eerst een use case model op. We definieerden zoveel mogelijk use cases en we schreven deze wellicht ook al volledig uit. Deze aanpak is echter niet altijd efficiënt. Aanhangers van agile methodieken stellen dan ook vaak dat use cases en use case modellen niet meer bij software ontwikkeling passen.

Wij vinden dit niet terecht. Onze ervaring leert dat use cases en use case modellen nog steeds een toegevoegde waarde hebben, ook voor software ontwikkeling. Het is echter wel zeer belangrijk om kritisch te blijven op de details die toegevoegd worden. Een use case model zonder in detail uitgeschreven use cases kan bijvoorbeeld al voldoende zijn voor een scoping vraag.

## Een use case model zorgt voor overzicht en richting

Een use case model is uitermate geschikt om snel en simpel een 'stip aan de horizon' te identificeren. Een simpele 'model storm' met de belangrijkste stakeholders kan snel tot een eerste use case model leiden. Bij een dergelijke 'model storm' worden de primaire use cases onderkend en krijgen ze (alleen) een korte omschrijving. Hiermee wordt de scope van het systeem vastgesteld.

Door hier ‘Just Enough’ details te beschrijven blijft het use case model van absolute waarde in elke agile aanpak. Het use case model biedt namelijk snel een overzicht van de 'big picture' van het project.

## Een use case model zorgt voor samenhang tussen user stories

Wanneer het budget geregeld is kunt u een (agile) team samenstellen om het systeem te realiseren. Dit team heeft een 'backlog' nodig: een lijst waarin beschreven staat waar we met welke prioriteit aan werken. Het (hoog niveau) use case model dat we eerder aanmaakten om de scope van het project aan te duiden kan hierbij als uitgangspunt dienen om user stories te identificeren en te beschrijven.

Met een duidelijk use case model blijft de samenhang tussen de stories duidelijk en werken zowel het team als de product owner altijd vanuit de 'big picture'.

## Een use case model als kapstok voor de backlog

Om goede user stories te kunnen onderkennen hebben we meer details nodig dan de eerder gemaakte beknopte omschrijving van de use cases. Deze verdieping kan per use case beperkt worden tot een puntsgewijs stappenplan en het onderkennen van logische uitzonderingen. Ook hier wordt dus gekozen voor ‘Just Enough’ details voor het moment.

Elke user story kan op een behapbaar deel van het proces gebaseerd worden. Zo implementeert iedere user story automatisch een stukje van een use case en dus ook de scope van het project. Het use case model en de use cases dienen dus als 'kapstok' voor de backlog.

Op dit punt hebben we een budget en een team en weten we wat we willen realiseren. De vraag is daarom terecht of het echt nodig is om op dit moment meer details per user story toe te voegen.

## Details in de bouwfase

Vanaf het moment dat we daadwerkelijk starten met bouwen is het ook nog steeds van groot belang om per story te kijken of we meer details zouden moeten toevoegen. Deze keuze hangt vaak af van diverse factoren (zie oranje kader). Hoe complexer de antwoorden op deze vragen zijn, hoe verstandiger het is om meer details toe te voegen.

We kunnen tijdens de hele bouwfase per user story vaststellen hoeveel detail er nodig is om de story te implementeren of over te dragen aan beheer. Voor de vastlegging van deze details kunnen we kiezen uit diverse werkproducten: de use case narratives (tekst), test cases, supporting information en zelfs use case realizations. In het volgende hoofdstuk gaan we nader in op de geschikte wijze van vastleggen.

De 'lijm' tussen al die werkproducten is de use case zelf. Elk werkproduct kent zijn eigen doel en per story kiezen we hoeveel detail er nodig is. Op deze manier worden use cases steeds verder verdiept maar enkel wanneer hiervoor een noodzaak of behoefte bestaat. Het eindeloos detailleren van een use case is geen doel op zichzelf!

- Hoe hoog is het gedeelde kennisniveau binnen het team en tussen het team en de stakeholders?
- Hoe 'volwassen' is het team?
- Hoeveel teams zijn er betrokken bij het implementeren van het systeem en wat is de geografische spreiding van de teams en de teamleden?
- Hoe makkelijk en frequent heeft het team toegang tot subject matter experts (materiedeskundigen)?
- Hoe strikt dient aan wet- en regelgeving voldaan worden?
- Is er sprake van een nieuw product met een korte verwachte levensduur of van een langlopend product waarop wellicht veel wijzigingen zullen plaatsvinden?
- Werken we met bewezen architectuur?

# 4. De meest geschikte requirements voor ieder moment

In de vorige hoofdstukken stonden we stil bij de relatie tussen ‘Just Enough’ en ‘Just in Time’. In dit hoofdstuk staan we stil bij de wijze waarop we ‘Just Enough’ informatie in requirements kunnen opnemen.

Hiervoor moeten analisten over een palet aan mogelijkheden beschikken want ‘one size does not fit all’. Requirements zijn geen eenpansmaaltijd.

## Requirements zijn geen eenpansmaaltijd

Wie kent ze niet? De klassieke ‘functionele ontwerpen’ of ‘programma’s van eisen’? Lijvige documenten waarin de requirements van vaak complexe IT-systemen allesomvattend zijn beschreven? Documenten waarin alles door elkaar staat, van high level eisen tot de beschrijving van de werking van specifieke systeemonderdelen? Hoe kunnen analisten in dit soort ontwerpen rekening houden met ‘Just in Time’ en ‘Just Enough’ en het geheel toch leesbaar houden? Het antwoord op deze vraag komt uit onverwachte hoek.

Een culinair diner bestaat vaak uit meerdere gangen van gerechten. Daarbij komen verschillende smaken bij elkaar op een mooi opgemaakt bord. Wanneer de chef alle ingrediënten van alle gerechten in één pan zou bereiden dan zouden de meeste klanten het eten waarschijnlijk laten staan!

Zo is het ook met requirements. Een goede set bestaat uit verschillende soorten requirements. Wanneer alle requirements in één document bij elkaar staan dan ontstaat er een brij die geen van de stakeholders optimaal zal smaken. De oplossing voor dit probleem is gelukkig verrassend eenvoudig:

**Benader uw requirements als een chef!**

Gasten in een restaurant willen ook niet altijd hetzelfde eten. Daarom maken chefs verschillende gerechten voor verschillende gasten. Logisch, toch?

Waarom zouden we onze requirements dan nog als eenpansmaaltijd serveren?

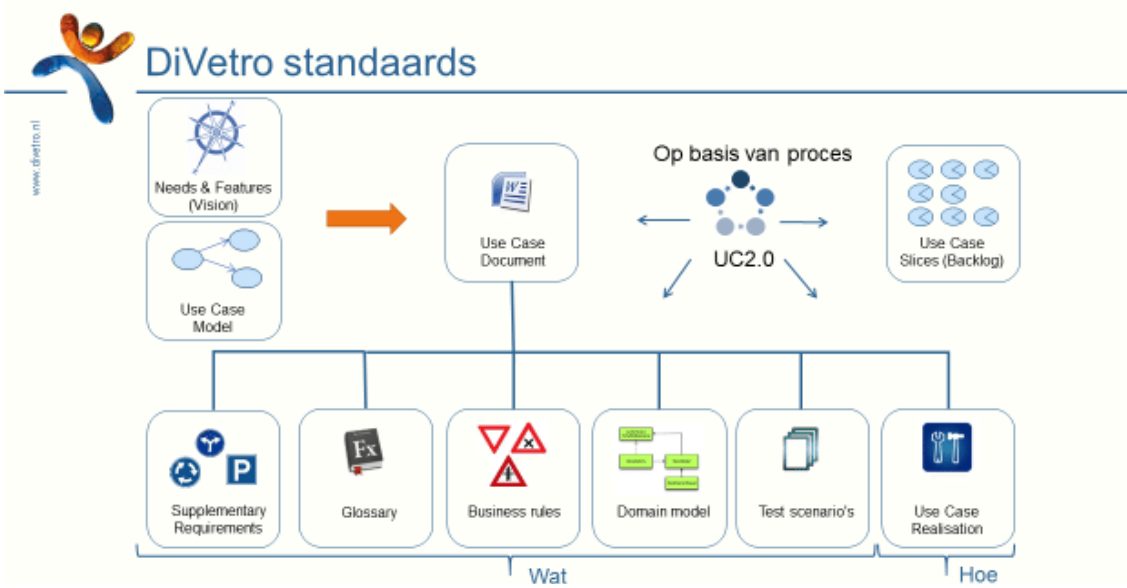
## Scheiding van smaken

Verschillende stakeholders of teamleden hebben in verschillende fases van een project vaak behoefte aan andere informatie. Zo zal een business vertegenwoordiger aan het begin van het project een overzicht willen zien van de high level requirements waar een oplossing voor wordt gevraagd.

Later wil hij lezen hoe dit zich vertaalt naar flows die deze needs en features invullen. Een ontwikkelaar wil kunnen lezen welke flows, bedrijfsregels en andere eisen hij in een sprint moet realiseren. En een tester wil weten welke testscenario's moeten slagen om een stuk functionaliteit goed te keuren.

**Zorg er daarom voor dat u uw requirements handig en gestructureerd opsplitst en bundelt.**

DiVetro maakt hierbij gebruik van een op UC2.0 gebaseerd standaard raamwerk met templates voor de verschillende soorten requirements. Het gebruik van deze templates zorgt ervoor dat alle vastgelegde informatie geordend en makkelijk terug te vinden is.



## Verschillende gangen met verschillende informatie

Wij serveren onze requirements in diverse gangen en bieden per gang diverse informatie aan. Alle stakeholders mogen vervolgens hun smaak bepalen en de juiste documenten kiezen.

### Eerste gang

We serveren altijd eerst een high level beeld van wat gewenst is:

- **Needs en Features** beschrijven de high level wensen van de stakeholders
- Het **Use Case Model** vertaalt dit naar high level doelen van de actors

Zorg ervoor dat u uw requirements handig en gestructureerd opsplitst en bundelt. Use Case 2.0 biedt daarvoor een handig raamwerk.

### Tweede gang

We volgen het agile gedachtegoed en beschrijven wat het team nodig heeft om het werk te kunnen doen. Daarvoor bieden we diverse smaken:

- **Use Case beschrijving**: deze beschrijft een systeem vanuit het gebruikersperspectief. Het beschrijft de actor, de initiator van de interactie en het systeem zelf als een opeenvolging van eenvoudige stappen.
- **Supplementary Requirements**: aanvullende requirements die vaak voor de gehele oplossing van toepassing zijn en niet bij één specifieke Use Case behoren. Vaak zijn dit non-functionals.
- **Glossary**: Een lijst met termen die in de diverse onderdelen worden gebruikt zodat een consistent vocabulaire kan worden toegepast.
- **Business rules**: bedrijfsregels / validaties.
- **Domain model**: een conceptueel model van het (business) domein van de klant. Vaak in de vorm van een UML klassendiagram.
- **Testscenario's**: een set testcondities en verwachte resultaten waarmee we kunnen bepalen of het systeem al dan niet correct werkt.

Met behulp van deze smaken kunnen we nu middels het ‘Just in Time’ en ‘Just Enough’ principe details toevoegen waar en wanneer het nodig is. Het wanneer zal vooral afhangen van het moment waarop een user story daadwerkelijk in een sprint geïmplementeerd zal worden.

De specifieke inhoud van een requirement bepaalt vaak de vorm van vastleggen:

- Wet- en regelgeving leent zicht bijvoorbeeld goed voor business rules.
- Procesbeschrijvingen en interacties tussen gebruikers en systemen zijn goed vast te leggen binnen een Use Case beschrijving.

## Derde gang

High level (technische) oplossingen kunnen worden vastgelegd in **Use Case Realizations**. Use Case Realizations beschrijven op een hoog niveau hoe delen van het systeem samenwerken om de gewenste oplossing te realiseren. Use Case Realizations kunnen diverse verschijningsvormen hebben. Veelgebruikte vormen zijn een sequence diagram of een interactie / scherm ontwerp.

In tegenstelling tot in een restaurant zijn onze ‘gangen’ niet sequentieel; ze komen iteratief tot stand. Bovendien beschrijven we alleen wat op dat moment nodig is voor het team en op het detailniveau dat het team nodig heeft. In de vorige hoofdstukken hebben we deze aanpak in detail beschreven.

DiVetro heeft bovenstaande aanpak succesvol gebruikt voor onder andere de specificaties van de ‘Burgerzaken modules’; specificaties die de burgerzakenprocessen beschrijven zoals zij in meer dan 380 gemeenten in Nederland gevolgd worden.

Klik hier om naar de website van de 'Burgerzaken module specificaties' te gaan. Deze specificaties beschrijven de burgerzakenprocessen zoals zij in meer dan 380 gemeenten in Nederland gevolgd worden.

## Conclusie: Requirements zijn geen eenpansmaaltijd

Iedere stakeholder heeft op elk moment in het project andere behoeften met betrekking tot de requirements in het project.

**De diepgang en de vorm van de requirements is afhankelijk van de fase van het project en de stakeholder die ermee aan de slag gaat.** Het is dan ook haast onmogelijk om één document te maken waarin de requirements voor alle stakeholders en teamleden optimaal beschreven zijn.

Daarom leggen we de requirements vast op de meest logische plaats en voegen we op een agile wijze fasegewijs details toe die nodig zijn om het werk te kunnen verrichten.

Door een vaste structuur als basis aan te houden vergroten we de vindbaarheid van de requirements op zowel hoog niveau als detailniveau. Door te kiezen voor een logische plaats en vorm in plaats van één dik document wordt de lezer niet langer afgeleid door zaken die op dat moment niet relevant voor hem zijn.

# 5. Verleden, heden en toekomst binnen requirements

Met de juiste set van requirements standaarden zijn we verzekerd dat we in iedere situatie de details in onze requirements op geschikte wijze kunnen vastleggen. Met een goede set van requirements standaarden kunnen we goed invulling geven aan het 'Just Enough' principe.

Het 'Just in Time' principe ligt in de praktijk vaak wat complexer. Analisten hebben immers de neiging om in hun ontwerpen alvast functionaliteit voor latere sprints uit te werken. Dit wordt vaak in de hand gewerkt door organisaties die wel zeggen dat ze agile werken maar die in de praktijk eerder een mini-waterval van analyse, bouw en testen gebruiken.

Het is belangrijk om ervoor te zorgen dat het verleden, het heden en de toekomst in de requirements niet door elkaar gaan lopen. In dat opzicht kunnen analisten nog wel wat leren van ontwikkelaars.

Zorg ervoor dat verleden, heden en toekomst in de requirements niet door elkaar gaan lopen.

## Wat analisten van ontwikkelaars kunnen leren

Het toepassen van versiebeheer met tooling zoals bijvoorbeeld SubVersion, GIT en TFS is voor een programmeur de normaalste zaak van de wereld. Door het gebruik van versiebeheer kunnen meerdere ontwikkelaars binnen hetzelfde project tegelijkertijd meerdere stories implementeren. Per story blijft duidelijk wie welke code op welk moment ontwikkelde.

Bij analisten is versiebeheer vaak minder gestructureerd. Hoe kunnen alle teamleden bijvoorbeeld de exacte wijzigingen in een ontwerp zien die een analist in sprint 4 voor een bepaalde user story heeft toegevoegd? Hier kunnen analisten van ontwikkelaars leren.

In een agile project gaat werkende software voor uitgebreide documentatie. Er is dan ook geen behoefte om in één keer complete use cases / ontwerpen uit te schrijven en 'af' te werken. De analist moet starten met high level basisversies en deze 'Just in Time' en 'Just Enough' verder uitwerken. Maar daar wringt vaak de schoen.

## Vooruitwerken in hetzelfde document

Analisten hebben de neiging om in hun ontwerpen alvast functionaliteit voor latere sprints uit te werken. Daarmee ligt het toekomstige werk van tevoren vast en blijft de analist het team voor. Dit vooruitwerken is niet altijd even handig.

Als een analist vooruitwerkt en zo uitgebreide ontwerpen oplevert, dan moeten de overige teamleden eerst uitzoeken welke functionaliteit ze voor deze sprint moeten ontwikkelen. Hoe meer analisten vooruitwerken, hoe langer teamleden moeten zoeken.

Sprintwijzigingen worden in de praktijk vaak duidelijk gemaakt met kleurtjes, hulpteksten, een versiebeheertabel of andere oplossingen, maar na laten we zeggen zeven sprints beginnen de kleurtjes wel op te raken…

## Versiebeheer met track changes

**Conform de agile principes kan iedere sprint de laatste zijn.** Analisten moeten hun werk dan ook in lijn houden met wat het team daadwerkelijk oplevert. Het is echter ook hun verantwoordelijkheid om ervoor te zorgen dat alle teamleden snel en eenvoudig de wijzigingen van een sprint kunnen achterhalen.

Dit is niet eenvoudig te bereiken met kleurtjes en ook niet met track changes. Track changes geeft dan wel aan wie op welk moment welke wijzigingen in een document heeft doorgevoerd, maar de functie houdt ook minder of geheel niet relevante wijzigingen bij zoals formattering en wijzigingen in nummeringen. Hierdoor leveren analisten al snel een ontwerp op dat eruitziet als een bonte kerstboom met opmerkingen en kleurtjes.

Het gevolg laat zich raden: de ontwikkelaars zien een gewenste wijziging over het hoofd en maken iets wat niet in lijn is met het ontwerp. In het beste geval komen de testers er op tijd achter en kan het team de afwijkingen nog voor de demo repareren.

Op zulke momenten denken ontwikkelaars “Hoe had ik dit kunnen weten?” en analisten denken “Lees de requirements dan toch. Daar staat alles in.” De overige teamleden denken “Dit moet anders, maar hoe?”

## Voorkom lezersmoeheid en bespaar tijd als team

De oplossing is verrassend eenvoudig: **analist, help uw lezer.** Wij raden analisten aan om hun documenten in lijn met de sprints en zonder 'track changes' bij te werken.

Voor elk Product Backlog Item (PBI) maken analisten een eigen specifieke versie van de relevante ontwerpen en daarin markeren ze zelf de relevante wijzigingen voor deze PBI met specifieke kleuren. Bijvoorbeeld blauw is nieuw en rood is vervallen (impliciet valt 'gewijzigd' hier uiteraard ook onder).

Het is een extra inspanning, maar het helpt al meteen bij het doorspreken van de wijzigingen met de overige teamleden. Daarna wordt deze versie van de specificaties aan de PBI gehangen en is deze beschikbaar voor iedereen die aan dit item werkt.

Op die manier is iedereen tevreden. Analisten maken een overzichtelijk document en de teamleden hoeven achteraf niet meer te zoeken naar het ontwerp én ze zien snel wat er gewijzigd is. Daarnaast biedt deze specifieke versie een handzame basis bij het analyseren van defects op de betreffende PBI.

Wanneer dit eenmaal geborgd is, kunnen de kleurtjes weer uit het ontwerp gehaald worden. De versie zonder markeringen kan vervolgens binnen de centrale repository opgeslagen worden. Net zoals een programmeur dus die zijn werkende code 'incheckt' voor de uiteindelijke oplevering.

**Het resultaat? Het ontwerp van de analist en de code van de programmeur zijn in lijn en iedereen is klaar voor een volgend PBI. Zoals het hoort!**

## Conclusie

In een agile omgeving is het belangrijk om de focus (enkel) te leggen op de relevante zaken van dat moment en die te vertalen naar het ontwerp. Toekomstige aanpassingen komen later wel.

Analisten kunnen zich beter concentreren op wijzigingen die nu belangrijk zijn en in lijn blijven met de rest van het team. Met een klein beetje extra inspanning en wat creativiteit kunnen we lezersmoeheid voorkomen en voor to-the-point documentatie zorgen die snel toegankelijk is.

# 6. Werken vanuit een totaalbeeld met behulp van UC2.0

In dit e-book hebben we uitgebreid stilgestaan bij het nut, de noodzaak en de implementatie van ‘Just in Time’  en ‘Just Enough’ door analisten. Maar om hier echt goed invulling te geven aan ‘Just in Time’ en ‘Just Enough’ is het zeer belangrijk om altijd vanuit een totaalbeeld te blijven werken en niet mee te gaan in het moment van de dag (sprint). De analisten van DiVetro maken hiervoor gebruik van UC2.0.

Bij Agile/Scrum projecten werken we op basis van user stories. Elke user story beschrijft hierbij enerzijds een gebruikersdoel en anderzijds een werkpakket voor het ontwikkelteam. Bij elke sprint realiseert het ontwikkelteam één of meerdere user stories.

Bij langere projecten ontstaan op deze manier al snel grote hoeveelheden user stories. Hoe meer stories er al zijn gemaakt, hoe uitdagender het wordt om het totaalbeeld vanuit de stories te schetsen. Daardoor is het vaak moeilijk om het overzicht te bewaren en de werking van het ontwikkelde systeem aan nieuwe projectleden of de beheerorganisatie uit te leggen.

User stories aan elkaar plakken is meestal arbeidsintensief en tijdrovend. Maar met de juiste aanpak is achteraf plakken niet nodig.

## Knippen vanuit het totaalbeeld

Bij DiVetro werken we bij grotere projecten al bij aanvang op basis van het totaalbeeld van het systeem. Dit totaalbeeld, de 'big picture', is van groot belang om tijdens het ontwerpen en ontwikkelen de juiste beslissingen te nemen. En dit totaalbeeld knippen we op tot werkpakketten. **Knippen is immers makkelijker dan plakken.**

Vanuit het totaalbeeld kunnen we op elk moment de werking van het systeem en het belang van de individuele user stories uitleggen. Daarbij maken we bij DiVetro gebruik van onze eigen standaarden, handige hulpmiddelen die gebaseerd zijn op de Use Case 2.0 practice van Ivar Jacobson International.

## Werken aan de hand van Use Case 2.0

Use Case 2.0 is een practice die onderdeel uitmaakt van de zogenaamde practice library van Ivar Jacobson International. Binnen deze practice library bevinden zich diverse best practices die invulling geven aan specifieke onderdelen van het ontwikkelproces.

De UC2.0 practice biedt bijvoorbeeld hulpmiddelen om requirements gemakkelijk en eenduidig tot een specifiek detailniveau uit te werken. Zo kent een Use Case beschrijving (narrative) bijvoorbeeld een 4-tal specifieke statussen (states) die ieder een specifiek detailniveau beschrijven.

Een 'Briefly Described' Use Case heeft bijvoorbeeld niet veel meer dan een beknopte omschrijving. Door vanuit deze statussen te werken is het voor analisten en andere lezers meteen duidelijk tot welk detailniveau een requirement is uitgewerkt. We kunnen ook per specifieke user story bepalen of de huidige diepgang nog steeds voldoende is.

Use Case 2.0 geeft ook voor diverse andere requirements producten specifieke handvatten waardoor het eenvoudiger wordt om eenduidig, op tijd en herhaalbaar werk van voldoende kwaliteit op te leveren.

'Just in Time' en 'Just Enough' dus. Dat werkt wel zo prettig.

# DANKWOORD

Dit e-book is ontstaan door de kennis en ervaring van onze professionals en onze klanten te bundelen.

Wij bedanken hierbij graag al onze klanten en collega's bij DiVetro die bewust en onbewust hebben bijgedragen aan de totstandkoming van onze inzichten en dit e-book. Zonder hen was deze publicatie niet mogelijk geweest.

DiVetro
analyse • sourcing • management

# OVER DE AUTEURS

## Dennis Geluk

Dennis is een senior informatie analist en partner van DiVetro. Hij heeft meer dan 15 jaar IT ervaring. Dennis is een gecertificeerd Use-Case 2.0 coach. Hij heeft bij verschillende organisaties een spilfunctie vervuld bij het implementeren van UC2.0, het invoeren van Agile werken en het verbeteren van requirements standaarden. Momenteel is Dennis coach en business analist bij de KLM en helpt hij diverse organisaties met vragen rond professionalisering.

## Marco Balvers

Marco is senior informatieanalist bij DiVetro. Hij heeft meer dan 15 jaar IT-ervaring. Marco helpt organisaties met Agile werken en het inpassen van Use Case 2.0 in hun bestaande werkwijze. Momenteel werkt Marco als analist binnen een Scrum team van de NS. Dit team is verantwoordelijk voor de ontwikkeling van Apps voor eerstelijns medewerkers.

DiVetro
analyse • sourcing • management

## Richard Hilekes

Richard is een informatie analist van DiVetro. Hij heeft meer dan 10 jaar ervaring binnen dit vakgebied bij verschillende organisaties en met diverse tooling (bijv. Enterprise Architect). De laatste jaren vooral in een Agile/Scrum omgeving. Richard is een gecertificeerd Use-Case 2.0 master. Momenteel is Richard informatie analist en informatie architect bij De Wever.

## Rob den Hollander

Rob is senior informatieanalist bij DiVetro. Hij heeft ruim 15 jaar ervaring met informatieanalyse in systeemontwikkeling en is gespecialiseerd in UC2.0 en agile werken. Momenteel helpt Rob verschillende opdrachtgevers met het ontwikkelen/ professionaliseren van agile teams.

DiVetro
analyse • sourcing • management

## Ronald Koenis

Ronald is een senior analist die meer dan 25 jaar in het IT vak zit. Zijn brede ervaring heeft hij in diverse rollen opgedaan. Hij was onder meer informatieanalist bij Albert Heijn, projectleider bij de gemeente Almere en hij is momenteel product owner en information engineer bij het Kadaster. Ronald is in staat complexe IT-oplossingen zo weer te geven dat die voor de business tegelijkertijd begrijpelijk en voldoende specifiek zijn, zodat IT ermee aan de slag kan.

## Selmar van Egmond

Selmar is een senior analist. Hij zit inmiddels zo'n 20 jaar in het vak en hij werkt sinds 2014 bij DiVetro. Voor Selmar begint een goed project met een gedegen analyse en het definiëren van een stip op de horizon. Een goede analyse gaat terug naar de werkelijke oorzaken van het probleem zodat een effectieve oplossing kan worden gezocht. De stip op de horizon zorgt voor een doelgerichte aanpak waarop alle betrokkenen zich kunnen richten.

DiVetro
analyse • sourcing • management

## Stanley Lachman

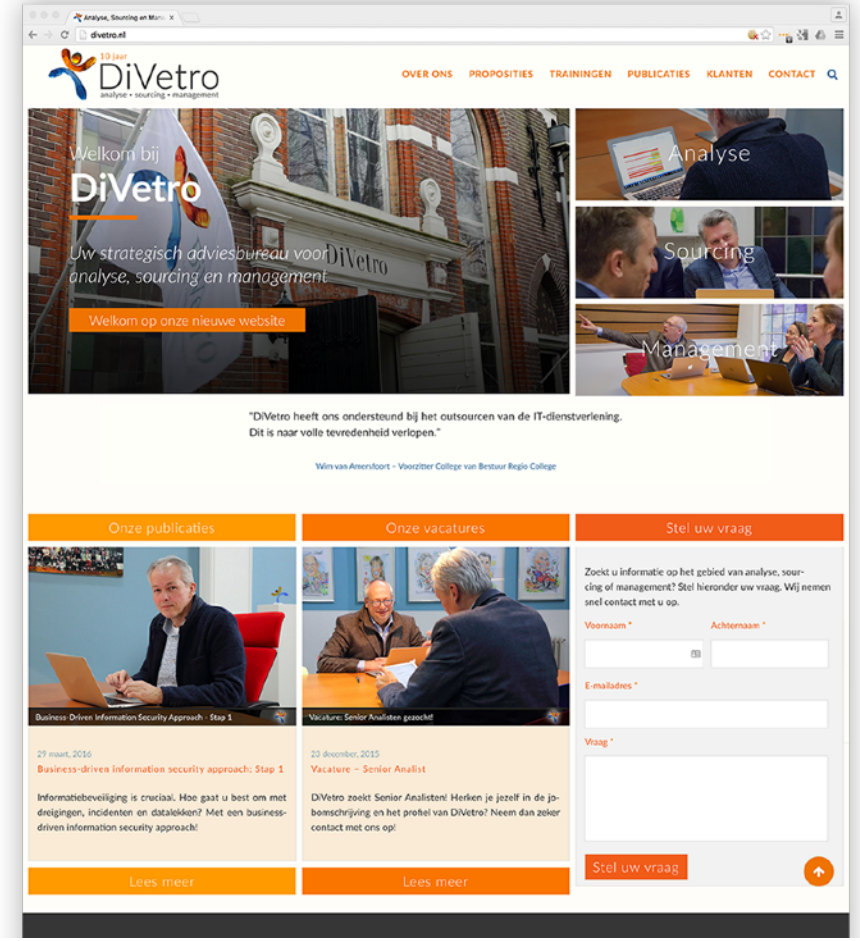
Stanley is een business- en informatieanalist bij DiVetro. Hij heeft ruim tien jaar ervaring binnen dit vakgebied. Stanley heeft de laatste jaren analyseopdrachten uitgevoerd in een agile SCRUM setting in dynamische en complexe omgevingen. Als spil tussen techniek en de business streeft Stanley ernaar diverse disciplines dichter bij elkaar te brengen om zodoende een bruikbare oplossing neer te zetten. Stanley is momenteel als business analist werkzaam bij de KLM, waar hij werkt aan het digitaliseren van informatie voor het vliegend personeel.

DiVetro
analyse • sourcing • management

# OVER DIVETRO

**DiVetro** is een onafhankelijk strategisch adviesbureau. Ons kantoor bevindt zich in Arnhem en wij werken voor opdrachtgevers in heel Nederland.

Wij beantwoorden en implementeren IT-gerelateerde vraagstukken over **Analyse**, **Sourcing** en **Management**. Wij werken vooral op projecten bij de (semi-)overheid, in de gezondheidszorg en in het onderwijs. Onze consultants hebben ruime ervaring op hun vakgebied.




# MEER INFORMATIE

Heeft u vragen naar aanleiding van dit e-book? Aarzel niet en neem vrijblijvend contact met ons op.

**U kunt ons bereiken via +31 (0)26 443 67 90 of info@divetro.nl.**

Op onze website www.divetro.nl vindt u aanvullende informatie over onze dienstverlening en werkwijze. We publiceren hier ook geregeld artikelen over nieuwe inzichten in ons vakgebied.