Agile requirements engineering & SCRUM

- Betere software en een efficiënter proces door requirements gedreven Scrum –

Scriptienummer: 664
Datum: mei 2013

Auteur:
C. Kleine Staarman
Master Informatica
Radboud Universiteit Nijmegen

Begeleiders:
A. van Rooij, Radboud Universiteit Nijmegen
S.J.B.A. Hoppenbrouwers, Radboud Universiteit Nijmegen
T. van Gessel, Reed Business Media

Auteur: ing. Chris Kleine Staarman

Studentnummer: 0638080

Opleiding: Master Informatica

Variant: Management en Technologie

Thema: Quality of Software Systems

Titel: Agile Requirements Engineering & Scrum

Afstudeernummer: 664

Versie: publiek

Datum: 27 mei 2013

Instituut: Onderwijsinstituut voor Informatica en Informatiekunde

Faculteit: Faculteit der Natuurwetenschappen, Wiskunde en Informatica (FNWI)

Universiteit: Radboud Universiteit Nijmegen

Begeleiders:

- dr. Arjan van Rooij, FNWI, Radboud Universiteit Nijmegen
- dr. Stijn Hoppenbrouwers, iCIS[1], FNWI, Radboud Universiteit Nijmegen
- ing. Tom van Gessel, Development Manager, Reed Business Media

[1] Institute for Computing and Information Sciences

Management samenvatting

Succesvolle productontwikkeling is voor veel bedrijven cruciaal. Er wordt dan ook gestreefd naar een optimaal proces om zo goed mogelijk aan de behoeften te kunnen voorzien.

De traditionele benaderingen van het softwareontwikkelproces is de laatste decennia duidelijk niet in staat gebleken om dit proces op een efficiënte manier vorm te geven. Software projecten vereisten grote investeringen, hadden een onvoorspelbaar karakter en slechts een lage kans van slagen. Er was daarom een nieuwe impuls nodig. Een nieuwe strategie en filosofie op het proces van softwareontwikkeling. Het antwoord hierop kwam door de Agile beweging.

Binnen het Agile gedachtegoed worden alle facetten van het ontwikkelproces op een wat andere manier benaderd. De aandacht verschuift hierbij van: planmatig, gecontroleerd, zorgvuldig en gedetailleerd, naar een nadrukkelijke focus op: efficiëntie, flexibiliteit, voorspelbaarheid en de *customer value*. Daarbij is tevens veel aandacht voor het stimuleren van innovatie, het leereffect en omstandigheden van het team. Deze Agile manier van werken is beduidend succesvoller gebleken en inmiddels de norm voor succesvolle productontwikkeling.

Het hanteren van een Agile strategie heeft een duidelijke impact op de werkwijze van het team. Zoals gebleken in dit onderzoek is dit met name het geval voor requirements gerelateerde activiteiten (een van de cruciale stappen in het ontwikkelproces). Er is een duidelijk verschil in visie gebleken tussen het traditionele requirements paradigma en het Agile gedachtegoed. Dit zorgt voor de nodige onduidelijkheid en is een belangrijke efficiëntiebeperking voor het team gebleken.

Er zijn verschillende methoden voorhanden om het softwareontwikkelproces Agile vorm te geven. Scrum is hiervan verreweg de populairste. In deze thesis is onderzocht hoe het proces van requirements engineering het beste binnen de Scrum ontwikkelmethode kan worden ingebed.

Binnen het theoretisch kader van dit document zijn alle relevante concepten toegelicht en is ook de Scrum ontwikkelmethode zorgvuldig in kaart gebracht. Om de context te verbreden is ook het Lean gedachtegoed hier geïntroduceerd. Op basis van deze theoretische onderbouwing is een visie gepresenteerd en een concept model opgesteld waarin de processen op een verantwoorde manier zijn geïntegreerd.

Doormiddel van een praktijkonderzoek zijn relevante knelpunten en behoeften binnen een softwareorganisatie in kaart gebracht. Deze praktijkinzichten zijn vervolgens verwerkt in het model. Dit heeft geresulteerd in een definitieve visie en modellering van het proces wat als aanvulling op de Scrum methode kan worden beschouwd.

De inhoud van dit document kan ontwikkelorganisaties helpen om Agile principes op een juiste manier toe te passen en in context te plaatsen. Daarnaast kan het teams helpen om requirements processen op een efficiënte manier vorm te geven en te combineren met een Agile werkwijze als Scrum.

Voorwoord

Voor u ligt het resultaat van mijn afstudeeronderzoek. Dit onderzoek is uitgevoerd ter afsluiting van mijn opleiding master informatica (variant: Management & Technologie) aan de Radboud Universiteit Nijmegen.

Mijn brede interesse in het informatica vakgebied en verdere behoefte om mij op dit gebied te ontwikkelen hebben mij gebracht tot de keuze van deze opleiding na mijn HBO studie. Al vrij snel ben ik parallel aan het volgen van deze opleiding gestart met een (bijna fulltime) baan in het bedrijfsleven. Doordat de opleiding alleen in voltijd variant wordt aangeboden, heb ik de nodige uitdaging gehad om een en ander goed te kunnen combineren. Het tempo moest daarom noodgedwongen wat worden bijgesteld, maar terugkijkend is het allemaal toch vrij goed gelukt. Ik kan terugkijken op een vruchtbare periode waarin ik veel geleerd heb en mezelf zowel theoretisch, als in de praktijk sterk heb kunnen ontwikkelen. Ik wil dan ook mijn werkgever bedanken voor de flexibiliteit die zij geboden hebben om deze combinatie mogelijk te maken.

Bij de uitwerking van deze scriptie heb ik getracht om een zo duidelijk mogelijke context te beschrijven, zodat het verslag voor een breed publiek toegankelijk is. Het op een logische manier bij elkaar brengen en beschrijven van alle concepten was een lastige puzzel, maar is uiteindelijk aardig gelukt. Graag wil ik iedereen bedanken die in welke vorm dan ook een bijdrage hieraan heeft geleverd.

In het bijzonder bedank ik: mijn begeleiders Arjan van Rooij, Stijn Hoppenbrouwers en Tom van Gessel voor de prettige begeleiding, het vertrouwen om de opdracht zoveel mogelijk zelfstandig uit te voeren en de constructieve- en stimulerende feedback gedurende dit traject. Ook wil ik mijn collega's en respondenten uit het praktijkonderzoek (Alfons Roerdink, Tim Oldenkamp, Jurjen Bersee, Esther van de Velden, Niels Winkel, Wendy Smeets, Tom van Gessel, Koenraad Bruins en Jacco Keuper) nadrukkelijk bedanken voor hun medewerking en bijdragen.

Tot slot spreek ik mijn dank uit aan Henriette Le Roy voor haar kritische blik op het eindverslag,
Geert Feron voor de inspirerende gesprekken over het Lean gedachtegoed (en zijn ervaringen hiermee binnen industriële productieprocessen) en natuurlijk mijn familie en vrienden die voor de morele ondersteuning en afleiding hebben gezorgd waardoor het voor mij allemaal te combineren was.

Dank allemaal en veel leesplezier gewenst,

Chris Kleine Staarman
Arnhem, mei 2013

DISCLAIMER: DIT BETREFT EEN PUBLIEKE VERSIE VAN HET DOCUMENT. ONDER ANDERE HET PRAKTIJKONDERZOEK EN DE AANVULLENDE BIJLAGEN ZIJN ALS VERTROUWELIJK AANGEMERKT EN DERHALVE IN DEZE VERSIE WEGGELATEN.

Inhoudsopgave

1 Inleiding

In dit hoofdstuk wordt een inleiding gegeven het onderzoek. Allereerst wordt de aanleiding van het onderzoek en de probleemstelling beschreven, vervolgens wordt toegelicht hoe het onderzoek heeft plaatsgevonden en welke deelvragen zijn geïdentificeerd om de hoofdonderzoeksvraag te kunnen beantwoorden. De inleiding wordt afgesloten met een begeleidend schrijven over de opbouw van dit verslag.

1.1 Aanleiding en probleemstelling

Agile ontwikkelmethoden zijn de laatste jaren sterk in populariteit toegenomen bij de ontwikkeling van software producten. Rede van deze beweging is de toenemende behoefte van organisaties om snel en effectief op veranderingen in te kunnen spelen en de uiteindelijke *customer value* te vergroten.

Een van de meest toegepaste Agile ontwikkelmethoden op dit moment is Scrum. Scrum is een Agile framework of methodiek voor het uitvoeren en managen van software projecten. Het beschrijft een aantal gedragsregels en processen waarmee een adaptief systeem van zelforganisatie kan worden bewerkstelligd, wat streeft naar een hoge mate van productiviteit en efficiëntie (een zogenaamde '*hyper productive state*'). Om deze toestand te bereiken moeten alle facetten van het proces op zijn plek vallen.

Een belangrijk aspect binnen het proces van softwareontwikkeling is het vergaren en vastleggen van klant- en systeemwensen (requirements). Dit proces vormt de input van het ontwikkelproces en is essentieel voor het ontwikkelen van een succesvol product. Waar dit proces binnen traditionele ontwikkelmethoden relatief veel aandacht kreeg, doordat het als aparte fase voorafgaand aan (technisch) ontwerp en implementatie werd uitgevoerd, wordt requirements engineering binnen Agile methoden als integrale activiteit gezien waardoor de focus in veel gevallen beperkter is.

Er is tevens een duidelijk verschil waarneembaar in de vorm en mate van detaillering die in de requirementsdefinitie wordt aangebracht. Binnen Agile methoden wordt duidelijk gekozen voor een pragmatische aanpak waarbij een beperkt niveau van detaillering wordt nagestreefd om een hoge mate van flexibiliteit te introduceren en niet noodzakelijke inspanning (*waste*) tot een minimum te beperken. Details worden hierdoor vaak pas in een (te)laat stadium opgehelderd waardoor deze verantwoordelijkheid steeds meer bij ontwikkelaars komt te liggen. Dit in tegenstelling tot veel requirementstheorieën waarbij juist gestreefd wordt naar een hoge mate van detaillering in de requirements fase en een duidelijke scheiding van verantwoordelijkheden tussen ontwikkelaar, analist en opdrachtgever.

Dit verschil in visie en aanpak zorgt voor de nodige onduidelijkheid en vormt een belangrijke beperking in de efficiëntie van het team. Uit de praktijk blijkt dan ook dat organisaties vaak moeite hebben om het proces van requirements engineering op een juiste manier vorm te geven en te combineren met een Agile werkwijze als Scrum.

Het kiezen van de juiste werkwijze impliceert een nadrukkelijke *trade-off* (afweging met consequenties) met betrekking tot risicobeperking, efficiëntie en kwaliteit.

1.2 Onderzoek

De doelstelling van dit onderzoek is het inventariseren van factoren die kunnen bijdragen aan een succesvolle integratie van het requirements engineering proces binnen het Scrum ontwikkelproces om uiteindelijk tot een model en aanvulling op de methode te komen waarin beide processen op een verantwoorde manier zijn geïntegreerd en geborgd.

Aan de hand van een literatuuronderzoek is relevante theorie verzameld op het gebied van Agile productontwikkeling, Scrum, Lean Development en requirements engineering.

Aangezien omgevingsaspecten en beperkende voorwaarden uit de praktijk een belangrijke rol spelen bij het succesvol implementeren van de methode is een kwalitatief praktijkonderzoek uitgevoerd om de huidige knelpunten en behoeften binnen een casus in kaart te brengen.

Het praktijkonderzoek is uitgevoerd bij Reed Business Media op de afdeling Product Development NL. Deze afdeling is verantwoordelijk voor de technische ondersteuning van het online uitgeefkanaal in Nederland en ontwikkelt daartoe online producten. De afdeling streeft Agile principes na om zo effectief mogelijk aan de behoeften van de interne uitgevers te kunnen voldoen. Zo wordt er gewerkt met meerdere Scrum teams en worden diverse *eXtreme Programming* (XP) *practices* toegepast in het ontwikkelproces. Doormiddel van diepte-interviews met verschillende betrokken disciplines in het ontwikkelproces zijn praktijkinzichten verkregen die vervolgens zijn verwerkt in het model. De uiteindelijke methode kan worden gezien als een aanvulling op Scrum.

1.2.1 Hoofdvraag

Op welke manier moet het proces van requirements engineering worden ingebed in de Scrum ontwikkelmethode?

1.2.2 Deelvragen

1. Welke aspecten van Scrum en requirements engineering zijn relevant?
   a. Welke theorie is relevant binnen deze scope?
   b. Hoe worden deze processen op dit moment in de praktijk gebracht binnen de casus?

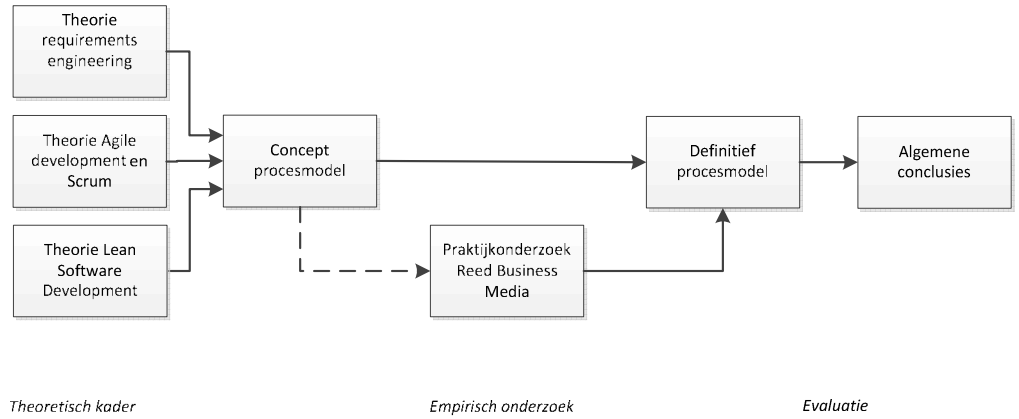
2. Op welke manier kan Lean Development bijdragen aan de integratie van beide processen om een meer holistische visie op het proces te introduceren?

3. Hoe moet het proces van requirements engineering worden ingericht om aan te sluiten bij de Scrum methode?
   a. Hoe zien de afzonderlijke processen er theoretisch uit?
   b. Hoe kunnen deze processen worden gemodelleerd?
   c. Op welke manier kunnen de processen worden geïntegreerd?

4. Welke behoeften en inzichten uit de praktijk zijn van belang en kunnen bijdragen aan een succesvolle implementatie?

1.2.3 Conceptueel model

Het conceptueel model in Figuur 1 toont op een schematische wijze de opbouw van het onderzoek:



Figuur 1 Conceptueel model

Het theoretisch kader beschrijft de resultaten van het literatuuronderzoek en vormt de basis van het procesmodel. Het empirisch deel beschrijft het praktijkonderzoek. De inzichten uit de praktijk zijn in het model verwerkt en er wordt afgesloten met conclusies, aanbevelingen en een reflectie.

1.3 Leeswijzer

De opbouw van dit document volgt de stappen zoals weergegeven in het onderzoeksmodel.

Er wordt gestart met een uitgebreid theoretisch deel waarin alle relevante theorie uit de literatuurstudie is toegelicht. Zo begint hoofdstuk twee logischerwijs met een introductie op het softwareontwikkelproces en diverse activiteiten die daarin worden uitgevoerd. Vervolgens worden de begrippen requirements en requirements engineering toegelicht, gevolgd door enige achtergrond over processen, technieken en kwaliteitskenmerken die hierop van toepassing zijn.

Hoofdstuk drie introduceert diverse methoden om het ontwikkelproces daadwerkelijk vorm te geven en verschillende strategieën die hier mogelijk zijn. Aangezien requirements theorieën veelal op een traditionele werkwijze gebaseerd zijn wordt deze context eerst beschreven. Vervolgens wordt een introductie gegeven op het Agile gedachtegoed en de verschillende stromingen daarbinnen.

Nadat de Agile principes en concepten zijn geïntroduceerd volgt in hoofdstuk vier een uitgebreide analyse van de Scrum ontwikkelmethode. Diverse eigenschappen, rollen, processen en tools zijn binnen dit hoofdstuk toegelicht. Middels een aantal procesmodelleringen is het Scrum ontwikkelproces gedetailleerd in kaart gebracht en zijn requirements gerelateerde activiteiten geïdentificeerd en geëvalueerd.

Als notatievorm voor deze procesmodellen is gebruikgemaakt van BPMN[2] 2.0. Deze notatievorm is de laatste jaren populair bij het modelleren van bedrijfsprocessen en maakt het mogelijk om processen en activiteiten op een overzichtelijke manier grafisch weer te geven. Een toelichting op de syntax van gebruikte modelelementen is terug te vinden in bijlage één.

Om een brede visie op het algehele proces te construeren is in hoofdstuk vijf het Lean gedachtegoed geïntroduceerd alsmede een Lean visie op requirements en requirements processen.

Hoofdstuk zes vormt de conclusie van het theoretisch kader. Verschillende concepten uit de literatuurstudie zijn hier samengebracht en vertaald naar een onderbouwde visie en concept van een geïntegreerd model.

Het twee deel van dit document beschrijft het empirisch onderzoek. In hoofdstuk zeven wordt het onderzoeksgebied en de methode van onderzoek toegelicht, gevolgd door de resultaten van het onderzoek en daaruit afgeleide conclusies.

Het derde deel van dit document is toegespitst op evaluatie bestaat uit drie onderdelen. Als eerste worden conclusies getrokken ten aanzien van het praktijkonderzoek. Vervolgens wordt het concept model uit het theoretisch kader geëvalueerd en aangevuld aan de hand van inzichten die uit het praktijkonderzoek zijn opgedaan. Tot slot worden algemene conclusies getrokken ten aanzien van het onderzoek en eventueel vervolgonderzoek.

[2] Business Process Model and Notation

Deel 1: Theoretisch kader

De probleemstelling uit hoofdstuk 1.1 beschrijft knelpunten van praktische aard die ontstaan bij het volgen van een Agile strategie en werkwijze. Om een goed beeld te kunnen vormen van de context waarbinnen het probleem zich afspeelt zijn in dit deel de drie theoretische pijlers uiteengezet die relevant zijn binnen deze thesis, te weten:

- Requirements en Requirements Engineering (deelvraag 3a-b)
- Agile Software Development (deelvraag 1a)
- Scrum (deelvraag 1a)
- Lean Software Development (deelvraag 2)

2 Software requirements in context

In deze sectie wordt een inleiding gegeven op het concept *software requirements* en het proces van *requirements engineering* binnen de context van softwareontwikkeling.

2.1 Inleiding

Het ontwikkelen van software is een specialistische aangelegenheid waarbij vaak verschillende disciplines benodigd zijn om van idee tot uitgewerkte oplossing te komen. Wanneer we kijken naar daadwerkelijke handelingen (of activiteiten) die nodig zijn om een softwareproduct te vervaardigen, dan kunnen we grofweg de volgende activiteiten onderscheiden:

- Het verzamelen en analyseren van requirements;
- Het ontwerpen van het product;
- Het bouwen van het product;
- Het testen van het product;
- Het opleveren van het product;
- De nazorg

Deze activiteiten worden hieronder toegelicht:

- Het verzamelen en analyseren van requirements:
  Voordat gestart kan worden met de technische processen, moet duidelijk zijn wat er precies ontwikkelt moet worden. De wensen en eisen (requirements) voor het systeem worden daarom geïnventariseerd, vastgelegd en met de opdrachtgever geverifieerd. Er vinden in dit proces dan ook verschillende interpretatie- en transformatieslagen plaats om van een idee tot een meer geformaliseerde beschrijving en uitwerking van een klantwens te komen. De beschrijving dient in eerste instantie om wensen kenbaar te maken aan systeemontwikkelaars, zodat zij het product op de juiste manier kunnen ontwerpen en implementeren. Centraal staat dan ook de vraag “wat” precies de behoefte van de klant is. Ook wordt vaak de beweegreden, het “waarom”, beschreven om een beter inzicht te verschaffen in de context en het doel van de opdrachtgever. Deze beschrijving wordt ook de “requirements specificatie” genoemd. Naast het afstemmen en opstellen

van de specificatie wordt ook nadrukkelijk aandacht besteed aan de functionele consistentie tussen requirements onderling en/of reeds bestaande delen van het product.

- Het ontwerpen van de oplossing:
  Nadat de requirements voldoende duidelijk zijn, wordt antwoord gegeven op de vaag “hoe” de systeemwens geïmplementeerd zou moeten worden. Hiertoe kunnen verschillende ontwerpactiviteiten plaatsvinden die met name de technische en/of grafische kaders van het systeem (of de wijziging) beschrijven. Dit wordt ook wel de technische specificatie of design genoemd.

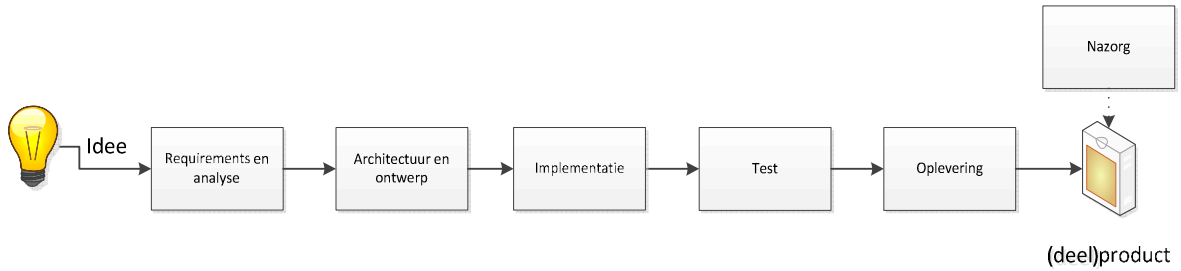
- Het bouwen van de oplossing:
  Op basis van de beschikbare informatie en kennis wordt de software ontwikkelt die het probleem voor de klant zou moeten oplossen.

- Het testen van de oplossing:
  Nadat de oplossing is ontwikkeld, moet worden gecontroleerd of deze voldoet aan de eerder gestelde criteria en of het uiteindelijke resultaat van juiste kwaliteit is.

- Het opleveren
  Nadat de ontwikkeling van een (deel)product gereed is kan deze worden opgeleverd. Dit proces omvat typisch het valideren met de opdrachtgever, bundelen en het in bedrijf nemen van het (deel)product.

- Nazorg
  De lifecycle voor het product stopt meestal niet wanneer het in productie is genomen. Gedurende het daadwerkelijke gebruik van de software ontstaan vaak nieuwe wensen en komen onherroepelijk bugs aan het licht die als onderhoudswerkzaamheden (of doorontwikkeling) kunnen worden beschouwd. Afhankelijk van de productstrategie wordt in meer of mindere mate geïnvesteerd in nazorg. Een typisch softwareproduct heeft echter vaak een levensduur van enkele jaren. De kans is daardoor groot dat een product zal moeten mee evalueren met de vraag en behoefte van de markt om een optimaal rendement uit de investering te behalen. Ook de ontwikkeling van nieuwe technologieën spelen daarbij een rol.

In het algemeen volgen de activiteiten elkaar sequentieel op, ongeacht de toegepaste ontwikkelmethodiek. Gebreken of defecten kunnen echter zorgen voor terugkoppeling naar eerdere stappen, waarna een deel van de cyclus opnieuw doorlopen moet worden. Er ontstaat zo een lineair model, wat de status van individuele systeemwensen gedurende het proces beschrijft. We zouden dit kunnen beschouwen als de *life cycle* of *value stream* van softwareontwikkeling aangezien iedere stap waarde toevoegt aan het uiteindelijke product of de kwaliteit daarvan.



Figuur 2 Value stream software product

Deze logische opeenvolging van activiteiten hoeft niet te betekenen dat het ontwikkelproces zelf ook op een dergelijke manier wordt gefaseerd. Zoals verderop in deze thesis duidelijk zal worden, kunnen weldegelijk afwijkende projectfasen en aanpak in het proces worden gehanteerd. Afhankelijk of de ontwikkelmethodiek een lineair, dan wel iteratief karakter heeft. Er wordt daarom bewust onderscheid gemaakt tussen de term “fase” en “activiteit.

Aangezien individuele activiteiten behoorlijk complex en specialistisch kunnen zijn, wordt een ontwikkelteam vaak samengesteld uit specialisten uit verschillende disciplines zoals: programmeurs, testers, analisten en architecten. Verantwoordelijkheden worden dan in de vorm van rollen verdeeld onder de teamleden. Binnen wat grotere organisaties zijn typisch specialisten actief op ieder individueel vakgebied. In kleinere organisaties worden rollen vaak gecombineerd.

Afhankelijk van de benodigde kwaliteit, behoeften en strategie, wordt in meer of mindere mate nadruk gelegd en geïnvesteerd in de verschillende activiteiten. Vanwege de vaak grote variatie in complexiteit en omvang van individuele systeemwensen en de daarbij benodigde expertise is het cruciaal om een goed afgestemde balans te vinden met betrekking tot de capaciteit en middelen die binnen ieder specialisme wordt geïnvesteerd.

Het gehele proces van softwareontwikkeling wordt vaak gezien als het proces van ontwerp en het daadwerkelijk coderen (implementatie), ofwel de engineering processen. In het algemeen krijgen deze activiteiten dan ook meer aandacht en expertise dan bijvoorbeeld de requirements- en testactiviteiten (Kulak & Guiney, 2004) Er ontstaat daardoor een onbalans in de focus op het proces, wat tal van knelpunten kan introduceren welke ten grondslag liggen aan de probleemstelling van deze thesis. Een juiste focus aanbrengen op het requirements proces is dus essentieel.

2.2 Software Requirements

2.2.1 Definitie

In het kort beschrijven software requirements de systeemwensen en eisen waaraan het product moet voldoen. Een meer formele definitie luidt:

> *"Requirements are the effects that the computer is to exert in the problem domain, by virtue of the computer's programming."*—Benjamin L. Kovitz

Requirements definiëren dus de effecten die een artefact moet hebben op het probleemgebied.

Een minder formele definitie luidt:

> *"A requirement is something that a computer application must do for its users."*—Kulak & Guiney

Een requirement beschrijft daarbij een specifieke functie, functionaliteit of kwaliteitseis waaraan het systeem moet voldoen en tevens de interactie met de gebruikers. Requirements stellen daarmee de scope van het product of individuele wens vast. (Kulak & Guiney, 2004)

2.2.2 Soorten requirements

Een requirement kan vele aspecten beschrijven. Er wordt daarom onderscheid gemaakt tussen verschillende soorten requirements. Op hoofdniveau wordt onderscheid gemaakt tussen functionele– en niet-functionele requirements.

2.2.2.1 Functioneel

Requirements kunnen functioneel van aard zijn. Ze beschrijven dan functies waaraan het systeem moet voldoen of die het aan de gebruiker moet bieden. Functionele requirements kunnen het volgende beschrijven:

- Gedrag
- Gegevens
- Foutafhandeling
- Dynamiek
- Presentatie
- Interactie
- Interfaces

2.2.2.2 Niet-functioneel

Niet functionele requirements beschrijven impliciete eigenschappen waaraan het systeem moet voldoen. Voorbeelden van dergelijke eigenschappen zijn wensen met betrekking tot:

- Betrouwbaarheid;
- Gebruiksvriendelijkheid;
- Efficiëntie;
- Onderhoud baarheid;
- Performance;
- Flexibiliteit in overdraagbaarheid (*portability*);

- Ontwikkelrichtlijnen of standaarden.

(In bijlage vijf is een uitgebreider overzicht te vinden van zogenaamde non-functional requirements.)

Afgezien van het onderscheid in functioneel en niet-functionele requirements kan ook onderscheid worden gemaakt in de verschillende stadia waarin requirements zich kunnen bevinden met betrekking tot de technische diepgang en doelgroep ten behoeve van communicatie. We onderscheiden daarbij User Requirements, System Requirements en de Design Specificatie.

2.2.2.3 User Requirements

Allereerst worden klantwensen geïnventariseerd en vastgelegd in de vorm van User Requirements. Deze requirements representeren de functionele of niet-functionele klantwensen (of features) voor het product vanuit de visie van de klant of gebruiker. User requirements worden typisch in natuurlijke taal beschreven en eventueel aangevuld met tabellen, diagrammen of afbeeldingen. User Requirements staan centraal in de conversatie met de klant en dienen daarom: begrijpelijk, goed leesbaar en gemakkelijk zijn te interpreteren door niet-technici.

2.2.2.4 System Requirements

User Requirements zijn in veel gevallen te highlevel en onvoldoende concreet om direct te worden omgezet in een technische implementatie. User Requirements worden daarom verder geanalyseerd en uitgewerkt naar een meer technische beschrijving waarbij rekening gehouden wordt met de opbouw van het systeem, integratie met andere- of reeds bestaande delen van het systeem en alle geldende beperkingen. Een systeem requirement beschrijft in meer detail wat de wensen en/of wijzigingen op het systeem moeten zijn en vormt daarmee het functioneel ontwerp.

2.2.2.5 Design Specification

Nadat de requirements op user- en systeemniveau duidelijk zijn kan worden gestart met het technisch ontwerp. De wensen worden verder geanalyseerd en er wordt antwoord gegeven op de vraag hoe de wens het beste geïmplementeerd kan worden. Zoals eerder aangegeven beschrijft deze stap het design (ontwerp) van de software en valt daarmee buiten de formele definitie van requirements. De design specification beschrijft in detail de technische aanpak en vormgeving van een systeemwens.

2.2.3 Kwaliteit

De kwaliteit van requirements wordt binnen het proces geborgd door in het specificatieproces nadrukkelijk aandacht te besteden aan de volgende activiteiten:

- Het oplossen van conflicterende requirements;
- Elimineren van redundante requirements;
- Beperken van het volume;
- Borgen van traceerbaarheid.

Het is erg lastig om te bepalen of een requirement van voldoende kwaliteit is. Dit is namelijk afhankelijk van vele factoren en kan voor iedere requirement verschillen. Enkele gemeenschappelijke kenmerken waaraan goede requirements voldoen zijn:

- Identificeerbaar;
- Atomair (niet deelbaar);
- Traceerbaar (te herleiden);
- Correcte taal (leesbaarheid doelgroep);
- Concreet (concreet taalgebruik, geen onduidelijke zinnen);
- Geaccordeerd (door opdrachtgever);
- Beschrijft wat, niet hoe (voorkom aannames met betrekking tot technisch ontwerp)
- Consistent (onderlinge consistentie);
- Uniform beschreven;
- Haalbaar;
- Specifiek;
- Meetbaar.

2.3 Requirements Engineering

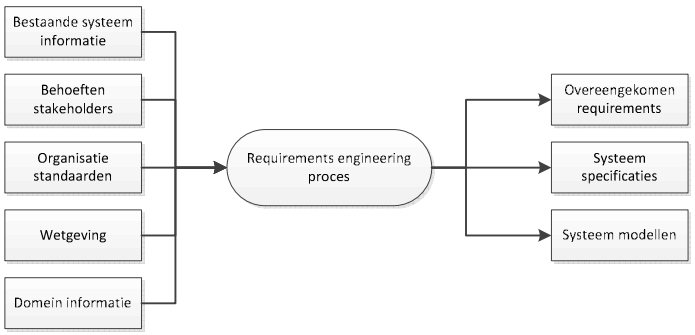
De totstandkoming van de requirements gebeurt in een proces genaamd 'requirements engineering'. Doel van dit proces is het overbruggen van het gat tussen het probleem en de oplossing. Hierbij worden alle, veelal informele, wensen geïdentificeerd en vertaalt naar een concrete geformaliseerde beschrijving.

Het proces van requirements engineering beperkt zich tot het beschrijven van het probleem, het wat en waarom. Het systeem wordt als black-box beschouwd en alleen het gedrag en interactie met de omgeving wordt beschreven. Het bepalen van de daadwerkelijke oplossing of oplossingsrichting vindt plaats in de ontwerp- of designfase en valt buiten het domein de requirements engineer. Omdat in de design fase in het algemeen veel technisch-inhoudelijke kennis en diepgang vereist is, wordt deze activiteit meestal door ontwikkelaars en/of architecten uitgevoerd en los gezien van de requirements.

Bij het uitwerken van requirements is echter ook sprake van ontwerp, maar dan op functioneel niveau. We onderscheiden daarom het functioneel ontwerp (F.O.) wat volgt uit het requirements proces, en een technisch ontwerp (T.O) wat volgt uit de design fase.

2.3.1 Processen

Het proces van requirements engineering is een exploratief proces. Dat wil zeggen, wensen liggen meestal niet voor het oprapen maar moeten vaak nog ontdekt worden. Uit de ervaringen met het watervalmodel (zie sectie 3.1.2) is gebleken dat de klant in veel gevallen niet in staat is om een complex systeem direct te kunnen overzien en alle wensen vooraf te communiceren. Er zijn dan vaak meerdere communicatiecycli nodig om wensen helemaal helder te krijgen. Afgezien van de klantinput zijn er ook verschillende andere bronnen die input leveren aan het requirementsproces. Het input/output model voor het requirements engineering proces kan als volgt worden weergegeven:



Figuur 3 Input/output model van requirements engineering proces.

Logischerwijs kan het proces van requirements engineering worden opgedeeld in de volgende deelprocessen:

1- Elicitatie;
2- Specificatie;
3- Validatie.

Afhankelijk van de scope kunnen ook deelprocessen als analyse/conceptualisatie of afstemming/onderhandeling worden onderscheiden, maar deze worden in het verloop van deze thesis als integraal onderdeel van het proces beschouwd.

2.3.1.1 Elicitatie

Elicitatie staat voor ophelderen, loskrijgen of ontlokken van informatie. Binnen dit deelproces worden alle relevante bronnen geraadpleegd om de exacte wensen te inventariseren en de systeemgrenzen vast te stellen. Het begrijpen van de context waarin het probleem speelt is essentieel om succesvol het systeem te kunnen ontwikkelen. Omdat slechts een deel van de informatie kan worden aangedragen dient zowel actieve, als reactieve elicitatie plaats te vinden om een volledig beeld te vormen van de wensen en eisen van het systeem. Typische activiteiten die plaatsvinden in het elicitatie proces zijn: het identificeren van stakeholders, het vaststellen van wensen en behoeften en het vaststellen van de context. Een deel van de analysefase wordt in dit proces doorlopen middels het doorspreken en ophelderen van de requirements.

Technieken die typisch gebruikt worden bij elicitatie zijn onder andere: het afnemen van interviews met stakeholders, brainstorm of andere groepssessies om wensen boven water te krijgen, probleem analyse, business rule analyse, scenario analyse, prototyping of checklists.

2.3.1.2 Specificatie

In het specificatieproces worden wensen en eisen zorgvuldig geanalyseerd en op een gestructureerde manier vastgelegd. Het primaire doel van dit proces is het communiceren van stakeholder wensen (requirements) richting de ontwikkelaars, maar dient ook als basis om de geïnterpreteerde wensen met de stakeholders te evalueren en te valideren. De requirements specificatie moet daarom goed te begrijpen en leesbaar zijn voor de opdrachtgever (en eventueel andere stakeholders) en wordt derhalve in natuurlijke taal opgesteld.

Een software systeem (of deel daarvan), wat vaak erg complex van aard is, in natuurlijke (en tevens begrijpelijke) taal beschrijven kan erg problematisch zijn. In tegenstelling tot meer formele talen, die in computers worden gebruikt, kan natuurlijke taal moeilijk exact worden uitgedrukt en zijn zinnen of zinsneden gemakkelijk voor meerdere interpretaties vatbaar. Het vormen van een steekhoudende specificatie, die vaak ook binnen een groot geheel moet vallen en gelijktijdig met tal van zaken rekening moet houden, is daarom erg lastig.

Er zijn dan ook diverse technieken en methoden om deze specificatie op te bouwen en gereed te maken voor technische analyse. Deze kunnen variëren van formeel en meer technisch van aard, tot erg pragmatisch[3]. De onderstaande tabel toont een overzicht van enkele traditionele, meer gestructureerde technieken:

Activity	Technique
Function Identification	Event Lists Entity Life Histories
Function Organisation	Data Flow Diagrams Transformation Schema Actigrams
Function Specification	Structured English Decision Tables Decision Trees State-Transistion Diagrams Transition Tables Precondition-Postconditions
Entity Identification	"Spot the nouns in the description"
Entity Organisation	Data Structure Diagrams Data Schema Entity-Relationship Diagrams
Entity Specification	Data Dictionary

Tabel 1 Gestructureerde analyse technieken (ESA, 1995)

Daarnaast is ook functionele decompositie een methode waarmee functionaliteit in kaart kan worden gebracht en omgezet kan worden in een specificatie. Deze methode wordt echter nog maar weinig toegepast omdat een dergelijke functionele benadering meestal niet in lijn is met de vaak object georiënteerde benadering die tegenwoordig in de ontwikkelfase wordt toegepast.

Hoewel bovenstaande technieken weldegelijk goede eigenschappen hebben en voor specifieke doelen nog steeds zeer bruikbaar zijn, zijn hedendaagse methoden in deze fase van ontwikkeling vaak minder technisch en meer op de klant gericht.

Binnen de context van deze thesis selecteren we twee klantgerichte requirements methoden die tegenwoordig veel gebruikt worden, te weten: Use Cases en User Stories. Deze worden hieronder kort toegelicht.

[3] Niet te verwarren met pragmatiek, wat binnen het requirements specialisme staat voor: "het mede in acht nemen, of beschouwen van de context van iets".

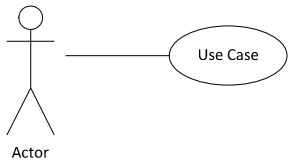
2.3.1.2.1 Use Cases

Use Cases beschrijven het gedrag van een systeem vanuit gebruikersperspectief. Use Cases richten zich op de interactie tussen het systeem en de gebruiker (of externe bron). Doormiddel van “actoren” en “events” worden initiator en gewenste acties en gedrag van het systeem beschreven. Bij het opstellen van Use Cases wordt typisch gebruikgemaakt van een Use Case Template. Hierin zijn onder andere de volgende velden opgenomen:

Naam	**De naam van de Use Case ten behoeve van identificatie.**
Status	De status of iteratie waarin de Use Case zich bevindt.
Samenvatting	Een korte samenvatting van de interactie die plaatsvindt in de Use Case.
Basic Course of Events	Beschrijft de interactie zoals die plaatsvindt in de “happy flow”.
Alternatieve scenario’s	Beschrijft minder gebruikelijke interactiepaden.
Exceptie scenario’s	Beschrijft het gedrag wanneer fouten optreden in de flow.
Triggers	De trigger van de Use Case. Een trigger is altijd een gebruiker of een extern systeem wat interactie initieert.
Aannames	Gedane aannames in de Use Case.
Pre condities	Voorwaarden waaraan moet worden voldaan, of die gelden wanneer de Use Case wordt geïnitieerd.
Post condities	Voorwaarden die moeten gelden na het doorlopen van de Use Case.
Business Rules	De business rules die van toepassing zijn op de Use Case.
Auteur	De auteur van de Use Case.
Datum	De datum of data waarop de Use Case is aangemaakt, gewijzigd en/of van status is veranderd.

Tabel 2 Use Case Template

In het algemeen worden richtlijnen (zoals beschreven in sectie 2.2.3) gebruikt om de kwaliteit van de specificatie te borgen.



Figuur 4 Use Case Diagram – basiselementen

Use Cases zijn onderdeel van de *Unified Modelling Language* (UML). UML beschrijft onder andere het Use Case Diagram waarin grafisch wordt weergegeven hoe Use Cases zich verhouden ten opzichte van elkaar en aanwezige actoren. Figuur 4 toont de basiselementen van een Use Case diagram. Daarnaast kunnen ook diverse soorten relaties en systeemgrenzen worden weergegeven. (Zie bijlage zes voor een voorbeeld Use Case diagram.)

Use Cases kenmerken zich door een vrij gedetailleerde benadering en uitwerking van de user requirements en richten zich met name op gebruikersinteractie.

2.3.1.2.2 User Stories

User Stories zijn oorspronkelijk een toepassing uit het eXtreme Programming paradigma (XP) maar worden ook in vele andere Agile methoden toegepast zoals Scrum. Een User Story richt zich op het in kaart brengen van een algemene wens van de klant. User Stories kenmerken zich door een betrekkelijk informele – en pragmatische benadering van requirements. Een User Story is opgebouwd uit de volgende onderdelen (bron: xprogramming.com, 2001):

- Een **card**:

  Dit is een kaartje, vaak in het formaat van een Post-It welke het mogelijk maakt om de User Story inzichtelijk te maken op het planningsbord. De card bevat slechts een beperkt deel van de informatie maar zou een korte doeltreffende titel moeten hebben en eventueel een korte introductie zodat deze gemakkelijk geïdentificeerd kan worden door het team.

- De **conversatie**:

  De conversatie beschrijft alle relevante communicatie met betrekking tot de wens. Binnen XP is het de *practice* dat requirements door de ontwikkelaar zelf worden opgehelderd bij de Product Owner Binnen Scrum wordt vooral gesproken over conversatie tussen de Product Owner en het team. De benodigde specificatie kan bestaan uit: notities, screenshots, diagrammen of schetsen, specificaties, delen uit e-mails et cetera. Zolang maar voldoende duidelijk is wat er ontwikkeld moet worden en binnen welke kaders de oplossing gevonden moet worden.

- De **confirmatie:**

  Voor iedere User Story worden acceptatiecriteria vastgesteld om ervoor te zorgen dat duidelijk is aan welke punten de oplossing moet voldoen. Nadat de functionaliteit geïmplementeerd is, wordt door de Product Owner een acceptatietest uitgevoerd om te valideren of de wens naar behoefte is opgelost.

User Stories beginnen vaak met een korte, samenvattende zin waarin zowel de wens als het doel wordt beschreven, alsmede de stakeholder die hiermee gediend wordt:

"Als [rol],zou ik graag willen dat [functie],zodat [doel] wordt bereikt"

Gevolgd door een verdere toelichting en andere (flexibele) vormen van informatie over de wens. Een User Story bestaat niet uit een vastgesteld formaat van requirements specificatie, maar kan wordt beschouwd als placeholder voor alle communicatie met betrekking tot de requirements voor een specifiek deel van het product. De informatie binnen deze placeholder wordt gedurende het proces uitgebreid en aangevuld met relevante gegevens.

Figuur 5 User Story Card (bron: leadingagile.com)

Om te bepalen of een User Story van goede kwaliteit is wordt gebruikgemaakt van de INVEST criteria. INVEST staat voor:

- *Independent*:

  Een goede User Story beschrijft een afgebakend deel van de functionaliteit en heeft geen afhankelijkheden met andere User Stories. Afhankelijkheden introduceren inflexibiliteit en geven problemen bij het inschatten, plannen en prioriteren.

- *Negotiable*:

  Een User Story beschrijft de conversatie over bepaalde functionaliteit en is geen contract. Totdat de User Story in een sprint is opgenomen moet deze ten alle tijden worden kunnen herzien of gewijzigd.

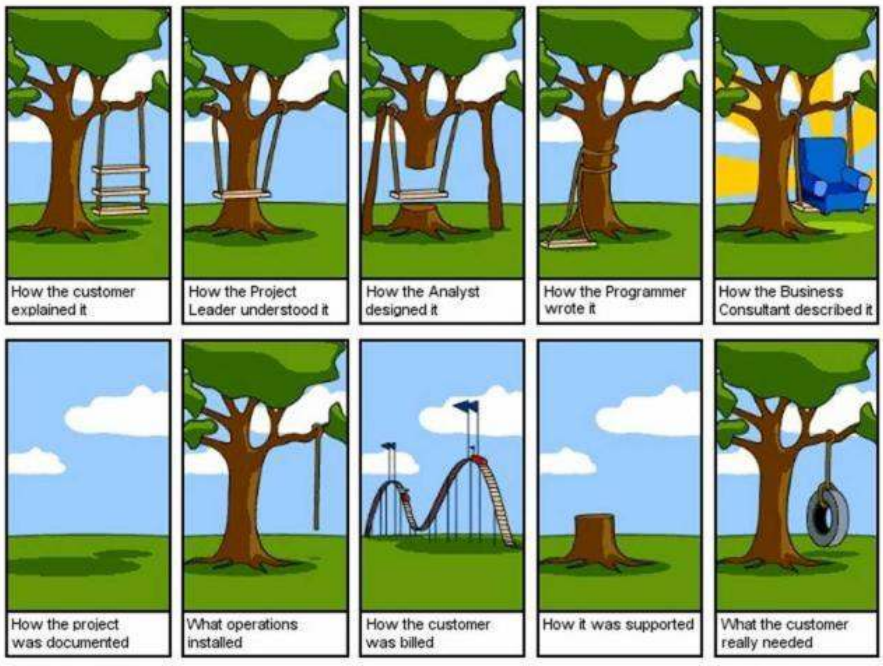
- *Valuable*:
  Een User Story moet waarde voor de gebruiker (stakeholder) hebben. Op basis van deze waarde worden User Stories op de product backlog geprioriteerd. Een User Story die geen waarde representeert zou verworpen moeten worden.
- *Estimable*:
  Het moet voor ontwikkelaars mogelijk zijn om een goede inschatting te kunnen maken van de omvang van de werkzaamheden. Hiertoe dient voldoende informatie beschikbaar te zijn. Deze inschatting wordt gebruikt om de product backlog op een juiste manier te prioriteren.
- *Small*:
  De omvang van een User Story moet beperkt zijn. User Stories moeten gemakkelijk binnen een iteratie kunnen worden gerealiseerd. Een iteratie bevat bij voorkeur ook meerdere User Stories om voldoende flexibel te zijn en weinig afhankelijkheden te hebben. Hier geldt het keep-it-simple principe, het streven is naar kleine, simpele en overzichtelijke brokken functionaliteit. Te grote User Stories (Epics) moeten worden opgesplitst in meerdere User Stories.
- *Testable*:
  Het moet mogelijk zijn om de User Story te testen. Er moeten test cases kunnen worden afgeleid en acceptatiecriteria moeten helder zijn om kunnen bepalen wanneer de User Story als “done” kan worden aangemerkt.

2.3.1.3 Validatie

Wanneer de specificatie voldoende is afgerond kan deze worden gevalideerd. Het doel van validatie is het controleren op juistheid en het verkrijgen van goedkeuring. Binnen dit proces wordt geverifieerd of de specificatie een juiste beschrijving is van de wensen voor het systeem. Doormiddel van een review of requirements-test wordt de juistheid en compleetheid van de requirement gevalideerd bij één of meerdere stakeholders. Als de requirement in orde is wordt hiervoor goedkeuring afgegeven.

2.3.2 De *challenge*

Het volledig kunnen doorgronden van de klantwensen en behoeften, en deze op een doeltreffende manier communiceren naar het ontwikkelteam is het primaire doel van requirements engineering. Zoals Figuur 6 in de vorm van een spotprent laat zien, treden over de gehele linie vaak interpretatiefouten op die leiden tot een product wat niet goed aansluit bij de werkelijke behoeften van een klant. Communicatie en een efficiënte informatieoverdracht zijn daarom essentieel. Requirements specificaties helpen om deze overdracht van informatie te borgen.



Figuur 6 Spotprent softwareontwikkeling - bron: knovelblogs.com - The importance of requirements engineering (2012)

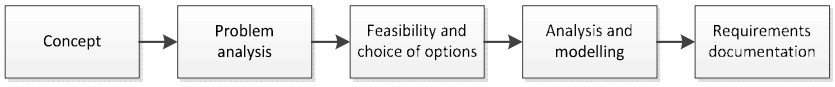
Het op een nauwkeurige en zorgvuldige manier eliciteren, specificeren en valideren van systeemwensen is een specialistische aangelegenheid en een vakgebied op zich. Er zijn binnen software organisaties dan ook meestal wel requirements engineers of analisten in dienst om deze taak voor hun rekening te nemen. Deze hebben vaak ook de rol om opdrachtgevers en klanten te begeleiden in het proces naar de opbouw van het product.

2.3.3 Proces strategie

Net als het ontwikkelproces kan het requirements engineering proces op verschillende manieren worden aangepakt. Er zijn dan ook diverse procesmodellen mogelijk die ofwel linear-sequentieel het proces beschrijven, ofwel een meer iteratief karakter hebben. De begrippen lineair, iteratief en incrementeel worden hieronder toegelicht.

2.3.3.1 Lineair

Lineair wil zeggen dat de verschillende fasen na elkaar, in volgorde worden doorlopen. Figuur 7 geeft een lineair model weer. De verschillende processen worden hierin sequentieel (van links naar rechts) doorlopen om tot de specificatie te komen.



Figuur 7 Lineair Requirements Engineering model (bron: Macaulay 1996)

2.3.3.2 Iteratief

Iteratieve processen zijn processen die korte herhalingscycli bevatten. Het proces wordt niet in één enkele cyclus doorlopen, maar opgeknipt in kleinere tijdseenheden (iteraties). Binnen de context van requirements engineering betekent dit dat requirements in een aantal terugkerende stappen worden opgesteld waarbij de specificatie binnen iedere iteratie in volwassenheid groeit. Binnen de context van softwareontwikkeling wordt hiermee de volwassenheid van het product bedoeld welke binnen een aantal iteraties wordt opgebouwd.

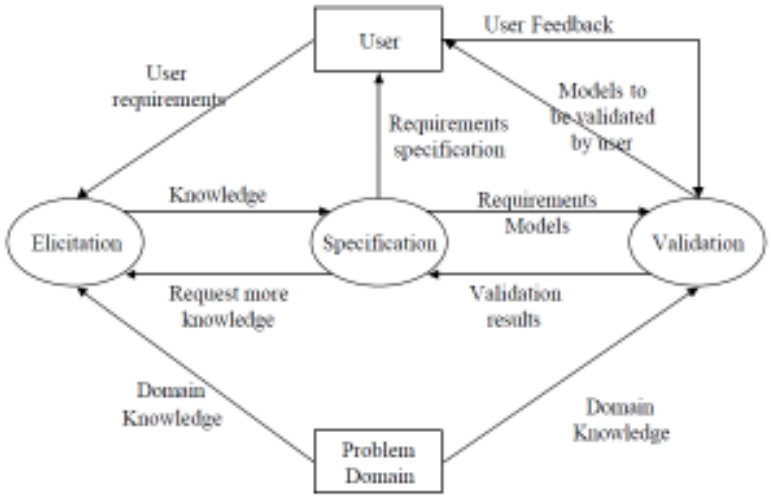
2.3.3.3 Incrementeel

Incrementeel houdt in dat een artefact uit verschillende delen (increments) wordt opgebouwd en/of opgeleverd. Iedere nieuwe increment is daarbij een uitbreiding of wijziging op hetgeen reeds gerealiseerd was.

Een incrementele aanpak zorgt ervoor dat het product niet ineens, maar in onderdelen kan worden opgeleverd. Hierdoor hoeft niet gewacht te worden tot het product helemaal gereed is, maar kan de ingebruikname al starten wanneer een bepaald volwassenheidsniveau is bereikt. Het product kan vervolgens in een later stadium worden aangevuld met overige functionaliteit.

Binnen iteratieve methoden wordt typisch iedere iteratie een increment van het eindproduct opgeleverd. Dit hoeft echter niet het geval te zijn. Er zijn ook iteratieve methoden die de output na iedere iteratie verwerpen en opnieuw beginnen om een optimale kwaliteit en efficiënte opbouw te stimuleren.

Een iteratief procesmodel van requirements engineering wordt weergegeven in Figuur 8. Het beschrijft een dynamisch proces, waarbij het mogelijk is om in korte iteraties de requirements tot stand te laten komen. Daarbij biedt het de mogelijkheid om terug te gaan naar vorige fasen, net zolang tot dat de requirement gereed en geaccordeerd is.



Figuur 8 Iteratief Requirements Engineering model (bron: Loucopoulos and Karakostas 1995)

In de werkelijkheid is het proces van requirements engineering zeer dynamisch. Er zijn vaak diverse feedbackloops met de opdrachtgever (of andere stakeholders) nodig om tot een juiste beschrijving van de requirements te komen en requirements moeten in feite meegroeien in de conceptualisatiefase van het product en zijn daarin continu aan veranderingen onderhevig. Iteratieve requirements methoden zijn dan ook beter in staat om alle wensen en behoeften in kaart te brengen.

Kulak & Guiney (2004) beschrijven een iteratief proces om requirements op een gestructureerde manier op te stellen. Zij bepleiten het gebruik van Use Cases en hebben een methode ontwikkeld om deze binnen drie iteraties te ontwikkelen. Deze iteraties worden genoemd:

- **Façade**: omvat een highlevel analyse op hoofdlijnen
- **Filled**: omvat een bredere en meer diepgaande analyse
- **Focussed**: omvat een vernauwende en gedetailleerde analyse

Iedere iteratie introduceert een wat andere focus en mate van diepgang. Afgezien van de sterke nadruk die tijdens de iteraties wordt gelegd op het opstellen en uitwerken van Use Cases, wordt ook aandacht besteed aan tools die helpen om de meer contextuele aspecten binnen een project in kaart te brengen. Dit zijn onder andere:

- *Mission, vision, values (document);*
- *Statement of work;*
- *Risk analysis;*
- *Prototype;*
- *Use cases en use case diagrams;*
- *Business rule catalog.*

De aanpak van Kulak & Guiney (2004) beperkt zich nadrukkelijk tot de requirements processen. Het model gaat impliciet uit van een uitgebreide requirements fase en een vrij gedetailleerde uitwerking. Waarbij de nadruk wordt gelegd op de kwaliteit en volledigheid van de requirements specificatie. Design en andere activiteiten worden buiten beschouwing gelaten. Er wordt niet beschreven hoe de aansluiting of integratie met andere activiteiten zou kunnen plaatsvinden. Aangezien de levenscyclus van requirements niet stopt niet nadat de specificatie is opgesteld, maar ook gedurende het ontwikkelproces nog aan verandering onderhevig is, kan de aanpak slechts beperkt holistisch genoemd worden.

3 Agile Software Development

In dit hoofdstuk worden verschillende processtrategieën en methoden beschreven die invulling geven aan de in hoofdstuk 2 omschreven ontwikkelactiviteiten. Daarbij staan met name de introductie en het achterliggende gedachtegoed van de Agile beweging centraal.

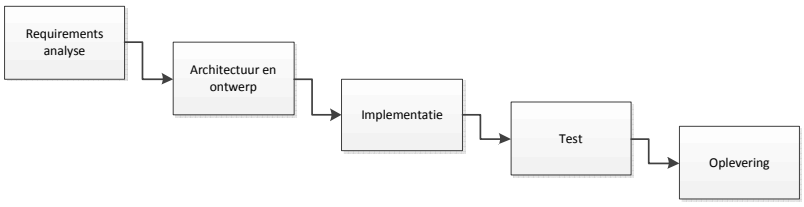
3.1 Achtergrond en definitie

Het ontwikkelen van softwaresystemen kan worden beschouwd als een zeer complexe aangelegenheid waarbij erg veel factoren een rol spelen in het uiteindelijke slagen van een project. Wanneer gekeken wordt naar de succesratio's van softwareprojecten in vergelijking tot andere vakgebieden, blijkt het erg lastig om dit proces succesvol te doorlopen. De *Standish Group Chaos Reports* concludeerde dat in 1994 slechts 16% van de projecten succesvol werd afgerond, 31% werd voortijdig afgebroken en bij het overige deel werden substantiële projectdoelen niet gehaald. In 2004: 29% succesvol, bij 18% afgebroken (Interview: Jim Johnson of the Standish Group) en in 2010: 37% succesvol, bij 21% afgebroken. Dit impliceert een significant risico voor bedrijven en instellingen ten aanzien van de investering en zorgt veelal voor frustraties bij alle deelnemers in het proces. (Sutherland & Schwaber, The Crisis in Software: The Wrong Process Produces the Wrong Results, 2012)

3.1.1 De watervalmethode

Sinds de opkomst van de informatietechnologie (IT) worstelt men met het controleren en beheersen van het software ontwikkelproces. Om meer grip te krijgen op dit proces is begin jaren zeventig een sequentiële methode geïntroduceerd, ook wel bekend als de watervalmethode. Er zijn vele varianten bekend (zoals het V-model), maar de essentie is dat volgens deze methode, complexe systemen het beste op een gefaseerde, sequentiële en planmatige manier gebouwd kunnen worden. Binnen de methode worden achtereenvolgens de volgende stappen doorlopen:

Er wordt gestart met het verzamelen en definiëren van alle systeemeisen (requirements). Deze worden met veel detail uitgewerkt en door de opdrachtgever geaccordeerd. Vervolgens wordt het systeem in zijn geheel ontworpen en pas daarna geïmplementeerd, getest en opgeleverd. Om de kwaliteit te borgen vindt eventueel tussen iedere fase een controle plaats om te beoordelen of het systeem nog voldoet aan alle requirements. Het proces wordt in die zin lineair gefaseerd op basis van de ontwikkelactiviteiten (zoals beschreven in sectie 2.1).



Figuur 9 Waterval model

Kenmerkend voor de watervalmethode is dat iedere fase eerst wordt afgerond, voordat een nieuwe wordt gestart. De fasering is strikt en het is dan ook niet mogelijk om terug te gaan naar een vorige fase.

Wijzigingen in de requirements worden vermeden omdat dit de planning volledig verstoort en het project in gevaar brengt. Requirements staan voor het hele project dus vast.

Het op deze manier opsplitsen van het proces brengt een duidelijke structuur aan die ervoor zorgt dat er eerst goed nagedacht wordt over wat er precies gedaan moet worden, vervolgens over hoe het doel het beste bereikt kan worden, om vervolgens pas te worden uitgewerkt, getest en opgeleverd. Vanuit menselijk oogpunt is dit een zeer logische en veel toegepaste benadering voor het oplossen van complexe problemen.

3.1.2 Het probleem met waterval

Het watervalmodel is in de jaren zeventig, tachtig en negentig vrij populair geweest. Met name omdat het proces in de periode daarvoor veelal ongestructureerd was. Hoewel de succesratio's slecht waren, hebben managers en opdrachtgevers vrij lang vastgehouden aan de methode en getracht deze van binnenuit te verbeteren door meer energie te steken in het plannen en managen van het proces en afspraken zoveel mogelijk in contractvorm vast te leggen. De problemen werden hierdoor echter zeker niet minder. De knelpunten in deze methode zijn als volgt vastgesteld:

- *Toename behoefte aan flexibiliteit*

  Nieuwe markten hebben zich de laatste decennia in hoog tempo ontwikkeld en worden steeds dynamischer. Dit leidt tot een kortere commerciële levensduur van producten. Innovatieflexibiliteit en een korte time-to-market worden daarin steeds belangrijker. Dit stelt steeds hogere eisen aan de productontwikkeling. Klanten verlangen de laatste technische snufjes, perfecte kwaliteit, snelle en tijdige levering en tegen lage kosten. Om te kunnen concurreren zullen bedrijven daarom snel en flexibel op veranderingen in de markt moeten inspelen. Productontwikkeling zal daarop moeten aansluiten.

  De watervalmethode gaat ervanuit dat complexe systemen in één enkele sequentie van handelingen gebouwd kunnen worden zonder dat de requirements gedurende het proces hoeven te worden herzien of rekening moet worden gehouden met veranderende condities in de business of op het gebied van technologie. Gezien de vaak lange doorlooptijd van projecten zorgt dit voor onvoldoende flexibiliteit, waardoor de klantbehoeften niet goed worden vervuld.

  Tevens wordt er vanuit gegaan dat de opdrachtgever bij aanvang volledig duidelijk heeft en kan toelichten wat de uiteindelijke requirements van het systeem moeten worden. Dit is vaak niet realistisch.

- *Big bang!*

  Er wordt gedurende het proces geen tussentijdse feedback verkregen of verwerkt. Het product wordt in principe na de laatste fase opgeleverd en aan de klant gepresenteerd. Het risico dat het product dan niet, of niet volledig, voldoet aan de verwachtingen van de klant is hierdoor groot.

Ook kan het product of de technologie na oplevering achterhaald zijn omdat softwareprojecten maanden tot jaren doorlooptijd kunnen hebben.

- *Contractvorm*

  De requirements worden in de beginfase met de opdrachtgever vastgesteld. Dit is een tijdrovend en vermoeiend proces waarbij vele omvangrijke documenten meerdere malen moeten worden doorgenomen om alle details vast te leggen en te accorderen. In de praktijk blijkt dat dergelijke omvangrijke documenten niet, of niet goed, gelezen worden. Het risico van foute interpretatie werkt daarom direct door op het eindproduct.

  Om risico's zoveel mogelijk te beperken worden afspraken en requirements vaak in contractvorm vastgelegd. Beide partijen dekken zich hiervoor zo goed mogelijk in. Gedurende het proces zorgt dit voor veel frustraties en een grimmige klant-leverancier verhouding.

- *Testen in de eindfase*

  Een ander nadeel van de watervalmethode is dat al het testwerk wordt uitgesteld tot het laatste moment. Bugs worden hierdoor pas in een laat stadium ontdekt waardoor de impact hiervan groot is. Het benodigde re-work leidt gemakkelijk tot onvoorziene vertraging in het project, het uitsluiten van functionaliteit of extra kosten.

  De benodigde capaciteit en planning wordt bepaald door het samennemen van alle schattingen en doorlooptijden. Uitloop in de ene fase betekent dus automatisch problemen in de volgende fase(n). Met name testactiviteiten en documentatie hebben een hoog risico om in het gedrang te raken omdat een accurate schatting voor een periode ver in de toekomst erg lastig te maken is en deadlines vast staan.

- *'Over de muur' cultuur*

  Doordat alle fases strikt gescheiden zijn is er weinig samenwerking tussen de verschillende disciplines. Dit zorgt ervoor dat teams zich maar beperkt verantwoordelijk voelen voor het eindproduct en uiteindelijk kunnen streven naar eilandoptimalisatie. Documentatie is de primaire bron voor communicatieoverdracht tussen verschillende disciplines en speelt daarom een belangrijke rol in de uiteindelijke kwaliteit van het product.

Deze factoren zorgen ervoor dat de watervalmethode, binnen de huidige markten en met de huidige behoeften, minder geschikt is voor het ontwikkelen van software en steeds minder vaak wordt toegepast.

De watervalmethode wordt tegenwoordig vaak aangehaald als voorbeeld van hoe softwareontwikkeling juist niet moet. Dit is maar ten dele terecht want in specifieke toepassingen kan een dergelijke werkwijze weldegelijk doeltreffend en soms zelfs noodzakelijk zijn. Dit is met name het geval voor sterk gereguleerde omgevingen, waar consequenties van eventuele fouten erg groot zijn (bijvoorbeeld levens

in gevaar kunnen brengen, of grote financiële risico's dragen). Het proces moet dan vaak aan strenge voorschriften of standaarden en procedures voldoen. Een Agile werkwijze is dan meestal niet geschikt. Voorbeelden van dergelijke omgevingen zijn: kritieke toepassingen in de lucht- en ruimtevaart, automobielindustrie of toepassingen in de medische wereld.

Ook als het niet mogelijk is voor de opdrachtgever om in voldoende mate betrokken te zijn bij het ontwikkelproces (een fundamentele voorwaarde in Agile methoden) kan beter voor een andere, meer traditionele, benadering gekozen worden.

3.1.3 Introductie van Agile

Halverwege de jaren negentig is men meer gaan nadenken over nieuwe methoden die het proces anders en beter zouden moeten vormgeven en het proces van softwareontwikkeling succesvoller zouden moeten maken. Hierbij kwamen iteratieve modellen meer onder de aandacht. Ook werd veel gebruikgemaakt van ervaringen en best practices die in de loop der jaren binnen softwareontwikkeling zijn opgedaan. Ook werd inspiratie gehaald uit strategieën en ervaringen van andere vakgebieden.

Hieruit is een nieuw gedachtegoed ontstaan waarin de overtuiging voortvloeit dat betere resultaten worden bereikt met een aanpak waarbij meer rekening wordt gehouden met de menselijke aspecten en beperkingen, meer aandacht is voor het leeraspect, het stimuleren van innovatie en waarbij de *customer value* en het aanpassingsvermogen centraal staan. Vanuit dit gedachtegoed zijn verschillende methoden voortgekomen die met de parapluterm 'Agile' worden aangeduid. Agile betekent vrij vertaald: flexibel of lenig.

Agile Software Development kan worden beschouwd als een conceptueel raamwerk bestaande uit een aantal waarden en principes. De principes binnen Agile methoden leggen de nadruk op samenwerking en het bouwen van werkende software waar mensen in een vroeg stadium ervaring mee kunnen opdoen en hun feedback kunnen terugkoppelen. De klant wordt actiever betrokken in het proces. Door te werken met cross-functionele teams kan in korte iteraties het product incrementeel worden opgebouwd en kan de klant voortdurend bijsturen richting het gewenste eindproduct.

3.2 Agile Manifesto

In februari 2001 is een groep van 17 specialisten en wetenschappers in Utah (V.S.) bij elkaar gekomen om nieuwe ontwikkelingen te bespreken en een alternatief te vinden op het tot dan toe gebruikte, documentatie gedreven en omvangrijke ontwikkelproces. Centraal stond de vraag: "Hoe kunnen we het ontwikkelproces beter vormgeven zodat we succesvoller worden in het leveren van software?". Hieruit is het Software Development Manifesto (latere benaming: Agile Manifesto) ontstaan wat een aantal principes beschrijft die gebaseerd zijn op succesvol gebleken theorieën en best practices uit verschillenden disciplines en vakgebieden. Er werd een gemeenschappelijke visie bereikt op de volgende punten:

- *Individuen en interacties boven processen en hulpmiddelen*
  Een goede samenwerking binnen teams is cruciaal in het proces van softwareontwikkeling. Het beste resultaat wordt behaald door zelfsturende teams waarbij verschillende disciplines en de

business betrokken zijn. Communicatie is aangemerkt als belangrijkste factor in succesvolle samenwerking en wordt de driver voor productiviteit en kwaliteit genoemd. De meest efficiënte manier om informatie te communiceren is face-to-face.

Het belang van een goede en efficiënte communicatie gaat boven dat van het volgen van processen.

- *Samenwerking met de klant boven contracten en onderhandeling*
  Betrek de opdrachtgever nauw en geef hem de mogelijkheid om gedurende het proces bij te sturen. Het van te voren vastzetten van requirements en contracten is nadelig voor het uiteindelijke product.

- *Werkende software boven uitgebreide documentatie*
  Er wordt gestreefd naar het snel opleveren van belangrijke functionaliteit en werkende software in korte iteraties. Hierdoor wordt het product incrementeel opgebouwd. Uitgebreide documentatie is ondergeschikt aan een werkend product. Pas als software helemaal af is en in productie kan worden genomen geeft deze een daadwerkelijke indicatie van de voortgang.

- *Aanpassen op veranderingen boven het uitvoeren van een plan*
  Door te werken in korte iteraties kan veel feedback van de opdrachtgever worden verkregen. Hoe eerder wijzigingen bekend worden, hoe gemakkelijker en goedkoper de oplossing over het algemeen is. Doordat het team focust op de iteratie en niet te ver vooruit kijkt kan het flexibel blijven ten aanzien van wijzigende wensen van de opdrachtgever.

Het Agile Manifesto is een belangrijke motor geweest in de Agile beweging. Het gaf een goede representatie van de eigenschappen en waarden binnen Agile methoden en de Agile filosofie. De principes lijken vanzelfsprekend maar adresseren precies die aspecten die binnen de softwareontwikkeling vaak voor problemen zorgen. Het manifest geeft duidelijk aan welke elementen extra nadruk en aandacht zouden moeten hebben ten opzichte van andere aanwezige elementen in het proces.

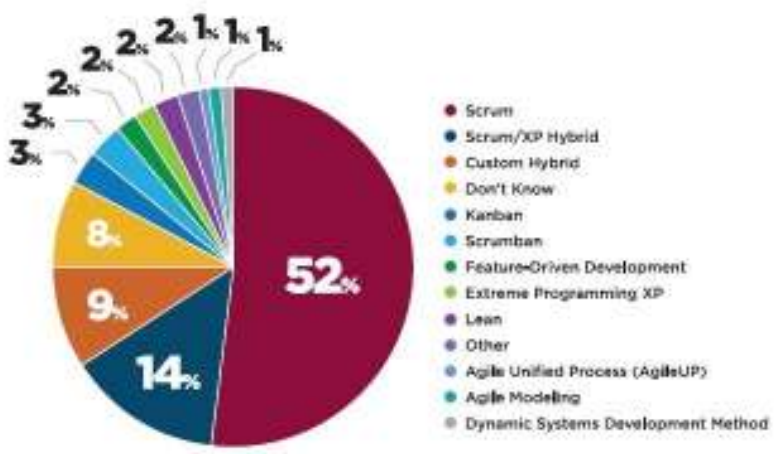
3.3 Agile ontwikkelmethoden

Agile ontwikkelmethoden zijn met name gefocust op hoe teams werken bij het produceren van software. Welke principes en praktijken helpen teams en welke waarde wordt door het team gecreëerd. Ze hebben daarbij voornamelijk een software engineering en software delivery focus.

Er zijn verschillende Agile methoden waarvan de bekendste zijn: Scrum, XP, Kanban, Crystal, FDD, DSDM en tegenwoordig wordt ook steeds vaker Lean Software Development in dit rijtje genoemd. Ook zijn er verschillende hybride vormen mogelijk.

Agile methoden worden tegenwoordig veel toegepast binnen organisaties. De *State of Agile Survey 2011* toonde aan dat 80% (van 6042 respondenten) Agile methoden gebruikt bij het ontwikkelen van software. Wanneer gekeken wordt naar het gebruik en toepassing van de verschillende Agile methoden is Scrum al enige tijd het meest populair. Uit eerdergenoemde survey bleek dat 66% van alle Agile

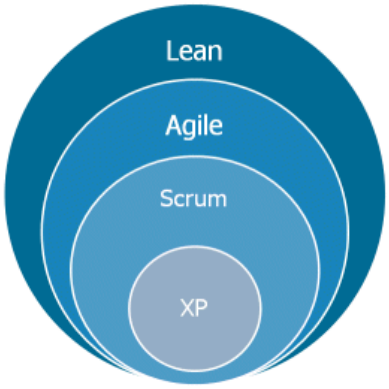
implementaties doormiddel van Scrum, al dan niet in combinatie met XP, werd uitgevoerd. (VersionOne, 2011)



Figuur 10 Gebruik van Agile methoden (bron: State of Agile Survey 2011, Version One Inc.)

Recent onderzoek gaf aan dat dit percentage in Nederland op zo'n 92% ligt. (Xebia, 2012) Dit dominante aandeel maakt Scrum de meest relevante ontwikkelmethode binnen de context van dit onderzoek en is dan ook gebruikt als uitgangspunt voor de verdere uitwerking.

Hoewel de verschillende Agile methoden vele overeenkomsten hebben en gebaseerd zijn op een gemeenschappelijke visie op het proces van softwareontwikkeling, zijn er weldegelijk verschillen met name op het toepassingsgebied en abstractieniveau. Zo richt eXtreme Programming zich voornamelijk op de totstandkoming van software (de daadwerkelijke engineering processen) en biedt tal van *best practices* om individuele ontwikkelaars efficiënt te laten samenwerken en goede kwaliteit code te leveren. Scrum beschouwt het proces op een wat hoger abstractieniveau en richt zich met name op het efficiënt plannen en uitvoeren van projecten en de daarvoor benodigde samenwerking tussen betrokken partijen.

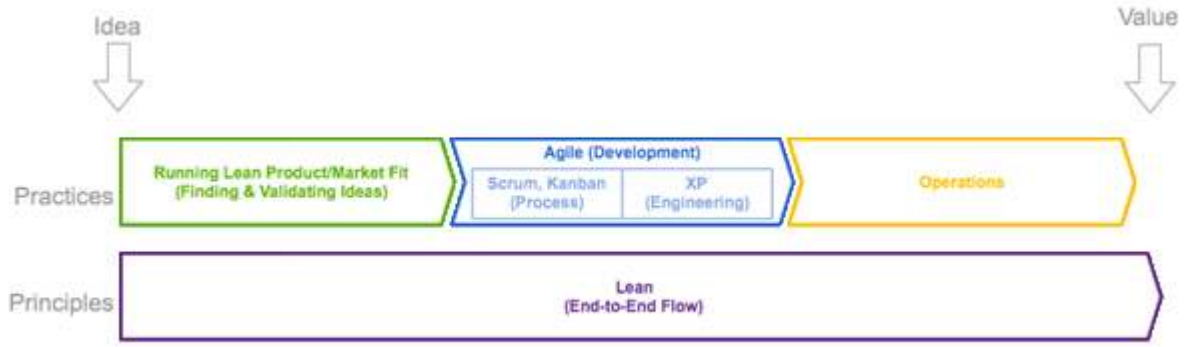


Figuur 11 Abstractieniveau van methoden (bron: blog.crisp.se)

Lean introduceert een meer holistische kijk op het ontwikkelproces en beschouwt het proces op organisatieniveau. De gehele value stream, van idee tot daadwerkelijk opgeleverd product staat daarin centraal. Veel principes van Lean Software Development zijn duidelijk terug te vinden in het Agile gedachtegoed en in de Scrum ontwikkelmethode. Het duidelijke verschil is dat Lean een top-down benadering hanteert, vanuit de organisatie. Agile en Scrum redeneren bottom-up, vanuit het team. (Robert Charette)

Lean, Agile en Scrum belichten als het ware ieder een ander deel van het spectrum, maar liggen weldegelijk in elkaars verlengde. Figuur 11 toont de verschillende begrippen ten opzichte van het abstractieniveau.

Een gouden combinatie ontstaat wanneer zowel aandacht is voor het proces op organisatieniveau (waarbij gestuurd wordt op de flow van waarde binnen de organisatie), als gelijktijdig een aansluitende aanpak op team niveau wordt gehanteerd. Om deze reden is het concept Lean Software Development in deze thesis dan ook onderzocht en gebruikt om de big-picture te zien van de individuele processen op team niveau.



Figuur 12 Samenhang Scrum, Lean en Agile (bron origineel: agileweboperations.com)

4 Scrum

In dit hoofdstuk wordt een toelichting gegeven van de Scrum ontwikkelmethode en het proces wat door Scrum beschreven wordt. Omdat dit proces de basis vormt voor het uiteindelijke procesmodel zijn alle relevante aspecten (en deelprocessen) van de methode in deze sectie uiteengezet, alsmede de visie en de metafysische eigenschappen van de methode om een zo goed mogelijk beeld te vormen van de context waarin de probleemstelling zich afspeelt.

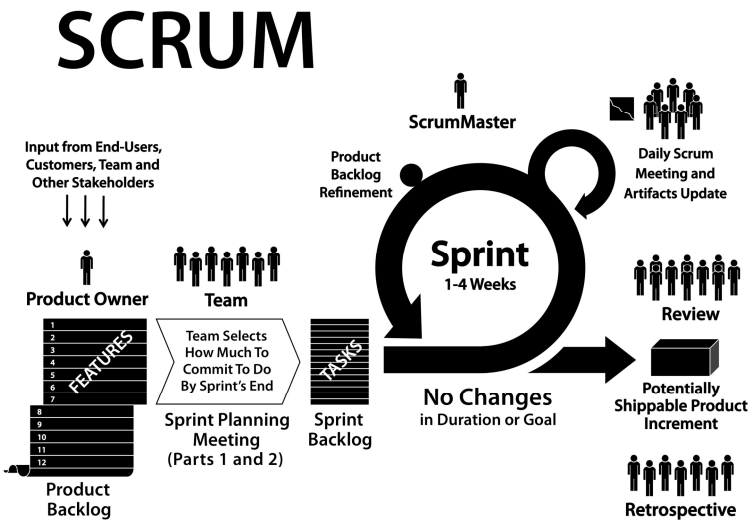
4.1 Inleiding

Er is veel geschreven over Scrum en Scrum practices. Om de methode zuiver te analyseren zijn in deze sectie met name publicaties van Jeff Sutherland en Ken Schwaber (grondleggers van Scrum) gebruikt als bron voor het beschrijven van de Scrum werkwijze. (Sutherland & Schwaber, The Scrum Papers: Nuts, Bolts, and Origins of an Agile Framework, 2011) (Scrum Inc, 2012) (Barton, Schwaber, & Rawsthorne, 2005) (Sutherland, Scrum Handbook, 2011)

4.2 Algemeen

Scrum is een Agile ontwikkelmethodiek die de laatste jaren erg populair is. Het wordt zowel binnen kleine organisaties, als bij grote software bedrijven als onder andere: Microsoft, Google, Yahoo en IBM en vele anderen veelvuldig toegepast bij het ontwikkelen van software producten. Vandaag de dag groeit Scrum nog steeds in populariteit en wordt inmiddels ook voor toepassingen buiten de IT gebruikt.

Scrum is een iteratieve- en incrementele methode waarbij het proces in feite wordt opgedeeld in korte iteraties met een vaste duur (Sprints). Een Sprint duurt doorgaans één tot vier weken. Binnen een Sprint werkt het team in nauwe samenwerking met de opdrachtgever (Product Owner) aan een deel van de gewenste functionaliteit en vindt na iedere iteratie een deeloplevering plaats. Het eindproduct wordt hierdoor incrementeel opgebouwd en kan gedurende dit proces voortdurend worden bijgestuurd.



Figuur 13 Scrum (bron: Sutherland & Schwaber, 2011)

De naam Scrum is afkomstig uit de rugbysport waarbij spelers zich na een dood spelmoment organiseren in een hechte formatie om gezamenlijk door te breken en een zo groot mogelijke afstand af te leggen met de bal. Hoewel een analogie met de ontwikkelmethodiek duidelijk te maken is op het gebied van team spirit, heeft Scrum als methode gelukkig geen van de geforceerde en riskante kenmerken. Scrum richt zich juist op het beperken van de risico's en het bewaren van de rust en focus binnen het team. De turbulente omstandigheden waarbinnen IT projecten vaak verkeren worden in zekere zin geaccepteerd en zoveel mogelijk beperkt door binnen een duidelijk afgebakende scope te werken en flexibel met veranderingen om te gaan.

Scrum is geen exacte methode of gedetailleerd proces, maar kan beter als procesmethode of framework voor productontwikkeling worden beschouwd. Scrum is gebaseerd op principes uit onder andere complexe (adaptieve) systeemtheorie, Lean manufacturing, kennismanagementstrategie en tal van best practices op het gebied van softwareontwikkeling. Scrum beschrijft in zekere mate het proces en de doelen die nagestreefd moeten worden. Een deel van de invulling wordt aan het team en de organisatie overgelaten. (Sutherland & Schwaber, The Scrum Papers: Nuts, Bolts, and Origins of an Agile Framework, 2011)

In het kort verloopt het proces als volgt. Bij de start van een sprint selecteert het team een aantal requirements uit een geprioriteerde lijst (de Product Backlog) en geeft commitment af om dit binnen de sprint te kunnen realiseren. Gedurende de sprint mogen de requirements niet meer wijzigen. Het team heeft hierdoor een tijdelijke, maar stabiele en overzichtelijke situatie en kan zich volledig focussen op de afgesproken set aan requirements. Dagelijks heeft het team een kort overleg (de *Daily Scrum*), waar de status en eventuele problemen worden besproken en het proces waar nodig wordt bijgestuurd om ervoor te zorgen dat het werk op tijd opgeleverd wordt en blokkades worden weggenomen. Na iedere sprint is er een oplevering en demonstratie aan de *stakeholders* en vindt tevens een evaluatie plaats. Hierbij wordt zowel het product, als het proces geëvalueerd. Figuur 13 geeft een globaal overzicht van het Scrum proces.

4.3 Eigenschappen en visie

4.3.1 Teams

Veel van de principes binnen Scrum en met name de visie over teams en samenwerking binnen teams is afgeleid van het werk van professor Nonaka en Takeuchi. Zij bestudeerde *high performing- en* cross-functionele teams bij bedrijven in sterk concurrerende markten (onder andere: Honda, Fuji, Xerox, NEC, Epson, Brother, 3M en HP).

In het paper '*The new new development game*' (Takeuchi & Nonaka, 1984) wordt geconcludeerd dat een veel beter resultaat in productontwikkeling kan worden bereikt door gebruik te maken van zelfsturende, cross-functionele teams. Door silo's weg te nemen en teamleden vanuit alle benodigde disciplines binnen één team te verzamelen, kan de beste oplossing worden gevonden voor het probleem. Om tot de beste oplossing te komen is input vanuit verschillende disciplines nodig. De zelfstandigheid van het team is tevens een belangrijke factor. Managers zouden teams scherpe doelstellingen moeten stellen en

vervolgens de handen moeten terugtrekken en het team zelfstandig met een oplossing laten komen. Hierdoor zal het team optimaal creatief zijn en zichzelf organiseren om uiteindelijk de meest efficiënte werkwijze te bereiken. (Takeuchi & Nonaka, 1984)

Al deze principes zijn terug te vinden in de manier waarop Scrum het teamaspect en samenwerking benadert. Hierbij worden aspecten als het proces, hiërarchie, structuur, management of leiderschap niet uitgesloten, maar op een andere (meer subtiele) manier benaderd.

4.3.2 Transparantie

De Scrum methode introduceert een hoge mate van transparantie in het proces. Transparantie tussen teamleden onderling, maar ook richting de opdrachtgever en management.

Na iedere iteratie (sprint) is direct zichtbaar hoe het proces verlopen is en wat het team heeft kunnen leveren binnen de gestelde timebox. Dit geeft de opdrachtgever regelmatig (en al in een vroeg stadium) terugkoppeling over de voortgang en geeft het een indicatie van de omvang van de nog resterende wensen en het tijdspad in de huidige situatie.

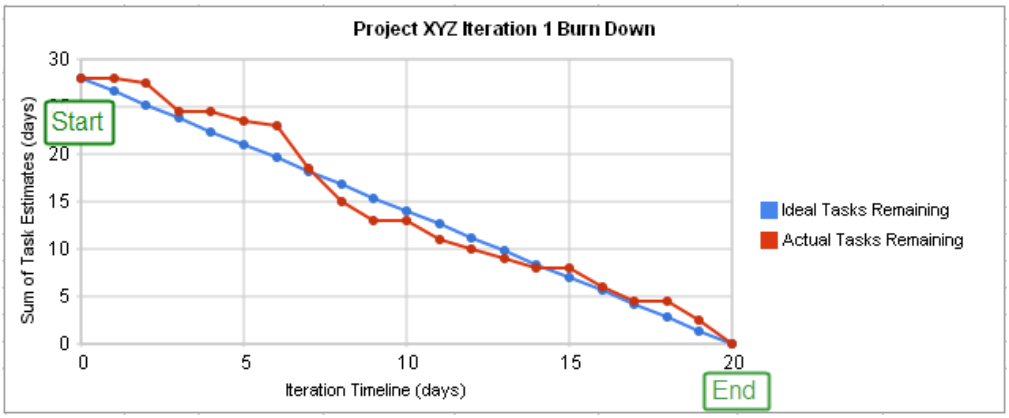
Door de dagelijkse synchronisatie van het team in de *Daily Scrum* is binnen het team ook al snel duidelijk wat de individuele bijdrage van teamleden is en hoe deze samenwerken met anderen. Dit kan voor conflicten in het team zorgen, maar legt een dergelijk probleem (en mogelijke stagnatie in het proces) wel direct bloot. De gezamenlijke verantwoordelijkheid zorgt ervoor dat teamleden elkaar scherp houden op het gebied van kwaliteit, productiviteit en samenwerking.

Bij het inschatten van de werkzaamheden door het team wordt gebruikgemaakt van zogenaamde *velocity points* (of story points). Dit zijn relatieve getallen (fictieve eenheden) die de omvang van het benodigde werk voor een klantwens representeren. Het gebruik van uren wordt sterk afgeraden omdat deze een directe relatie hebben tot kosten. De relatie tot kosten leidt af van de hoofdzaak, het op een realistische manier in kaart brengen van de hoeveelheid werk.

Doordat het team gedwongen wordt om te kiezen uit een vaste reeks getallen (een deel van de Fibonacci reeks) wordt het gedwongen om niet te diep (tot op het uur nauwkeurig) na te denken over de inschatting, maar snel een relatieve omvang te bepalen ten opzichte van een referentie story. Het team kan hierdoor zonder veel moeite een redelijk accurate schatting afgeven.

4.3.2.1 Burn Down Chart

De *burn down chart* is een grafiek die inzicht geeft in de overkoepelende status van de sprint. De *burn down chart* wordt dagelijks bijgewerkt door het team en geeft het verloop van de benodigde hoeveelheid werk weer gedurende de sprint. Hierbij wordt het totaal aantal resterende uren (of totaal aantal resterende User Stories) uitgezet tegen de tijd.

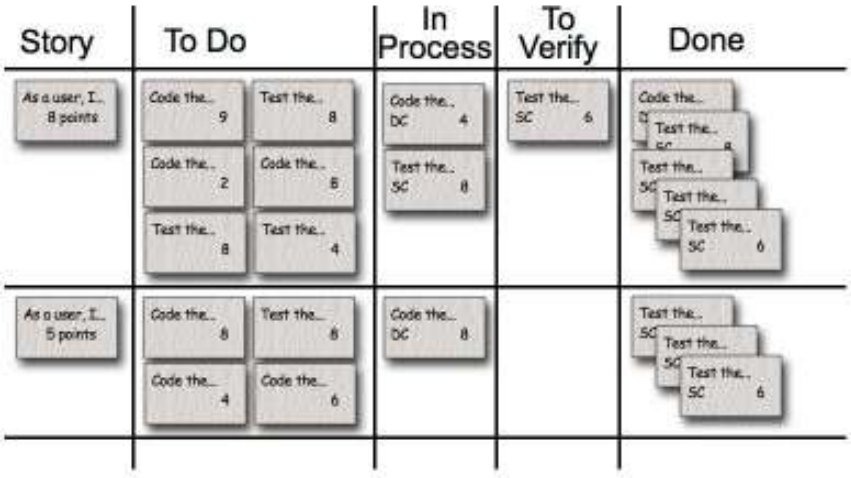


Figuur 14 Burn down chart

Een ideale lijn kan worden vastgesteld en geeft het streeftempo aan om alle werkzaamheden binnen de tijd af te ronden. In de grafiek kan ten alle tijden eenvoudig worden afgelezen of het team op schema ligt en of het einddoel haalbaar is binnen de gestelde tijd.

4.3.2.2 Scrum Board

Het Scrum bord is een visuele weergave van de sprint backlog waarbij de status van zowel de individuele taken, als User Stories, in één oogopslag zichtbaar zijn. Alle sprintactiviteiten worden in de vorm van briefjes of kaartjes op het bord bevestigd. Het bord is opgedeeld in een aantal kolommen die de status representeren. De exacte invulling is aan het team, maar ten minste moet worden aangegeven of aan de ontwikkeling van een taak begonnen is ("todo" dan wel "in progress"), of een taak in de test- of verificatiefase zit ("to verify"), of dat een taak is afgerond ("done"). Eventuele tussenfasen (als bijvoorbeeld ten behoeve van een *peer review*) kunnen worden toegevoegd indien behoefte is aan een gedetailleerdere status. Het bord is bij voorkeur fysiek aanwezig in de ruimte waar het team werkt, maar kan ook digitaal zijn. Hiervoor zijn diverse tools beschikbaar.



Figuur 15 Scrum bord

Het Scrum bord wordt (ten minste) dagelijks bijgewerkt door het team en geeft ten alle tijden een actuele weergave van de status van alle activiteiten weer.

Het Scrum bord is een zeer bruikbaar middel om eventuele problemen of vertragingen inzichtelijk te maken. De User Stories zijn in volgorde van prioriteit op het bord van boven naar beneden weergegeven. Bij een goed verloop van de sprint zouden alle kaartjes zich dus van boven naar beneden, van links naar rechts moeten verplaatsen. Ook kan aan het Scrum bord eenvoudig worden afgelezen of aan te veel activiteiten tegelijk wordt gewerkt of dat werk zich ergens opstapelt. Het team, of de Scrum Master zal in dat geval direct actie ondernemen om ervoor te zorgen dat de flow zo optimaal mogelijk is. In uiterste gevallen kan worden geëscaleerd (zie sectie: 4.7.3 - Escalatie).

4.3.3 Feedback

Binnen Scrum is veel aandacht voor het leeraspect van teams en individuen. De continue feedbackcyclus van '*inspect en adapt*' staat centraal in de methode en stelt het team in staat om zich voortdurend te kunnen aanpassen op de omstandigheden om het proces beter, efficiënter en prettiger te laten verlopen.

4.3.4 'Hyper productivity'

Typisch wordt steeds meer van de softwareontwikkelaar gevraagd. Projectmanagers (in reguliere projecten) hebben vaak de neiging om het proces onder druk te zetten om belangrijke deadlines te halen. Dit gaat echter automatisch ten koste van de kwaliteit en zorgvuldigheid (een menselijk aspect). De beslissing hiertoe wordt vaak impliciet genomen om de klant of manager tevreden te houden, maar kan veel extra problemen in het proces introduceren.

Het ontwikkelen van software kan worden beschouwd als een creatief beroep. Afgezien van kennis, ervaring en inzicht is de output van het proces sterk afhankelijk van de creativiteit en het oplossingsvermogen van de ontwikkelaar. Er is dan ook geen lineair verband tussen de hoeveelheid moeite die in het proces gestoken wordt en het bereikte resultaat. Het opvoeren van de druk heeft hierdoor maar een beperkt effect. Goede ideeën en efficiënte oplossingen komen vaak juist ten tijde van ontspanning en rust. Door het team onder druk te zetten wordt zowel de creativiteit als zelfontplooiing beperkt en neemt de kans op fouten duidelijk toe. (Sutherland & Schwaber, The Scrum Papers: Nuts, Bolts, and Origins of an Agile Framework, 2011)

De vraag naar sneller, meer en betere producten zal altijd blijven bestaan. De filosofie binnen Scrum is dat hier het beste op geacteerd kan worden door het creëren van een constant tempo en de productiviteit van het team juist te vergroten door het uitsluiten van *waste*. De waste wordt zoveel mogelijk beperkt door volledig te focussen op de (door de klant aangegeven), belangrijkste functionaliteit en een pragmatische aanpak van het proces. Het constante tempo en herhalen van korte cycli zorgt tevens voor een betere voorspelbaarheid van het proces.

Het tempo van het team wordt ook wel de *velocity* genoemd. De *velocity* is een momentopname en wordt gemeten door het aantal story points wat binnen de sprint daadwerkelijk is gerealiseerd op te tellen. Deze statistische gegevens worden door het team gebruikt om de capaciteit van een volgende

sprint te bepalen en tevens door de Product Owner om de planning en prioritering bij te stellen en verwachtingen bij stakeholders te managen.

4.4 Rollen

Binnen Scrum worden de volgende rollen onderscheiden:

4.4.1 Product Owner

De opdrachtgever wordt vertegenwoordigd door de rol van Product Owner (P.O). Deze is onder andere verantwoordelijk voor het product en het maximaliseren van de *return on investment (R.O.I)*.

Een belangrijke taak van de Product Owner is het opstellen en onderhouden van de Product Backlog. De Product Backlog is een centrale, geordende lijst van alle wensen voor het product. Deze wensen kunnen zijn ingebracht door de P.O zelf, of door andere stakeholders. Het is aan de Product Owner om alle belangen en wensen van stakeholders en sponsors af te wegen en deze te vertalen naar de Product Backlog. Dit is een continu proces en de Product Backlog kan dan ook voortdurend aan wijzigingen onderhevig zijn. Ieder item op de backlog wordt tevens door de Product Owner voorzien van een geplande Business Value. Zie sectie 4.5.2.

Doormiddel van het aanbrengen van prioriteit wordt de lijst geordend en kan de aandacht worden gericht op de items die op dat moment het meeste van belang zijn. De Product Owner heeft door deze prioritering direct invloed op de volgorde van implementatie en dus de werkzaamheden van het team (op basis van de iteratie-intervallen). Voor de langere termijn (sprint overschrijdend) stelt de Product Owner een *roadmap* op en bepaalt daarbij de bijbehorende releaseplanning. Het managen van draagkracht met betrekking tot al deze beslissingen en een goede communicatie richting stakeholders is tevens een verantwoordelijkheid van de Product Owner

Richting het team heeft de Product Owner ook een zeer belangrijke rol, namelijk het communiceren van alle wensen en het nemen van beslissingen met betrekking tot het product en de totstandkoming daarvan. Dit zowel tijdens de plannings- of voorbereidingsfase, als tijdens het ontwikkelproces zelf. De Product Owner fungeert in feite als de schakel tussen het team en alle stakeholders en sponsors. Het team heeft op deze manier één centrale contactpersoon voor alle vragen en kan zich verder volledig richten op het ontwikkelen van de oplossing. De Product Owner dient gedurende het proces direct benaderbaar (en liefst ter plaatse aanwezig) te zijn om eventuele onduidelijkheden op te helderen zodat het team niet geblokkeerd wordt en zo efficiënt mogelijk kan werken.

Scrum beschouwt het ontwikkeltraject niet per definitie als project, maar als continu proces wat herhaald wordt totdat de wensen zijn voldaan, of besloten wordt om niet meer te investeren. Toch kan Scrum ook weldegelijk in projectvorm worden gebruikt. De traditionele rol van project manager wordt binnen Scrum bewust vermeden. Indien meer behoefte is aan projectmanagement, komen deze taken voor een deel bij de Scrum Master te liggen en voor een deel bij de Product Owner.

4.4.2 Team

Het team bestaat uit vijf tot negen personen en implementeert de wensen van de Product Owner.

Het team bevat alle benodigde expertise van verschillende disciplines en achtergronden. Er is een redelijke mate van vrijheid en het team is zelfsturend, maar is daarbij ook verantwoordelijk voor het halen van een sprint en de kwaliteit van het product. Het team werkt nauw samen en streeft naar een hoge team performance en efficiëntie, dit boven die van de individu. Het bepaalt zelf hoeveel werk het op de schouders neemt binnen een sprint en wat de onderlinge werkverdeling is. Door het herhaaldelijk maken van relatieve inschattingen (en het evalueren daarvan) wordt het team getraind om snel, accurate schattingen af te afgeven. De benodigde ontwikkelinspanning van het product kan hiermee inzichtelijk gemaakt worden. Het team is doorgaans naar één tot drie sprints redelijk in staat een betrouwbare inschatting te maken. Naarmate het proces vordert worden de inschattingen betrouwbaarder.

Scrum beschrijft dat het essentieel is om teamleden 100% toegewezen te hebben aan een sprint. Het parallel werken aan meerdere projecten of andere werkzaamheden veroorzaakt een significante beperking op de productiviteit van ontwikkelaars omdat deze continu moeten task-switchen en met veel complexe problemen gelijktijdig bezig zijn.

Afhankelijk van de behoeften kan het team bestaan uit ontwikkelaars, testers, analisten, user interface designers, database experts, architecten en/of andere kennisgebieden.

Toch wordt in Scrum geprobeerd om specifieke rollen zoveel mogelijk te vermijden. Er zou gestreefd moeten worden naar een team van multi-skilled developers die alle rollen in het proces kunnen invullen. De overdracht wordt op die manier nog verder beperkt. Het team heeft een gezamenlijke verantwoordelijkheid voor het resultaat en zouden elkaar zoveel mogelijk moeten helpen om de sprint tot een succes te maken. De "*Go where the work is*" mentaliteit. Hoewel een nobel streven, wordt ook onderkend dat dit in de praktijk niet altijd haalbaar is. Specialistische taken kunnen soms onvermijdelijk bij bepaalde individuen terecht komen vanwege specifieke expertise. Het vinden van de juiste balans in expertise en ervaring binnen het team is dus erg belangrijk.

Naast het ontwikkelen van de oplossing wordt het team ook betrokken bij het maken van keuzes voor oplossingen en levert hiervoor een waardevolle input aan de Product Owner. Deze laat zich door het team informeren over mogelijke oplossingen, inschattingen en logische opbouw van het product of architectuur. Gezamenlijk stemmen zij de eisen vast waaraan de software zou moeten voldoen. Het compleet maken van de requirements is dus een samenspel tussen het team en de Product Owner.

Doordat het team zelfsturend is bepaalt het zelf hoe dit proces het beste kan worden ingericht. Door met regelmaat te evalueren (zowel ten aanzien van het product, als het proces) en de werkwijze waar nodig bij te stellen, zal het team uiteindelijk streven naar de meest efficiënte werkwijze.

4.4.3 Scrum Master

De Scrum master is verantwoordelijk voor een succesvol verloop en de inzichtelijkheid van het Scrum proces. Hij zorgt dat de methode zo goed mogelijk wordt ingezet en begeleid het team waar nodig in dit proces. De Scrum master maakt deel uit van het team en heeft daarin een begeleidende rol. Tevens faciliteert hij het team en vangt alle onnodige interrupties af zodat het team focus kan houden op de

werkzaamheden. De Scrum Master organiseert meetings en zit deze voor, zorgt ervoor dat voldoende voorbereidingen zijn getroffen bij de start en afronding van een sprint en dat er een goede communicatie tussen het team en de opdrachtgever is.

De Scrum master zou geen traditionele (project) manager moeten zijn, maar bij voorkeur iemand met technische achtergrond die een goed overzicht heeft van het product, de werkzaamheden en alle gebruikte technieken. Afhankelijk van de behoeften van de organisatie en de volwassenheid, of zelfstandigheid van de team, kan de rol van Scrum Master variëren van meewerkend en faciliterend tot een meer coördinerende, begeleidende of sturende rol in het ontwikkelproces.

4.5 Product Backlog

De Product Backlog is een op basis van prioriteit geordende lijst van alle systeemwensen, bugs en andere taken die door het Scrumteam moeten worden uitgevoerd. De belangrijkste items met de hoogste prioriteit staan bovenaan de lijst.

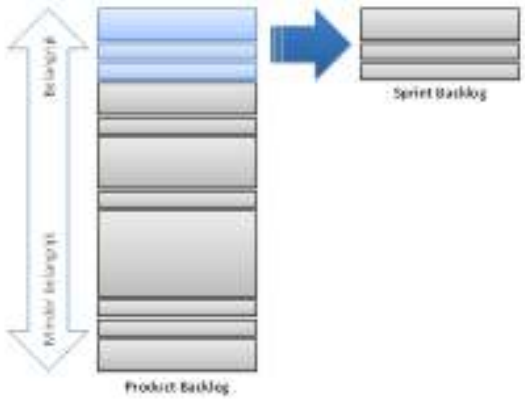
Idealiter zijn er geen afhankelijkheden tussen de items op de lijst en representeren deze ieder een duidelijk afgebakend deel van het product. Er wordt onderkend dat dit in de praktijk vaak lastig te bewerkstelligen is. Het team en de Product Owner zullen daarom voldoende aandacht moeten besteden aan het bewaken van onderlinge consistentie en afhankelijkheden tussen bepaalde activiteiten.

De Product Backlog introduceert flexibiliteit in het proces doordat de samenstelling en ordening van de lijst ten alle tijden kan worden aangepast. Hiermee kan op basis van de interval tussen iteraties worden gestuurd op de werkzaamheden van het team en de totstandkoming van het product.

“Scrum is not a big-bang, it’s like a guiding missile” - Jeff Sutherland (2011)

De requirements kunnen daardoor ook geleidelijk tot stand komen en hoeven niet allemaal ineens uitgewerkt te worden. Toch is het van belang om de Product Backlog zo snel mogelijk te vullen met alle tot dusver bekende wensen. De volledigheid van de backlog is met name van belang voor de prioritering en releaseplanning. Niet alle backlogitems hoeven tot in detail te zijn uitgewerkt, maar het hebben van overzicht is wel van belang om een goede afweging te kunnen maken met betrekking tot de prioriteit en investering.

Bij het bepalen van prioriteit is het tevens belangrijk dat de omvang van de werkzaamheden en de benodigde resources voor een klantwens inzichtelijk zijn. Scrum faciliteert hierin door gebruik te maken van story points (velocity points). De initiële impactanalyse maakt het mogelijk om al in een vroeg stadium de R.O.I. in kaart te brengen en zorgvuldiger afwegingen te maken met betrekking tot verdere investering.



Figuur 16 Product Backlog, Sprint backlog (bron: whitehorses.nl)

Het bovenste deel van de Product Backlog zou in voldoende mate uitgewerkt moeten zijn zodat het team efficiënt de planning- en ontwikkelfases kan doorlopen. Naarmate de prioriteit minder wordt, zijn items op de productbacklog typisch in een minder concreet stadium.

Een Product Backlog is in principe nooit compleet of klaar, maar altijd in beweging. Gedurende de levensduur van het product blijven er altijd nieuwe wensen of bugs binnenkomen. Enkel wanneer het product de 'end of life' fase in gaat droogt de backlog op of wordt deze geschrapt.

4.5.1 Items op de Product Backlog

De Product Backlog is een centrale verzamelplaats voor alle type wensen of wijzigingen voor het te ontwikkelen product.

Op de Product Backlog staan zowel hele grote wensen die zich nog in de conceptuele fase bevinden, als ook kleine meer concrete wensen. Een backlog item kan bestaan uit:

- User Story:

Zie toelichting in sectie 2.3.1. De omvang van de specificatie kan verschillen per User Story. User Stories met grote brokken functionaliteit, worden typisch opgesplitst in meerdere User Stories totdat het mogelijk is om ze in de sprint op te nemen en een concrete inschatting te maken van de werkzaamheden.

- Taak:

Het diepste niveau wat bereikt wordt is het taakniveau. Een taak beschrijft een concrete, afgebakende handeling welke ertoe bijdraagt om (een deel van) het probleem uit de User Story of Bug op te lossen. De omvang van een taak wordt ingeschat in uren. Taken zijn niet bedoeld om met stakeholders te communiceren, maar dienen zuiver voor het team om de handelingen inzichtelijk te maken en deze binnen het proces te volgen. Een taak kan daarom technisch beschreven zijn.

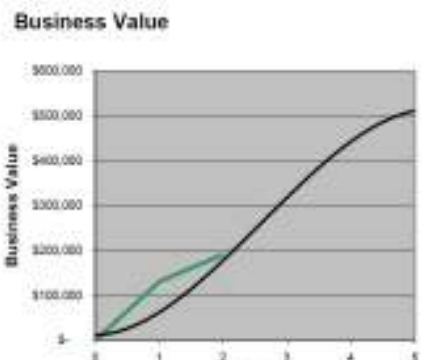
- Bug:

Naast alle items die te maken hebben met nieuwe wensen, worden ook bestaande problemen (bugs) op de productbacklog opgenomen. Een bug wordt typisch ook opgesplitst in één of meerdere taken om de stappen in het proces inzichtelijk te maken.

4.5.2 Business Value

Scrum beschrijft dat items op de backlog tevens voorzien zouden moeten worden van een business value. Doormiddel van de Business Value kan in kaart worden gebracht hoeveel waarde voor de Business daadwerkelijk wordt gecreëerd. In eerste instantie voorziet de Product Owner de individuele items op de backlog van een beraamde Business Value. Het team heeft daardoor inzicht in het belang van de wensen en kan zich richten op de items waarbij in korte tijd, zoveel mogelijk waarde gecreëerd kan worden.

Naast de Business Value hebben items op de backlog ook een prioriteit die uiteindelijk het belang aangeeft. Vaak is deze gebaseerd op de Business Value, maar ook andere factoren kunnen invloed hebben op de gewenste volgorde van implementatie. Het verloop en toename van de Business Value gedurende de sprints kan ten behoeve van rapportage inzichtelijk worden gemaakt met een Burn Up grafiek zoals te zien in Figuur 17. (Barton, Schwaber, & Rawsthorne, 2005)



Figuur 17 Business Value Burn Up (Barton, Schwaber, & Rawsthorne, 2005)

Nadat een backlog item volledig is opgeleverd kan een nacalculatie gedaan worden op de daadwerkelijk opgeleverde Business Value.

Sprint	Planned Business Value	Earned Business Value
0	$ -	$ -
1	$ 100,000	$ 130,814
2	$ 150,000	$ 191,861
3	$ 300,000	
4	$ 475,000	
5	$ 500,000	

Figuur 18 Business Value nacalculatie (Barton, Schwaber, & Rawsthorne, 2005)

4.6 Sprint Backlog

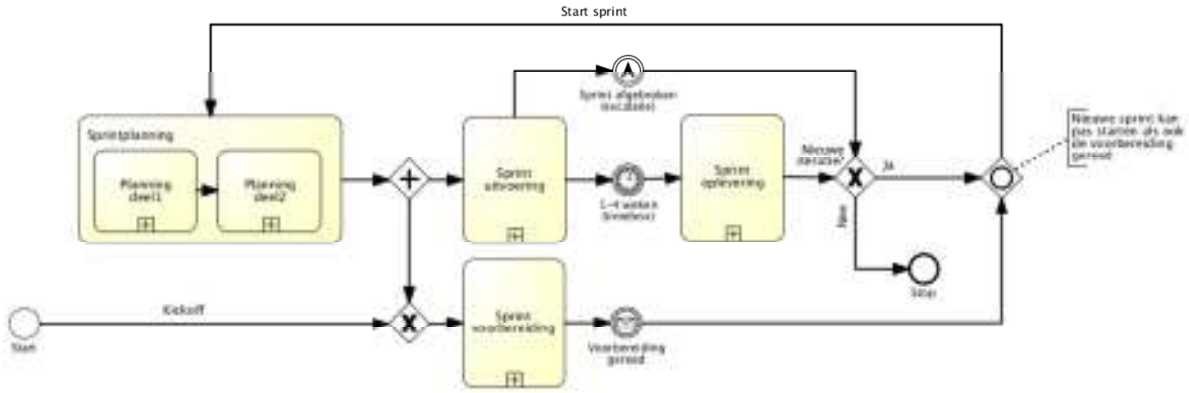
De sprintbacklog is de geprioriteerde lijst van backlogitems welke door het team geselecteerd zijn om binnen de sprint op te leveren. In tegenstelling tot de productbacklog is het voor de Product Owner niet toegestaan om wijzigingen in de sprintbacklog aan te brengen. Gedurende de sprint staan de scope dus vast. Direct bij de start van de iteratie splitst het team de geselecteerde User Stories op in concrete ontwikkelactiviteiten en taken en voegt deze toe aan de sprint backlog.

4.7 Het Scrum proces

Om te beoordelen hoe het proces van requirements engineering het beste geïntegreerd kan worden binnen de Scrum methode is het van belang inzicht te hebben in de verschillende activiteiten binnen

het, door Scrum beschreven, proces. Hiertoe zijn diverse procesmodellen opgesteld die de verschillende stappen in het Scrum proces in kaart brengen. Zoals vermeld in sectie 4.2 beschrijft Scrum een framework voor het proces, waarbij een deel van de invulling aan het team en de organisatie wordt overgelaten. De procesmodellen kunnen daarom niet als universeel worden beschouwd maar geven wel een goed beeld van de activiteiten en de volgorde van activiteiten die in het Scrum proces worden doorlopen.

Figuur 19 geeft een model van het proces op hoofdniveau weer waarbij de fundamentele deelprocessen worden weergegeven.



Figuur 19 Scrum procesmodel

In de volgende secties zal op ieder individueel deelproces worden ingezoomd om de activiteiten daarbinnen in kaart te brengen.

4.7.1 Sprintvoorbereiding

De voorbereidingsfase zoals beschreven in Figuur 19, wordt niet als zodanig beschreven in de Scrum methode. Wel worden verschillende activiteiten genoemd die voorafgaand aan de sprintplanning uitgevoerd zouden moeten worden. Ook beschrijft Scrum een aantal processen (met name voor de Product Owner) die buiten de sprint zouden moeten plaatsvinden. Aangezien al deze activiteiten gerelateerd zijn aan de voorbereiding van een nieuwe sprint, zijn deze als deelproces “voorbereiding” aangemerkt.

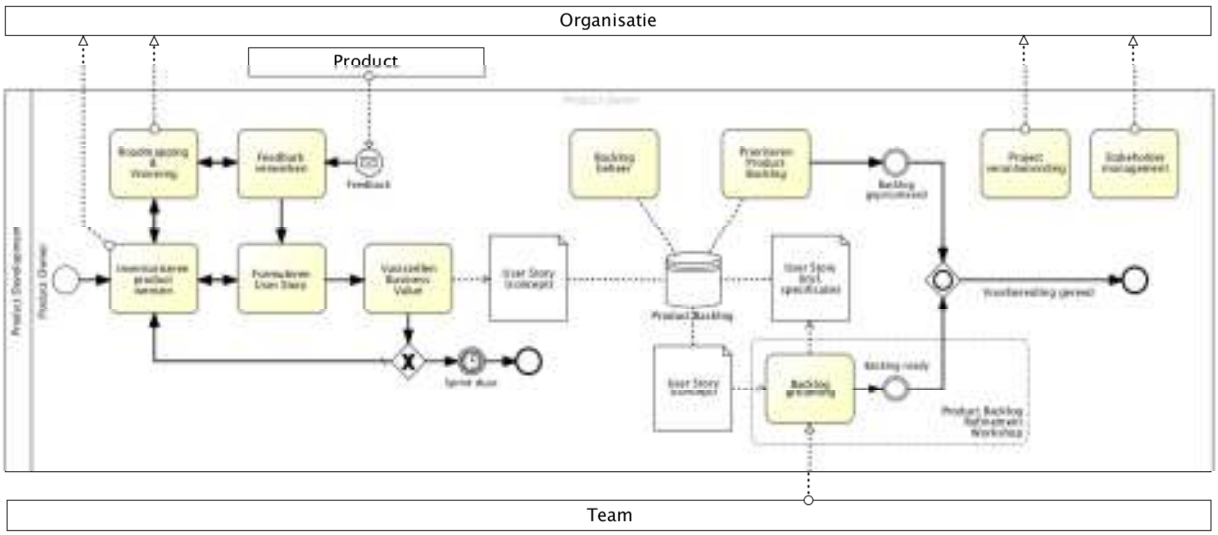
Bij de start van een project wordt vaak de eerste sprint toegewijd aan voorbereiding, dit wordt dan Sprint 0 genoemd. Het doel van deze iteratie is het gereed maken van alle facetten die nodig zijn voor het efficiënt kunnen starten van het project. Denk hierbij aan technische voorbereiding zoals het opzetten een automatisch build- en releaseproces, vaststellen van code standaard en richtlijnen, inrichting van tools et cetera. Ook vindt op hoog niveau system engineering plaats, zoals het vaststellen van de architectuur. Het wel of niet gebruiken van een Sprint 0 wordt in Scrum open gelaten. Het wordt slechts onderkend als een mogelijkheid indien nodig. Het primaire doel is namelijk om zo snel mogelijk

te starten met bouwen om snel waarde voor de klant te kunnen leveren en feedback te kunnen ontvangen.

Het proces van requirements voorbereiding voor de eerste iteratie behoeft uiteraard wat meer capaciteit, omdat daarin initieel de backlog wordt opgesteld. Afgezien daarvan zijn de voorbereiding activiteiten gelijk aan de voorbereiding in alle daarop volgende iteraties. Dit is dan ook als één deelproces weergegeven.

Het proces wordt geïnitieerd door de Product Owner. Deze begint met het opstellen van een Product Backlog. Er wordt onder andere een roadmap en releaseplanning vastgesteld en doormiddel van het toekennen van prioriteit aan de User Stories op de Product Backlog wordt de mogelijke scope voor de sprint bepaald. Het team bereid zich tevens voor op de sprint door ervoor te zorgen dat de requirements voldoend zijn opgehelderd om direct door het team te kunnen worden opgepakt. Als de Product Backlog voor ongeveer anderhalve sprint voldoende is uitgewerkt en voorzien is van prioriteit, kan gestart worden met de sprintplanning. Direct na de sprintplanning wordt (indien van toepassing) direct weer gestart met de voorbereidingsfase voor een volgende sprint. De voorbereidingsfase loopt daarmee parallel aan de ontwikkelfase.

De activiteiten die voor de Product Owner worden beschreven kunnen als volgt in kaart worden gebracht:



Figuur 20 Scrum procesmodel - Sprintvoorbereiding (Product Owner)

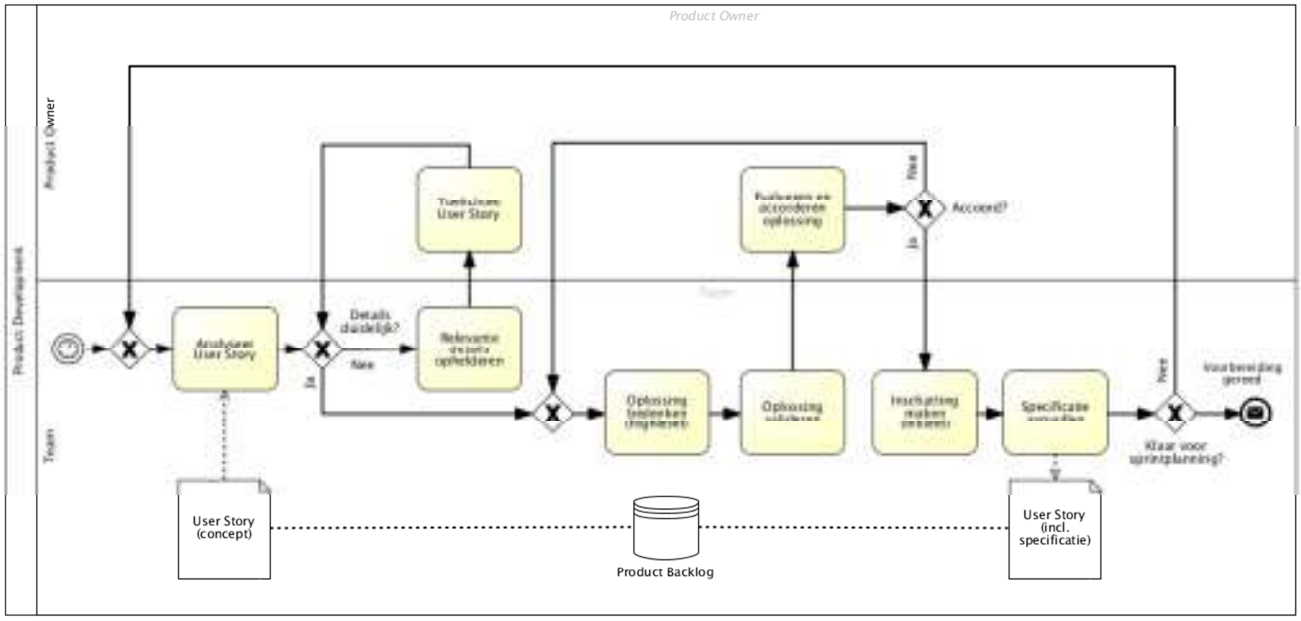
Duidelijk is de centrale rol die de Product Owner heeft in het requirements proces. Zowel elicitatie, specificatie als validatie processen worden aan de Product Owner toebedeeld. Het team richt zich met name op analyse en design.

4.7.1.1 Backlog Grooming

Om ervoor te zorgen dat de productbacklog zo goed mogelijk up-to-date blijft en voor sprintplanning voldoende is uitgewerkt, beschrijft Scrum een proces genaamd *Backlog Grooming*. Hierbij loopt het team in een gezamenlijke sessie (*Product Backlog refinement workshop*), met de Product Owner door het bovenste deel van de backlog heen. Hierbij worden requirements doorgesproken, opgehelderd en worden mogelijke oplossingen afgestemd. Aan de hand hiervan wordt de User Story aangevuld / bijgewerkt en voorzien va een initiële (of bijgewerkte) inschatting.

Deze activiteit borgt de kwaliteit van de productbacklog en representeert de sprintvoorbereiding van het team. Hoe vaak of wanneer backlog grooming plaatsvindt wordt aan het team overgelaten. Aangezien de voorbereidingswerkzaamheden parallel aan de sprint worden uitgevoerd, adviseert Scrum zo'n vijf tot tien procent van de totale sprint capaciteit hiervoor te reserveren.

Het proces van Backlog Grooming kan als volgt worden gemodelleerd:



Figuur 21 Scrum procesmodel – Voorbereiding - Product Backlog Refinement (grooming)

4.7.2 Planning

Als zowel de sprintvoorbereiding gereed is en de product backlog van prioriteit is voorzien, kan gestart worden met de planningssessie. Deze bestaat uit twee delen.

4.7.2.1 Planningssessie deel 1

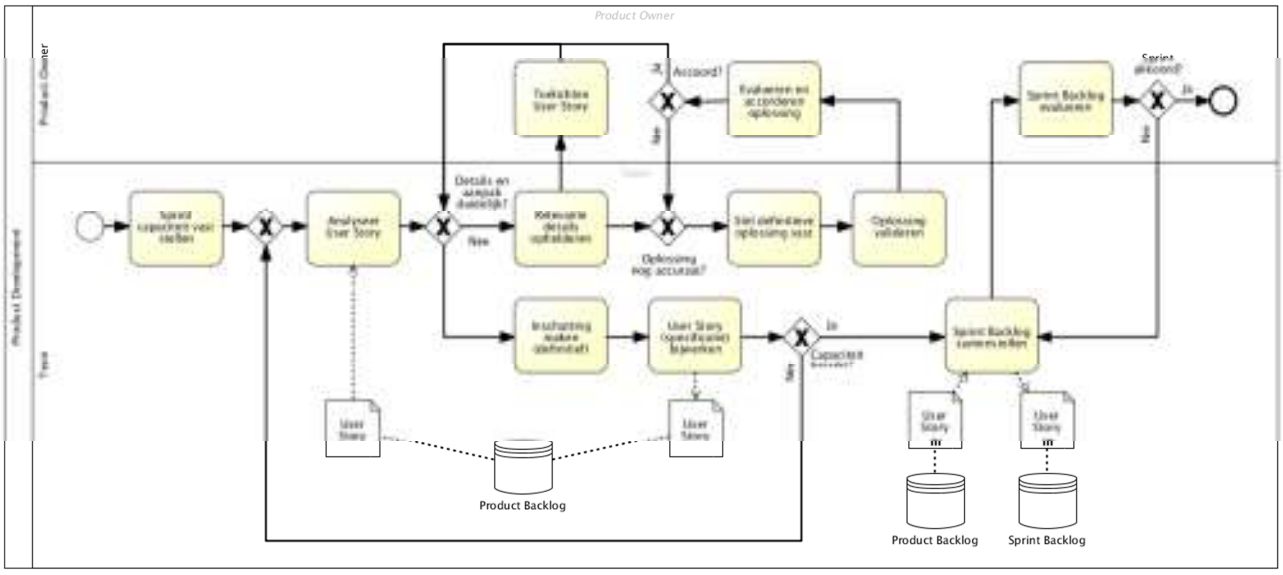
Bij het eerste deel van de sprintplanning is zowel de Product Owner, als het voltallige team en de Scrum master aanwezig. Bij aanvang stelt het team op basis van voorgaande resultaten en beschikbare capaciteit de doelen met betrekking tot de omvang van de sprint vast. Vervolgens worden de User Stories één voor één toegelicht en met het team besproken. De Product Owner geeft waar nodig een

aanvullende toelichting en maakt besluiten met betrekking tot de functionaliteit. Als de wensen en aanpak voor iedereen duidelijk is kan een inschatting worden gemaakt.

Planning Poker

Het inschatten van User Stories gebeurt doormiddel van de *Planning Poker* game. Het team schat de omvang van de requirement in door gezamenlijk een waarde hieraan toe te kennen uit de vaste reeks getallen {0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 100} welke op speelkaarten zijn afgedrukt. Deze getallen worden *story points* of *velocity points* genoemd en representeren een fictieve waarde. De waarde wordt door de teamleden bepaald aan de hand van relatieve omvang ten opzichte van eerder ingeschatte requirements. Hierbij wordt vaak een referentie user story gekozen met de omvang van ongeveer drie story points. Planning Poker maakt het mogelijk om snel en efficiënt een globale inschatting te krijgen, zonder dat alle exacte details in overweging genomen moeten worden.

Na het bespreken van de user story kiest ieder teamlid afzonderlijk een kaart waarvan de waarde volgens hem het meest overeenkomt met de omvang van de requirement. Vervolgens toont iedereen gelijktijdig zijn kaart. Eventuele verschillen worden besproken en er wordt ofwel opnieuw gepokerd, ofwel gezamenlijk een uiteindelijke waarde bepaald. Alle nieuwe, relevante informatie met betrekking tot de user story wordt vervolgens aan de documentatie toegevoegd. Als de ter doel gestelde omvang in story points is bereikt, wordt de voorlopige sprint backlog samengesteld en geëvalueerd en volgt deel twee van de planningsmeeting.

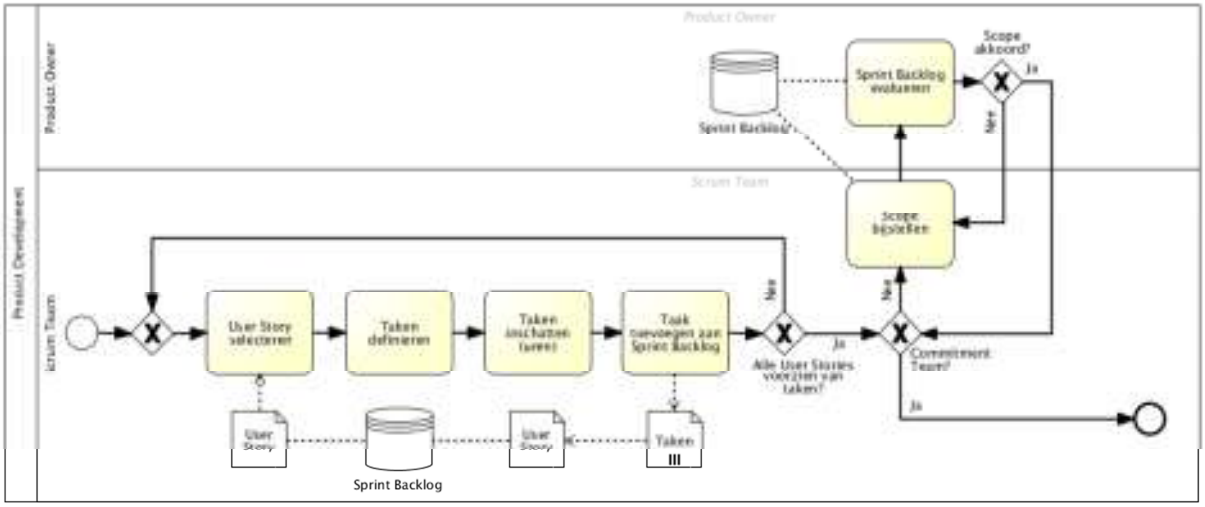


Figuur 22 Scrum procesmodel - Planning deel 1

4.7.2.2 Planningssessie deel 2

In de tweede fase is het niet noodzakelijk dat de Product Owner aanwezig is. Deze is vaak druk en moet niet onnodig worden opgehouden. Het team gaat aan de slag met de geselecteerde User Stories en specificeert voor iedere user story de concrete taken die nodig zijn om de functionaliteit te implementeren. Zo worden verschillende ontwikkelactiviteiten als taak opgenomen, maar ook inricht,

test- en integratietaken. Alle taken worden voorzien van een initiële urenschatting. Deze kan gedurende het proces nog wijzigen, maar geeft al een vroegtijdig inzicht in de omvang van de werkzaamheden. Als alle User Stories zijn ingeschat evalueert het team of het totaal aan User Stories en taken haalbaar is om binnen de gestelde tijd op te leveren. Indien nodig wordt de scope in overleg met de Product Owner bijgesteld. Uiteindelijk committeert het team zich aan de voorgestelde scope en kan met de ontwikkeling worden gestart.



Figuur 23 Scrum procesmodel - Planning deel 2

4.7.3 Uitvoering

De duur van een sprint kan variëren van één tot vier weken. Afhankelijk van de benodigde flexibiliteit, risico's of behoefte aan terugkoppeling tijdens de ontwikkelfase. Gedurende de sprint worden alle taken uitgevoerd die nodig zijn om de User Stories te implementeren en volledig op te leveren. User Stories worden binnen een Sprint op volgorde van prioriteit opgepakt. De volgorde kan worden beïnvloed door onderlinge afhankelijkheid. Het team is hierin leidend.

Gedurende de uitvoeringsfase van de Sprint bespreekt het Team dagelijks de voortgang en eventuele problemen of afhankelijkheden tijdens de *Daily Scrum* meeting.

4.7.3.1 *Daily Scrum*

De *Daily Scrum* (ook wel D*aily Standup* genoemd) is een korte team meeting van maximaal vijftien minuten waarin het team de voortgang en status van lopende zaken bespreekt. Om een actieve houding te stimuleren (en lange discussie te voorkomen) wordt de meeting niet zittend, maar staande gehouden. Er wordt een ronde gemaakt en ieder teamlid vertelt wat hij sinds de vorige *standup* heeft kunnen bijdragen aan de taken en doelstellingen van het team en wat hij die dag denkt bij te dragen. Een belangrijk aspect van de *Daily Scrum* is ook het bespreken van blokkades. Alle factoren die het team weerhouden van voortgang worden met hoge prioriteit opgepakt (deze worden als impediment beschouwd). Indien een blokkade wordt vastgesteld wordt tijdens de *Daily Scrum* direct actie ondernomen. Hierbij kunnen indien nodig andere activiteiten worden stilgelegd tot dat de blokkade is opgeheven.

Het team werkt nauw samen en helpt elkaar waar nodig om een zo goed mogelijk einddoel te bereiken. De status van alle activiteiten wordt bijgehouden en inzichtelijk gemaakt op het Scrum bord.

4.7.3.2 Escalatie

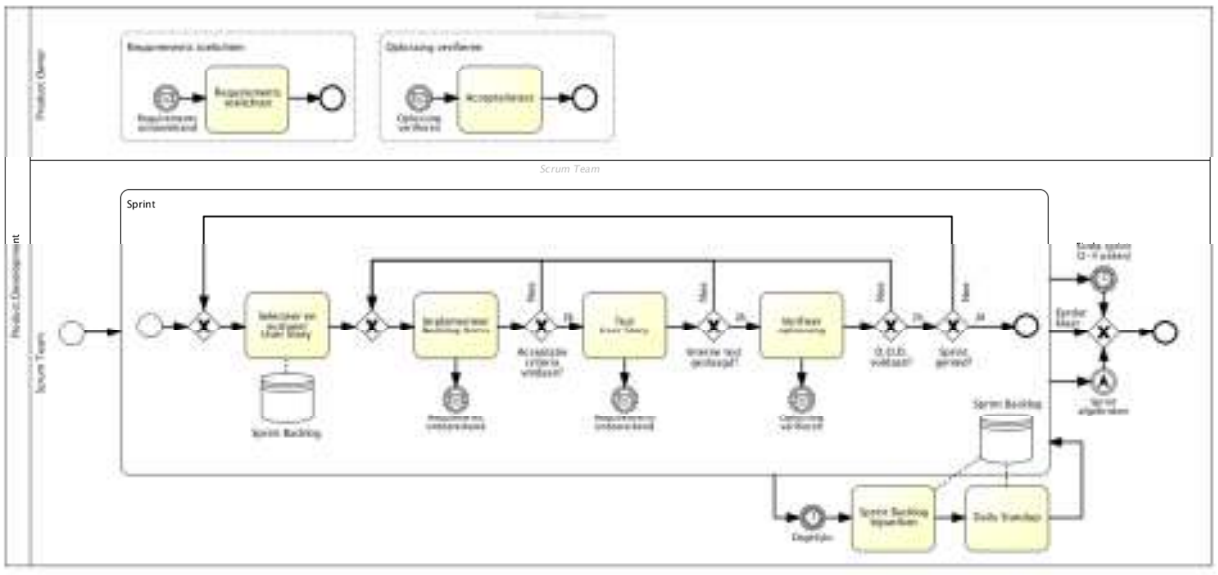
Indien zich gedurende het proces problemen voordoen, of het scope niet meer haalbaar lijkt, wordt de volgende aanpak gehanteerd:

Allereerst zal het team zelf een oplossing proberen te vinden voor het probleem. Hierbij worden alle benodigde middelen binnen het team ingezet. Werkzaamheden kunnen worden stilgelegd en de focus van het team wordt bijgesteld om snel tot een oplossing te kunnen komen.

Slaagt het team er niet in het probleem zelf op te lossen, dan kan zowel horizontaal als verticaal geëscaleerd worden. Zo kan bijvoorbeeld expertise van buiten het team worden bijgeschakeld of om hiërarchische sturing worden verzocht.

Als ook dit het probleem niet oplost kan de scope van de sprint worden bijgesteld. Het is van belang dat het doel van de sprint altijd haalbaar is. Een te hoge druk in het team werkt averechts op de effectiviteit. Het team bepaalt daarom altijd zelf wat het denkt te kunnen leveren binnen de gestelde termijn. In overleg met de Product Owner worden requirements uit de sprint gehaald zodat de sprint alsnog succesvol kan worden afgerond. Een sprint wordt nooit verlengd.

Indien de omstandigheden van dusdanige aard zijn dat het doorzetten van de sprint geen waarde meer toevoegt of dermate chaotisch is, kan de sprint worden afgebroken. Het team stopt dan direct met de werkzaamheden en bereid zich voor op een nieuwe sprint.



Figuur 24 Scrum procesmodel - Uitvoering

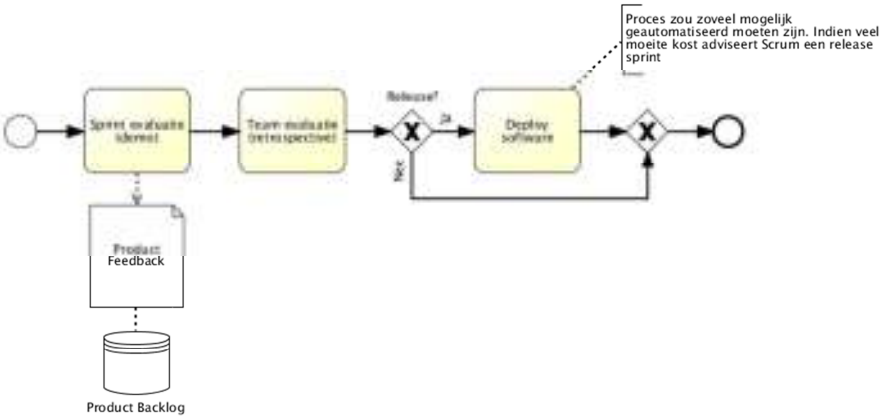
4.7.3.3 DONE

Vanzelfsprekend is het streven om alle geselecteerde User Stories volledig af te ronden voor het einde van een sprint. Toch is dit in de praktijk niet altijd haalbaar. Zeker wanneer het schattingsproces onvoldoende accuraat is gebleken of wanneer andere (onvoorziene) factoren de voortgang hebben vertraagd. Het is binnen Scrum cruciaal dat features waaraan gewerkt wordt, ook volledig afgerond worden binnen een sprint. Onder volledig afgerond wordt niet alleen verstaan dat de code compleet is, maar ook dat de feature volledig getest is en er geen openstaande punten zijn. De feature moet helemaal klaar zijn om opgeleverd te kunnen worden. Het team stelt daartoe een lijst met criteria vast waaraan een user story in het algemeen moet voldoen om als "done" te worden aangemerkt. Dit wordt de "*Definition of Done*" (D.o.D) genoemd. Zie bijlage zeven voor een voorbeeld D.o.D. Een user story die aan het einde van de sprint niet of niet volledig voldoet aan de D.o.D. wordt niet opgeleverd en wordt terug op de Product Backlog geplaatst zodat deze in een volgende sprint opnieuw ingepland kan worden.

4.7.4 Afronding

4.7.4.1 Demo

Na afloop van de sprint wordt het opgeleverde product met de stakeholders geëvalueerd. Hiertoe wordt een demosessie georganiseerd waar alle geïnteresseerden welkom zijn. Het team licht de nieuw ontwikkelde functionaliteit toe en er is ruimte voor feedback. De demosessie is een uitstekende manier om gebruikers en andere stakeholders te betrekken bij de ontwikkeling en ze al in een vroeg stadium feeling te laten krijgen met het product.



Figuur 25 Scrum procesmodel - Oplevering

4.7.4.2 Retrospective

Aan het einde van de sprint wordt een evaluatiesessie gehouden, de *sprint retrospective* genaamd. Tijdens deze sessie, waarbij alleen het team en Scrum master aanwezig is, evalueert het team de afgelopen sprint. Zo wordt besproken wat er goed ging, maar met name ook wat er beter zou kunnen (of moeten) om het proces beter te laten verlopen. De retrospective is een open sessie en het team moet gestimuleerd worden om alles bespreekbaar te maken en zelf na te denken over mogelijke

verbeterpunten of oplossingen. Het team stelt vervolgens concrete punten vast waar de volgende sprint aan gewerkt gaat worden.

De continue evaluatie zorgt ervoor dat het team zich voortdurend verbetert en efficiënter kan werken. Dit levert uiteindelijk meer *customer value* op, maar ook prettigere werkomstandigheden voor het team.

Aan het einde van de sprint kan ervoor worden gekozen om de software daadwerkelijk in productie te nemen (release), of om deze stap over te slaan en op een ander tijdstip te releasen. Hierna is het proces voltooid en wordt eventueel een volgende iteratie gestart.

Scrum beschrijft dat voldoende aandacht moet worden besteed aan het snel en efficiënt kunnen doorlopen van het releaseproces. Er wordt gepleit voor het zoveel mogelijk automatiseren van het deploy- en testproces zodat efficiënt continue integratie kan plaatsvinden (Continious Integration principe uit XP). Wanneer het releaseproces veel handmatige handelingen vereist zou gekozen moeten worden voor het uitvoeren van een Release Sprint. Uiteraard wordt ernaar gestreefd om zo snel mogelijk nieuwe waarde voor de Product Owner te creëren. Een Release Sprint zou daarom vermeden moeten worden.

4.7.4.3 Velocity

De performance van het team, ook wel *team velocity* genoemd wordt na afloop van de sprint bepaald. De velocity wordt bepaald door de som van het aantal story points van alle daadwerkelijk opgeleverde User Stories.

4.8 Scrum of Scrums

Scrum is een methode die zich gemakkelijk laat schalen. Er wordt gewerkt met relatief kleine teams waardoor de beschikbare velocity ontoereikend kan zijn. De capaciteit kan in dit geval gemakkelijk worden opgeschaald door meerdere teams parallel aan een product te laten werken en de backlog tussen de teams te verdelen.

Wanneer met meerdere teams gewerkt wordt, is de *Scrum of Scrums* een techniek waarmee communicatie en synchronisatie tussen teams bewerkstelligd kan worden. De Scrum of Scrums is korte een meeting waarbij van ieder team één afgevaardigde aanwezig is. Tijdens dit overleg, wat veel weg heeft van de *Daily Scrum* meeting, wordt de voortgang van ieder team besproken alsmede eventuele blokkades en onderlinge afhankelijkheden.

Afgezien van operationele afstemming tussen teams onderling, kan de Scrum of Scrums ook worden gebruikt voor evaluatie (retrospective) over teams heen, waarmee trends of gezamenlijke focuspunten met betrekking tot het proces kunnen worden vastgesteld.

4.9 Scrum en requirements

Binnen Scrum worden requirements verondersteld aanwezig te zijn op het juiste niveau van specificatie voordat aan de planningssessie kan worden begonnen. Als richtlijn wordt aangehouden dat de hoeveelheid uitgewerkte requirements ongeveer anderhalve sprintcapaciteit omvat alvorens met de planningssessie te starten. (Sutherland & Schwaber, The Scrum Papers: Nuts, Bolts, and Origins of an Agile Framework, 2011)

Het begrip requirements wordt in Scrum overigens vermeden. Scrum praat liever over “enabling specification” of “mini-enabling specification”. Wat neer komt op een korte, adequate beschrijving van de wens. Binnen de definitie van deze thesis weldegelijk een (of meerdere) requirement(s). De specificatie kan neer komen op een simpele alinea, of een document van drie tot vijf pagina’s. Uit praktische bevindingen blijkt dat zelfs de grootste features binnen vijf pagina’s beschreven zouden moeten kunnen worden. Doorgaans is echter vaak geen lange beschrijving nodig. (Scrum Inc, 2012). De oplossing kan daarentegen weldegelijk complex en omvangrijk zijn. De omvang van een User Story (in story points) wordt binnen Scrum beperkt door grote brokken functionaliteit op te splitsen in meerdere User Stories, die gemakkelijk in een sprint gerealiseerd kunnen worden.

Hoe het requirements proces precies vorm gegeven wordt valt buiten de scope van de methode en wordt grotendeels ter invulling aan de organisatie overgelaten. Scrum beschrijft een belangrijke rol voor de Product Owner voor wat betreft het ophelderen en communiceren van requirements zowel tijdens de planning sessie, op de Product Backlog, als tijdens development. Er is ook uitgebreide aandacht voor het managen van de *flow* van nieuwe requirements richting het ontwikkelteam. Hierbij staan de productbacklog en sprintbacklog centraal en worden requirements in de vorm van User Stories beschreven. Ook hier is weer een grote rol beschreven voor de Product Owner. Naast het opstellen van User Stories is deze ook verantwoordelijk voor het managen en prioriteren van de Product Backlog en dient er voor te zorgen dat deze alle actuele wensen voor het product representeert. Het is de verantwoordelijkheid van het team om er voor te zorgen dat de User Stories voldoende zijn opgehelderd om met de implementatie te kunnen starten. De rol van de requirements engineer (indien aanwezig in het proces) vervaagt hierdoor. Typische requirements engineering taken worden zowel bij de Product Owner (meestal de klant of opdrachtgever met weinig kennis op dit vakgebied), als bij het team als geheel belegd. Het risico bestaat dat de requirements daardoor niet goed worden vertaald naar een implementatie, of dat het team tijdens de sprint veel tijd aan het ophelderen van vraagstukken moet spenderen.

Door juist te focussen op directe communicatie tussen het team en de opdrachtgever wordt getracht de formalisatieslag te minimaliseren. Er kan op deze manier erg flexibel worden gewerkt. Nadrukkelijke voorwaarde hiervoor is wel dat de Product Owner voldoende betrokken moet zijn en deze veelzijdige rol op een juiste manier invult. Wat betreft de benadering en verantwoordelijkheden van het compleet krijgen van requirements, kiest Scrum geen duidelijke lijn. Deze liggen bij het team als geheel. Het gaat daarbij uit van Multi-Skilled developers die een brede rol kunnen vervullen in het proces.

De wensen worden door de Product Owner in de vorm van User Stories beschreven. Het initiële requirements proces wordt daarbij door de Product Owner uitgevoerd. Tijdens de Product Backlog Refinement workshop evalueert het team de User Stories en verzorgt een meer diepgaande analyse op de requirements. Er ontstaat daarbij een specificatie die zowel functioneel als technisch voldoende duidelijk moet zijn om binnen een sprint te kunnen opleveren. De mate van detail kan variëren en is afhankelijk van inzichten en informatiebehoeften van het team, de complexiteit en andere factoren. Scrum beschrijft dat alleen het strikt noodzakelijke moet worden vastgelegd. Dit houdt in dat niet per definitie alle mogelijke details hoeven te worden uitgewerkt, maar een beknopte heldere toelichting zal voldoen.

Doordat requirements processen tijdens een sprint worden uitgevoerd en niet als aparte fase voorafgaand aan systeemontwikkeling, worden ze als integrale activiteit gezien in het proces. Er moet daarom capaciteit in het team beschikbaar zijn om voldoende “grooming” werkzaamheden te kunnen verrichten.

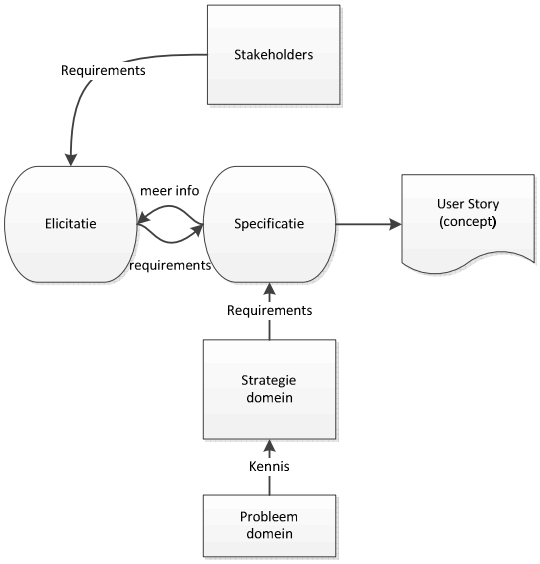
Hoewel de rol van requirements engineer niet wordt beschreven binnen Scrum, wordt er weldegelijk gesproken over multidisciplinaire teams waarin alle benodigde specialismes zijn vertegenwoordigd. Het ligt voor de hand dat hierin ook een rol voor analyse en specificatie, dan wel requirements engineering wordt ingevuld als daar behoefte aan is. Er is daardoor weldegelijk ruimte voor een meer gedegen requirements proces.

Het iteratieve karakter van sprints levert duidelijk beperkingen als het gaat om voorbereiding van specificaties ten opzichte van de traditionele benadering (waterval). Dit geldt zowel voor de requirements specificatie, als de technische specificatie (design). Het doel van Scrum is om zo snel mogelijk te starten met bouwen om snel waarde voor de klant te kunnen leveren en feedback te kunnen ontvangen. Alle voorbereidende handelingen moeten in een periode van twee tot vier weken (sprintduur) zo goed mogelijk zijn afgerond. Daarbij wordt telkens gefocust op de, op dat moment, belangrijkste wensen die zich bovenaan de Product Backlog bevinden.

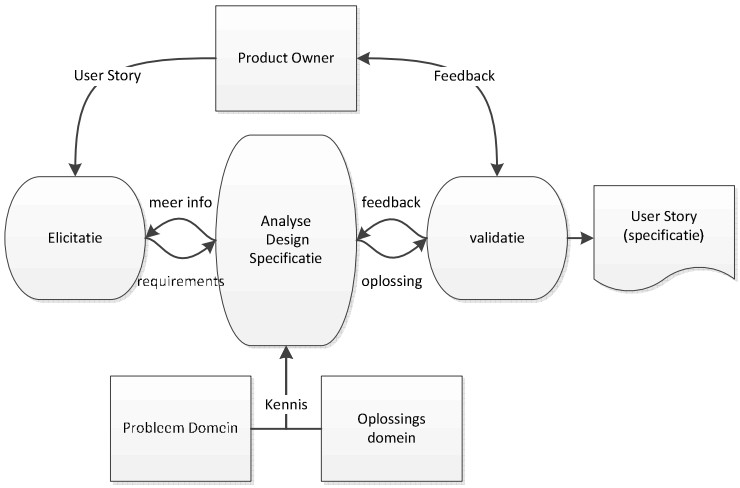
Een juiste voorbereiding is dus essentieel om de voortgang van het team niet te remmen. Het team zou direct na de afronding van een sprint moeten kunnen starten met de planning van een volgende sprint. Het afrondings- en sprintplanningsproces worden daarom vaak op dezelfde dag (direct na elkaar doorlopen).

Het iteratieve requirements model uit Figuur 8, waarin de requirements engineer een hoofdrol speelde in het requirements proces en elicitatie bij stakeholders, is binnen Scrum vervangen door een tweeledig model waarin zowel de Product Owner als het team een rol hebben in het vaststellen van de requirements. Zie Figuur 26 en Figuur 27.

Hierbij worden de zo genaamde User Requirements door de Product Owner opgesteld. De technische analyse die leidt tot System Requirements en eventuele Design Specification wordt door het team verzorgt.



Figuur 26 Scrum requirements model – Sprintvoorbereiding - Product Owner



Figuur 27 Scrum requirements model – Sprintvoorbereiding / planning - Team

5 Lean Software Development

In dit hoofdstuk wordt een toelichting gegeven op het Lean gedachtegoed binnen de context van softwareontwikkeling en de relatie tot requirements.

5.1 Inleiding

Lean kan worden beschouwd als een processtrategie om optimale efficiëntie en optimalisatie in productieprocessen te bereiken. Het kent vele specifieke principes, practices en tools en vormt een omvangrijk geheel. In dit hoofdstuk zijn de algemene principes en waarden uiteengezet die centraal staan in de methode. Vervolgens is gekeken in hoeverre deze van toepassing zijn en kunnen bijdragen aan de requirements processen binnen Scrum (deelvraag 2).

5.2 Algemeen

Lean is een filosofie en manier van werken die sinds de jaren tachtig veel wordt gebruikt voor het organiseren en verbeteren van operationele bedrijfs- en productieprocessen. Lean vindt zijn oorsprong in de Japanse automotive industrie (Ruffa, 2008) en is gebaseerd op het succesvolle model van Toyota (TPS, Toyota Production System) waarmee een grote voorsprong op de concurrentie is behaald door een hoge mate van procesoptimalisatie. (Womack & Jones, 1996) De concepten en tools van het Lean productieproces en de benaming als zodanig zijn echter pas vele jaren later voor het eerst beschreven door Womack en Jones in het boek "The machine that changed the World: The Story of Lean Production" (1990).

Lean richt zich op het optimaliseren van het proces door de doorlooptijden te verkorten en efficiëntie te vergroten. Lean heeft zijn oorsprong in het productontwikkelproces binnen industriële en logistieke toepassingen. Toch worden veel van deze principes en methoden ook steeds vaker in andere vakgebieden toegepast. Zo ook binnen de software branche. Het proces van Softwareontwikkeling kent grote verschillen ten opzichte van (fabrieks-)productieprocessen. In plaats van het creëren van een tastbaar product, vaak in serie of massa productie, wordt een uniek intellectueel eigendom gecreëerd wat in grote mate afhankelijk is van de kwaliteiten, creativiteit en efficiëntie van het ontwikkelteam. Iedere klantwens is uniek en er worden vaak vele klantwensen tegelijk opgeleverd. Toch is veel van het Lean gedachtegoed ook zeer effectief gebleken in het proces van softwareontwikkeling.

Lean Software Development beschrijft de toepassing van Lean principes binnen de scope van softwareontwikkeling. Lean Software Development (of Lean Development) is niet zozeer een ontwikkelmethode, maar introduceert een meer holistische kijk op het ontwikkelproces door te focussen op de flow van waarde binnen de organisatie. Dit in tegenstelling tot de meeste Agile methoden die zich meer richten op het proces van de totstandkoming van software (het *engineering* proces).

De volgende cyclus staat binnen het Lean gedachtegoed centraal: (Womack & Jones, 1996)

- Identificeer de waarde (*value*) die in het proces gecreëerd wordt per productgroep vanuit de visie van de klant.

- Breng de waarde stroom (*value stream*) op organisatieniveau in kaart door het creëren van een zogenaamde *value stream map* waarin alle stappen in het proces worden opgenomen inclusief doorlooptijd en productieaantallen.
- Creëer een flow. Identificeer de handelingen die geen waarde toevoegen aan het product (*waste*) en elimineer deze. Door de overgebleven handelingen te stroomlijnen kan een optimale flow van waarde worden gecreëerd.
- Pull: De hoeveelheid voorraad (of vooruitwerken) moet worden beperkt. De flow moet gestuurd zijn door de vraag van de klant. Werk wat zich opstapelt draagt niet bij aan de waarde, beperkt de flexibiliteit en wordt als *waste* beschouwd. Door het '*work in progress*' te beperken wordt overzicht en focus gecreëerd en een *pull* mechanisme bewerkstelligd.
- Perfect: Door het voortdurend optimaliseren van het proces wordt gestreefd naar een ideale situatie waarbij waarde gecreëerd wordt zonder *waste*.

Deze principes worden hieronder toegelicht binnen de context van softwareontwikkeling.

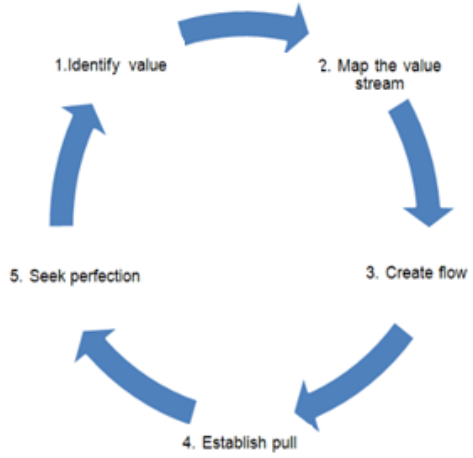
Figuur 28 geeft deze continue cyclus weer. Deze principes worden hieronder toegelicht binnen de context van softwareontwikkeling.

5.3 Principes

Bob Charette beschreef verschillende principes van Lean Software Development. Een uitgebreidere visie op het Lean gedachtegoed binnen softwareontwikkeling is gegeven in het werk van Mary en Tom Poppendieck. Zij benoemde onder andere de volgende principes van Lean Development (Poppendieck & Poppendieck, 2003):

1. Het elimineren van *waste*;
2. Stimuleren van kennisopbouw;
3. Beslis zo laat mogelijk: Just In Time (JIT);
4. Zo snel mogelijk waarde leveren;
5. Versterk teams en respecteer menselijke aspecten;
6. Borgen van kwaliteit;
7. Bekijk de *big picture* en optimaliseer het geheel.

Deze principes worden hieronder toegelicht binnen de context van softwareontwikkeling.



Figuur 28 Elementen van Lean (bron: imakenews.com)

- Elimineren van waste:
  Waste staat voor verspilling en omvat alle handelingen die geen waarde bijdragen aan het product. Centraal staat de waarde die gecreëerd wordt, deze wordt echter beperkt door de waste. Waste kan zowel in het product optreden, als in het proces. In sectie 5.4 worden de verschillende type waste verder toegelicht binnen de context van softwareontwikkeling.

- Stimuleren van kennisopbouw:

Het ontwikkelen van software producten is specialistische aangelegenheid. Het ontwikkelen van een oplossing is geen stap-voor-stap herhaling van een aantal voorspelbare handelingen, maar een creatief proces en een zoektocht in de totstandkoming van het product. Het team wat het product ontwikkelt doorgaat zowel een leertraject met betrekking tot kennis over het product, als tevens hoe het proces daarnaartoe wordt vormgegeven. Wat gaat wel- en niet goed. Er zou dus voldoende aandacht moeten zijn voor kennisopbouw en persoonlijke ontwikkeling van teamleden en evaluaties van processen. Het streven hierin is om te komen tot zogenaamde *high performing* teams met een hoog kennis en volwassenheidsniveau.

- Uitstellen van beslissingen:
  Beslissingen die gedurende het proces worden genomen zijn veelal gebaseerd op kennis en ervaring die met het product is opgedaan. Beslissingen die voor over een lange periode in de toekomst worden genomen zijn onbetrouwbaarder omdat deze voornamelijk gebaseerd zijn op inschattingen of voorspellingen. Door beslissingen zo lang mogelijk uit te stellen en precies op het moment dat het ter zake doet te behandelen (Just in time), worden besluiten op basis van de meest actuele kennis en inzichten genomen. Het risico dat later op beslissingen moet worden teruggekomen (en dus extra werk wordt gecreëerd), wordt hierdoor beperkt.

- Snel waarde leveren:
  Het ontwikkelen van grote ingewikkelde system en oplossingen maakt het proces onnodig complex en omvangrijk waardoor tal van knelpunten ontstaan. Het doel zou moeten zijn om alles simpel, duidelijk en overzichtelijk te houden (*keep it simple-principe*). Door het hele proces op te knippen in kleine stukjes (iteraties) wordt het product incrementeel opgebouwd en kan de focus worden gelegd op een klein, maar belangrijk deel van het product. Er wordt zo na iedere iteratie waarde geleverd aan de klant. De order-to-cash tijd kan hierdoor worden verkort.

- Respecteer mensen:
  Het team wat betrokken is bij de ontwikkeling van softwareproducten is zoals eerder beschreven bij voorkeur een multidisciplinair team van experts op ieder benodigd vakgebied. Deze personen weten als de beste hoe het product zou moeten worden opgebouwd en hoe het proces daartoe het meest efficiënt kan worden ingevuld. Zij streven van naturen naar hoge efficiëntie omdat ze graag in korte tijd een mooi product willen neerzetten waar de klant tevreden mee is. De gedachte binnen Lean Development is dan ook om het team een grote mate van vrijheid te geven en dit naar eigen inzichten te laten inrichten.

- Borgen van kwaliteit:
  De kwaliteit van een software product is erg belangrijk en vertegenwoordigd een deel van de uiteindelijke waarde. Kwaliteitsproblemen leiden tot bugs en falen van het product. Dit doet vanzelfsprekend afbreuk aan de waarde van het product. Het introduceert nieuw werk en is dus een bron van verspilling. Doel van het Lean proces is: het zo efficiënt mogelijk leveren van de juiste producten met precies de juiste kwaliteit. Met name dat laatste is interessant, want binnen de individuele specialismes (onder andere requirements en development) is het streven juist vaak naar

een zo hoog mogelijke kwaliteit. Het proces moet dus faciliteren dat de mate van kwaliteit door de opdrachtgever kan worden beïnvloed, doel voor het team is het bereiken van de optimale kwaliteit binnen de gestelde kaders.

- Big picture:
  Optimalisatie zou de gehele flow moeten omvatten. Daarvoor is het nodig om de *big picture* te zien van hoe het product tot stand komt. Door de stroom van waarde in kaart te brengen kunnen knelpunten worden geïdentificeerd en kan optimalisatie worden gericht op precies die plekken waar de flow wordt beperkt. Sub-optimalisatie verslechterd de integratie en overgang van processen. De focus zou daarom moeten liggen op het geheel.

  Het principe van *Jidoka*, zoals genoemd in Lean, kan ook hier worden toegepast. Zodra een blokkade of knelling in de flow wordt geconstateerd, moet het tempo van de andere processen naar beneden worden bijgesteld (of stopgezet) om te voorkomen dat werk zich gaat opstapelen. Vergelijkbaar met de: 'productielijn stop' procedure in fabrieksprocessen. Het heeft geen zin om andere activiteiten door te laten gaan als zich elders in de keten een probleem voordoet. Dit introduceert waste. Kanban is een Lean toepassing waarmee de flow en werkvoorraden in kaart kan worden gebracht en worden gestuurd. Sinds circa 2006 wordt Kanban ook binnen softwareontwikkeling toegepast.

In feite komen al deze principes neer op het leveren van waarde aan de klant. Daarbij worden binnen Lean de volgende kernaspecten centraal gesteld:

1) Waarde (*value*): waarde is het belangrijkste en dient zo efficiënt mogelijk geleverd te worden.

2) Flow: de flow bepaalt in grote mate de waarde die geleverd wordt. Door de aandacht te vestigen op de flow binnen de organisatie kan het proces van waardecreatie worden geoptimaliseerd.

3) Waste: de flow binnen de organisatie wordt beperkt door *waste*. Waste zijn alle handelingen die niet direct bijdragen aan de waarde binnen de flow. Het elimineren van waste zorgt ervoor dat de *flow* toeneemt en uiteindelijke *customer value* geleverd kan worden.

5.4 Waste

Waste zijn alle handelingen die niet of niet direct bijdragen aan de waarde binnen de flow en dus extra ballast creëren. Het elimineren van waste is een belangrijk speerpunt van Lean om efficiëntie in het proces te vergroten. In deze sectie wordt het begrip Waste toegelicht binnen de context van softwareontwikkeling.

> *"Current software projects spend about 40 to 50 percent of their effort on avoidable rework" - B. Boehm, V. R. Basili, "Software Defect Reduction Top 10 List," IEEE Computer, Jan 2001*

Binnen het proces van softwareontwikkeling kunnen de volgende zeven waste factoren worden onderscheiden. (Poppendieck & Poppendieck, 2003) Deze factoren zijn afgeleid van de *seven types of waste,* zoals beschreven door Womack & Jones.

1. *Deels afgerond werk*
   Werk wat niet, of niet geheel, is afgerond kan niet worden opgeleverd en draagt dus niet bij aan de *customer value*. De houdbaarheid van deels afgerond werk is tevens beperkt en introduceert zowel task switching, als overdracht om het werk alsnog af te ronden. Het op een later tijdstip afronden van werkzaamheden is dus minder efficiënt.

2. *Extra processen*
   Processen en procedures die het ontwikkelproces onnodig bureaucratisch maken vertragen het proces.

3. *Extra features*
   Ongebruikte functionaliteit is een grote bron van waste. Uit onderzoek is gebleken dat gemiddeld 64% van de opgeleverde features niet of nauwelijks wordt gebruikt. (Johnson, 2002) Deze features kosten wel geld en onderhoud en kunnen tevens nieuwe (onnodige) defecten introduceren. Ongebruikte functionaliteit kan worden geïntroduceerd door niet goed doordachte, of geformaliseerde klantwensen, of doordat ontwikkelaars buiten de scope van de requirements treden. Ook het opnieuw bedenken of uitwerken van reeds beschikbare oplossingen wordt als waste beschouwd.

4. *Task switching*
   Task switching is de tijd die verloren gaat bij het omschakelen van werkzaamheden. Zeker in het geval van softwareontwikkeling kost het ontwikkelaars de nodige inspanning om na een onderbreking (door andere activiteit) alle details weer helder op het netvlies te krijgen en tools gereed te maken om het werk te continueren. Het combineren van verschillende (simultane) werkzaamheden beperkt aanzienlijk de focus en productiviteit en wordt derhalve als bron van waste beschouwd.

5. *Wachten en vertraging*
   Het wachten op andere mensen, processen of tools is pure waste en stimuleert tevens ‘task switching’.

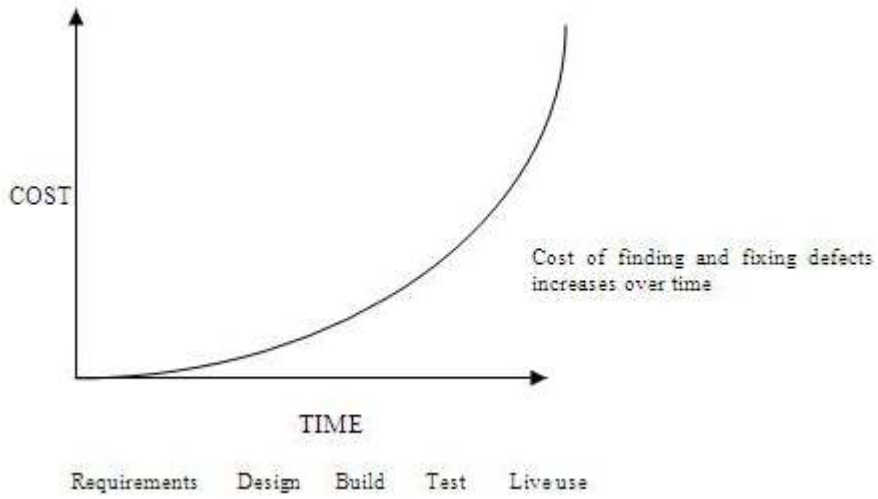
6. *Overdracht*
   Het overdragen van werkzaamheden introduceert fouten en beperkt de mate waarin werknemers zich verantwoordelijk voelen voor de oplossing. Bij overdracht gaat tevens veel informatie verloren. Gemakshalve kan ervanuit worden gegaan dat ongeveer 50% van de informatie bij iedere overdracht verloren gaat. (Poppendieck & Poppendieck, 2006.) Tevens heeft de nieuwe persoon enige tijd nodig om zich alle informatie eigen te maken.

7. *Defecten*
   Defecten (of bugs in software) dragen vanzelfsprekend niet bij aan de waarde van het product of klant tevredenheid. Defecten kunnen de waarde van het product zelfs verminderen indien klanten bijvoorbeeld ontevreden zijn en wegblijven.

Om bugs te kunnen oplossen zijn nieuwe handelingen vereist. Dit introduceert nieuw werk wat voorkomen had kunnen worden. De omvang van de *waste* is daarbij afhankelijk van het moment dat de bug gevonden wordt. Hoe later in het proces, hoe groter de impact. Poppendieck en Poppendieck geven daarbij de volgende formule:

$$Waste = (impact\ of\ defect) \times (time\ defect\ lies\ undetected)$$

Ook vanuit andere vakgebieden (als onder andere testtechnieken) wordt dit probleem onderkend. De onderstaande figuur geeft een indicatie van de stijging van kosten ten opzichte van de fase waarin de bug wordt ontdekt.



Figuur 29 Kosten van defecten t.o.v. detectietijd (bron: http://istqbexamcertification.com)

5.5 Lean en Requirements

Lean richt zich op het optimaliseren van waardecreatie in de gehele cyclus van de totstandkoming van een product. De principes en toepassingen dringen echter door tot diep in de individuele processen om de daadwerkelijke optimalisatie te bereiken. Hoewel het zeer interessant is om de gehele cyclus te beschouwen en te optimaliseren, is omwille van de scope in deze thesis enkel de focus gelegd op aspecten die van invloed kunnen zijn op requirements-gerelateerde activiteiten en de visie die daarop toegepast zou moeten worden. Zoals uit hoofdstuk 2 is gebleken beperken deze activiteiten zich niet tot de voorbereidingsfase, maar komen deels ook terug in andere fasen van het proces.

In de volgende secties bekijken we de kernaspecten van Lean vanuit een requirements perspectief. Vervolgens leiden we daaruit conclusies af ten aanzien van de probleemstelling.

5.5.1 Value

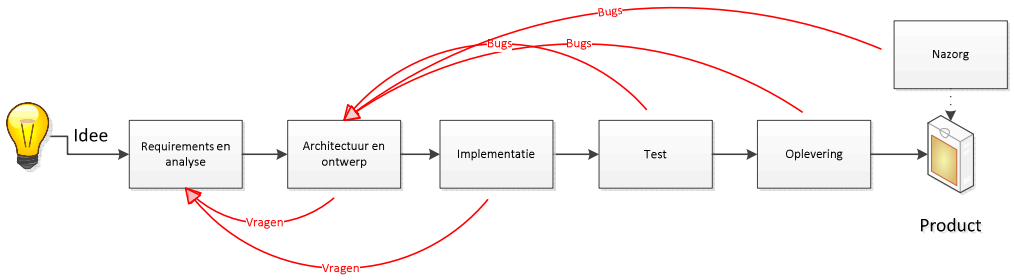
Value is de uiteindelijke waarde die gecreëerd wordt. Binnen software ontwikkeling is dit een software product wat in meer of mindere mate waarde creëert voor de opdrachtgever. Afgezien van omgevingsfactoren wordt de omvang van waarde bepaald door de mate waarin de oplossing aansluit bij de wensen van de klant. Requirements omschrijven deze wensen en zijn in die zin de input voor het proces. Requirements zelf vertegenwoordigen geen waarde, wel helpen ze de waardecreatie door

ervoor te zorgen dat systeemwensen zorgvuldig gecommuniceerd en overgedragen kunnen worden. Ze **borgen** daarmee de kwaliteit en waarde. Binnen Scrum wordt getracht de overdracht te beperken door korte communicatielijnen tussen het team en de Product Owner Door veelvuldige face-to-face communicatie is minder specificatie en documentatie nodig om tot de juiste oplossing te kunnen komen. De rol van requirements specificaties is dan minder van belang en zou zelfs als waste kunnen worden aangemerkt. Deze sterk informele benadering van requirements kan zeer succesvol werken in een klein team met een zeer betrokken opdrachtgever, echter wanneer de complexiteit en omvang van een product toeneemt en daarbij ook het aantal deelnemers in het proces, neemt automatisch ook de communicatie en overdracht van informatie toe. Requirements specificaties worden dan weer opnieuw belangrijk om zo weinig mogelijk informatie verloren te laten gaan.

Een optimale efficiëntie bereiken in het requirements proces is dus een balans die gevonden moet worden in de mate van formaliteit en diepgang waarmee requirements worden gehanteerd.

5.5.2 Flow

De flow is het proces waarin alle activiteiten worden doorlopen. In sectie 2.1 zijn de activiteiten beschreven die worden doorlopen bij het ontwikkelen van een software product. Daarbij zijn er tal van terugkoppelingen die kunnen plaatsvinden indien bijvoorbeeld requirements niet toereikend zijn of opnieuw geëvalueerd moeten worden, maar ook bugs (fouten) vertegenwoordigen een belangrijk deel van de inefficiëntie. Deze terugkoppelingen zorgen ervoor dat een deel van de flow opnieuw doorlopen moet worden. Figuur 30 geeft dit in de *value stream* weer.



Figuur 30 Value Stream inclusief terugkoppeling

Binnen Lean wordt gebruikgemaakt van een techniek genaamd *value stream mapping* om de flow van waarde en waarde creërende activiteiten in kaar te brengen. Daarbij worden tevens aantallen, percentages en tijden vermeld (zoals *cycle time* en *value adding time)*. Het opstellen en analyseren van een complete value stream map gaat buiten de scope van deze thesis maar kan weldegelijk een goed beeld geven van de plaatsen waar waste optreed en de omvang daarvan. Een voorbeeld van een *value stream map* voor een software organisatie is te vinden in bijlage twee. We zullen we ons beperken op de flow binnen de requirements processen en de terugkoppelingen die daarvan afhankelijk zijn.

Om een optimale flow te bereiken dienen volgens Lean de terugkoppelingen te worden geminimaliseerd en de individuele processen geoptimaliseerd. Daarbij is het met name van belang hoe processen op

elkaar aansluiten. Zeker wanneer Agile methoden worden gebruikt met een afwijkende projectfasering waardoor activiteiten in verschillende fasen overlappen.

Een belangrijk principe van Lean is het zo laat mogelijk nemen van beslissingen. Met betrekking tot requirements betekent dit dat details pas op het laatste moment worden opgehelderd wanneer de informatie daadwerkelijk nodig is. Dit wordt ook wel "*Lazy requirements engineering*" genoemd of "*Just-in-time requirements".* Deze aanpak vereist dat het requirements proces snel en efficiënt doorlopen kan worden. Als deze aansluiting ontbreekt, worden ontwikkelaars geconfronteerd met onduidelijke systeemwensen en treden allerlei vormen van waste op die de flow stagneren.

Een belangrijke eigenschap van iteratieve methoden is dat ook slechts een beperkte voorbereidingstijd beschikbaar is omdat iteraties elkaar snel opvolgen. Het is daarom van belang dat binnen de beperkte voorbereidingstijd precies de juiste requirements worden uitgewerkt en gereed gemaakt voor implementatie.

Scrum besteed veel aandacht aan het managen en inzichtelijk maken van de flow, maar richt zich daarbij voornamelijk op de implementatie, test en oplevering. Het begrip requirements wordt luchtig toegepast en ook de voorbereidingsfase wordt slechts highlevel beschreven als het gaat om requirements. De flow wordt inzichtelijk gemaakt middels het Scrum bord. De totale voortgang en het tempo wordt doormiddel van de *burn down chart* inzichtelijk gemaakt. De status en voortgang in de voorbereidingsfase wordt echter nergens geadresseerd. De voorbereidingsfase waarin het merendeel van de requirements engineering activiteiten plaatsvinden, is de input van de engineering processen en zou derhalve meer aandacht moeten krijgen om een juiste aansluiting te vinden en te voorkomen dat de flow stagneert (terugkoppeling plaatsvindt) in daarop volgende processen.

5.5.3 Waste

Waste wordt getypeerd door alle handelingen die geen waarde toevoegen aan de flow. Binnen het proces van requirements engineering kunnen de volgende bronnen van waste worden vastgesteld:

5.5.3.1 Overspecificatie:

Zoals eerder beschreven vormt de requirements specificatie voor ontwikkelaars de basis voor alle uit te voren ontwikkelwerkzaamheden. Ontwikkelaars moeten op basis van de beschikbare specificatie een duidelijk beeld krijgen van hetgeen precies ontwikkeld moet worden en binnen welke kaders deze oplossing gevonden moet worden. Het toevoegen van niet relevante informatie doet de omvang van de specificatie toenemen zonder dat dit waarde aan het product toevoegt. Het kost deelnemers in het proces dan meer moeite om de essentie vast te stellen. Het risico ontstaat dan dat de specificatie niet goed gelezen wordt en belangrijke details over het hoofd worden gezien.

Uiteraard is alle energie die gestoken wordt in het specificeren van onnodige details verloren moeite en een *waste* of time.

5.5.3.2 Onderspecificatie:

Wanneer relevante details ontbreken in de specificatie, ontstaat onduidelijkheid. Het team moet dan zelf moeite steken in het ophelderen van details. Dit introduceert extra communicatie, task switching en mogelijke wachttijd. Ook ontstaat het risico dat ontwikkelaars zelf aannames doen, die op hun beurt weer tot andere vormen van waste kunnen leiden.

Voordat aan de ontwikkeling wordt gestart, is de benodigde inspanning door ontwikkelaars ingeschat op basis van de tot dan toe beschikbare informatie. Wanneer daar belangrijke details ontbreken zal deze inschatting onbetrouwbaar zijn. Kleine details kunnen namelijk grote impact hebben in de software wereld. Aangezien deze inschatting direct gebruikt wordt om de planning en scope vast te stellen, kunnen hier problemen optreden. Aangezien binnen Agile projecten de factor tijd (beschikbare ontwikkeltijd) en daarbij ook de beschikbare resources (teamleden) vast staan, betekent dit automatisch een variatie in de scope.

5.5.3.3 Fouten in de specificatie:

Aangezien de specificatie door ontwikkelaars als uitgangspunt wordt gebruikt voor het bedenken en uitwerken van de oplossing (en de specificatie door het team vaak als heilig wordt beschouwd) leiden fouten in de specificatie automatisch tot fouten in het product.

Deze fouten kunnen ontstaan door interpretatiefouten in de analyse fase (wanneer de wensen van de opdrachtgever niet goed begrepen worden), of door een onjuiste specificatie (wanneer de wensen wel goed begrepen worden, maar niet goed worden verwoord).

Afgezien van de waste factoren die binnen het requirements proces zelf kunnen optreden zijn er ook verschillende bronnen die in andere processen optreden als gevolg van een van de bovenstaande factoren. Wanneer we kijken naar de bronnen van waste uit sectie 5.4 dan zijn hier met name “3. Extra features”, en “7.Defecten” relevant. Beide vormen van waste kunnen optreden wanneer requirements fouten bevatten of ontoereikend zijn.

5.5.3.4 Voorraad

Ook het opbouwen van voorraden wordt door Lean beschouwd als waste. Voorraad kost geld en levert niet direct waarde op. Daarnaast is voorraad beperkt houdbaar. Dit geldt ook voor requirements. Gedurende het proces veranderen de inzichten en omstandigheden. De focus zou daarom moeten liggen op slechts de belangrijkste features die op korte termijn geïmplementeerd kunnen worden.

5.6 Conclusie

Lean introduceert een scherpe visie als het gaat om efficiëntie en flow in het proces. Deze visie wordt grotendeels ondersteund in de Scrum ontwikkelmethode, maar kan ook binnen individuele deelprocessen verder worden toegepast om optimale flow te bereiken.

Wanneer we kijken naar het proces van requirements engineering dan kunnen de volgende conclusies worden getrokken die als richtlijn worden overgenomen bij het integreren van de processen.

Allereerst wordt het belang van requirements onderstreept. Binnen Scrum wordt de hoeveelheid waarde binnen het proces gerepresenteerd door de *Business Value*. De uiteindelijke waarde die het team echter daadwerkelijk oplevert is afhankelijk van hoe goed de oplossing aansluit bij de wensen van de klant. Requirements spelen daarbij een belangrijke en kwaliteit borgende rol en zouden daarom voldoende aandacht moeten krijgen. Wel moet worden opgemerkt dat requirements specificaties slechts een middel zijn om informatie vast te leggen en over te dragen tussen verschillende partijen. Een middel wat minder van belang is wanneer snel en efficiënte communicatie bewerkstelligd kan worden, echter in alle andere gevallen weldegelijk noodzakelijk en bepalend is. Een beperkte hoeveelheid specificatie zal überhaupt aanwezig moeten zijn om wensen op een efficiënte manier in het proces te kunnen managen, maar de mate van diepgang en detaillering in de specificatie is een balans die met name afhankelijk is van de informatiebehoefte van de ontwikkelaar en informatieaanlevering van de Product Owner.

Binnen de Scrum methode wordt veel aandacht besteed aan het managen en in kaart brengen van de flow. Daarbij wordt echter voornamelijk op de ontwikkel, test en verificatie activiteiten gefocust. De voorbereidingsfase die weldegelijk erg belangrijk is om efficiënt het Scrum proces te kunnen starten (en doorlopen) wordt daarin onderbelicht en de flow van dit deelproces wordt onvoldoende in kaart gebracht. Net als in het ontwikkelproces, is het van belang om ook in deze processen inzicht te krijgen in de status en het verloop van het voorbereidingstraject zodat de algehele flow goed gemanaged kan worden en het voorbereidende activiteiten tijdig zijn afgerond voordat een nieuwe iteratie wordt gestart.
Het feit dat requirements pas op het laatste moment (Just-in-time) worden opgehelderd en uitgewerkt stelt niet alleen hoge eisen aan de requirements processen, maar ook aan de technische voorbereiding (het design). De tijdsdruk die hierdoor ontstaat kan consequenties hebben op de kwaliteit van de specificatie.

Zoals we hebben gezien kent de value stream van softwareontwikkeling tal van terugkoppelingen die een belangrijke bron van waste zijn in het proces. Een deel van deze waste is te wijten aan niet goed functionerende requirements processen. Wanneer we puur kijken naar de waste in (of veroorzaakt door) requirements processen dan zijn de bronnen vooral: onderspecificatie, overspecificatie en fouten in de specificatie. Deze leiden vervolgens weer tot ongevraagde features en bugs.

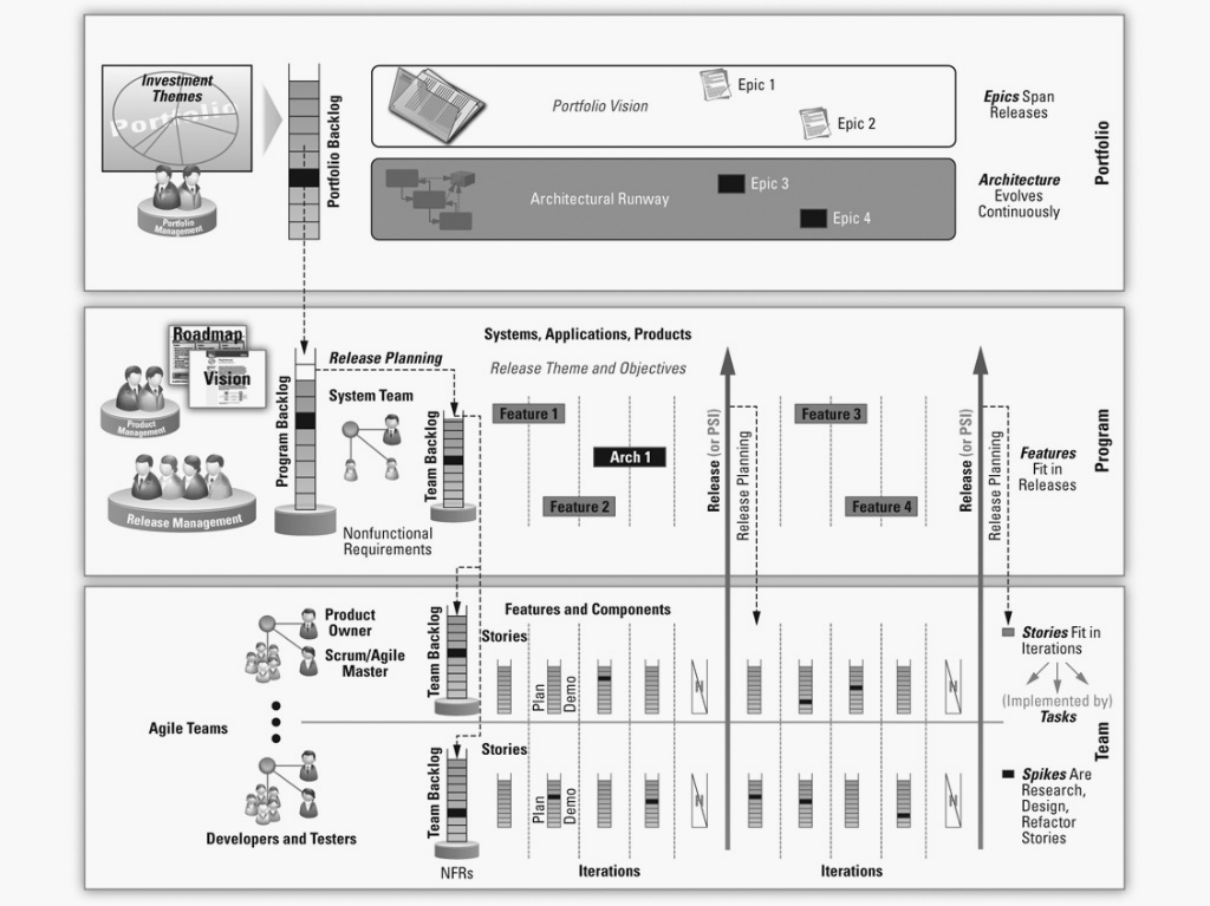
Een andere interessante techniek die binnen Lean gebruikt wordt is het standaardiseren van werk waar mogelijk. Met betrekking tot requirements kan gedacht worden aan templates voor User Stories of

afspraken in het ontwikkelproces. Wanneer we echter kijken naar non-functional requirements dan ligt ook hier weldegelijk een kans om zowel efficiëntie, als meer duidelijkheid en nadruk voor dit, vaak ondergeschoven kindje te krijgen. Een groot deel van de non-functionals zijn vaak op tal van User Stories van toepassing en zouden zowel binnen een project, als (highlevel) ook tussen teams en afdelingen herbruikbaar kunnen zijn.

Lean beschrijft ook de noodzaak om de big picture te zien van de gehele flow. Requirements evolueren daarin gedurende het proces van concept naar meer gedetailleerde uitwerking op de Product Backlog. Het requirementsproces raakt dus raakt dus meerdere niveaus van product management. (Leffingwell, 2011) beschrijft een op Lean gebaseerde procesvisie op software productontwikkeling waarbij hij onder andere de volgende niveaus van abstractie beschrijft en tevens de rol van requirements daarbinnen.

- Portfolio
- Programma
- Team

Zie onderstaande figuur:



Figuur 31 Lean - Agile visie op productontwikkeling (Leffingwell, 2011)

Uit Figuur 31 wordt duidelijk op welk niveau requirements zich kunnen bevinden binnen complexere constellaties (opgeschaald Scrum) en hoe deze wensen vervolgens hun weg vinden van concept naar meer uitgewerkte, concrete activiteit voor het team.

Voor wat betref requirements maakt hij onderscheid in grote (nog vrij abstracte) wensen, minder grote wensen (maar nog te groot voor een sprint), en concrete wensen (die binnen een sprint kunnen worden opgelost). Hij maakt daarbij de volgende indeling:

- Epic:
  Een Epic story beschrijft een conceptueel idee of visie met betrekking tot het product. Een epic wordt doorgaans opgesplitst in één of meerdere Feature of User Stories.

- Feature:
  Een epic story beschrijft een relatief grote wijziging. De omvang van een feature story is dermate groot of abstract dat deze niet binnen één iteratie kan worden opgeleverd. Een feature story dient daarom, naarmate meer details bekend worden, opgesplitst te worden in één of meerdere User Stories.

- User Story:
  Op User Story niveau zijn de wensen dermate concreet dat de requirements in meer detail kunnen worden uitgewerkt. De omvang van een user story zou dusdanig afgestemd moeten worden dat meerdere User Stories binnen een iteratie kunnen worden opgeleverd. Een user story wordt door het team opgesplitst in één of meerdere taken. Als de wens voorkomt uit een technische behoefte wordt ook wel de term Tech Story gebruikt.

Alle type Stories kunnen door het team worden voorzien van storypoints ter indicatie van de omvang. Het spreekt voor zich dat inschattingen van uitgewerkte User Stories betrouwbaarder zijn dan het geval is bij meer abstractere wensen.

Wanneer we kijken naar procesverbetering dan sluit de Plan-Do-Check-Act cyclus van Scrum goed aan bij het Lean gedachtegoed op procesverbetering. Ook A3 Problem Solving, een andere Lean practice, wordt daarin genoemd. Zie bijlage drie en vier voor een toelichting op de PDCA-cyclus en A3 methode. Gezien de knelpunten die in of door de requirements processen kunnen optreden zou een nadrukkelijke aandacht op procesverbetering ook hier voor verbetering kunnen zorgen.

6 Conclusies t.a.v. het literatuuronderzoek

Dit hoofdstuk vormt de conclusie van het theoretisch kader. Diverse factoren die, vanuit een theoretische invalshoek, kunnen bijdragen aan een succesvolle integratie van Scrum en requirements activiteiten, worden in dit hoofdstuk samengevat en geëvalueerd. Op basis van de conclusies die hieruit zijn afgeleid, is vervolgens een concept procesmodel opgesteld waarbinnen de requirements processen binnen Scrum zijn ingetekend. Dit hoofdstuk geeft antwoord op deelvraag: 3b.

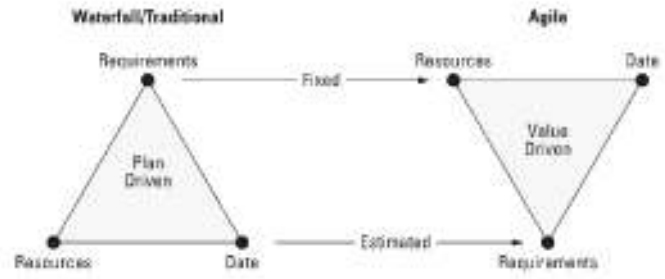
Allereerst zullen we concluderen waarom het lastig is om het proces van requirements engineering op een correcte manier binnen Scrum vorm te geven.

6.1 Inleiding

Agile methoden en processen introduceerden een nieuwe benadering en aanpak van het softwareontwikkelproces. Een transformatie van een plan gedreven aanpak naar een flexibele manier van software produceren. Zoals in voorgaande secties is beschreven kenmerken Agile processen zich door informaliteit en een minimum aan benodigde specificatie. De nadruk ligt binnen deze processen met name op de verbale communicatie en sociale aspecten binnen het ontwikkelteam. Theorieën met betrekking tot requirements en requirementsprocessen hanteren in het algemeen een meer traditionele benadering van het ontwikkelproces waarbij juist de nadruk wordt gelegd op formaliteit, documentatie en het volgen van een gestructureerd plan.

6.2 Verschil in visie

Omdat beide benaderingen op een aantal fundamentele punten verschillen kan het lastig zijn om het traditionele proces van requirements engineering succesvol toe te passen en te beschouwen binnen een Agile context.



Figuur 32 Golden triangle - transformatie (bron: Dean Leffingwell - Agile Software Requirements, 2011)

De volgende fundamentele verschillen kunnen worden onderscheiden met betrekking tot software requirements:

- Managementmodel: Bij een plan gedreven aanpak (traditioneel, waterval) staan de requirements in de beginfase van het project vast. De benodigde resources en doorlooptijd wordt aan de hand daarvan ingeschat (ook wel de *golden triangle* genoemd). Het proces leunt daarbij zwaar op inschattingen uit de beginfase. Agile methoden zijn puur *value* gedreven en hebben een adaptief karakter. Er wordt gewerkt met vaste tijdseenheden en vaste resources. Het aantal op te leveren requirements wordt daarbij geschat en is dus variabel. Door nauwe samenwerking met alle partijen

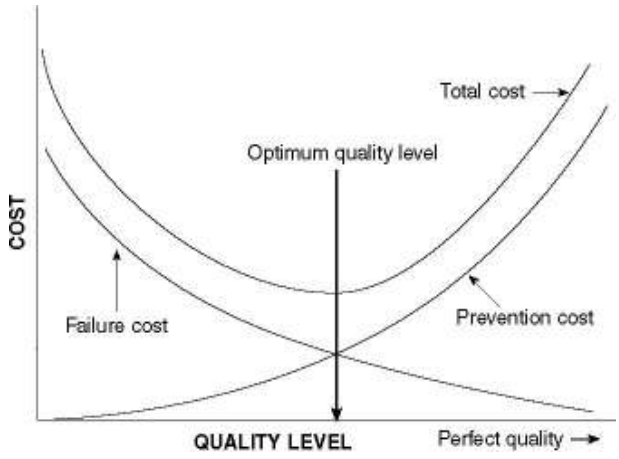
wordt getracht het maximale uit de iteratie te halen. Deze transformatie wordt weergeven in Figuur 32.

- Documentatie: Requirements engineering is erop gericht om een zo goed mogelijke beschrijving te geven van het te ontwikkelen systeem en leunt daarbij zwaar op documentatie. Agile methoden daarentegen zien documentatie als bron voor waste en proberen de omvang daarvan zoveel mogelijk te beperken door face-to-face communicatie centraal te stellen.

- Samenwerking: Bij traditionele methoden speelt documentatie een belangrijke rol in de overdracht van informatie naar andere disciplines of fases in het ontwikkelproces. Er wordt dan ook de nadruk gelegd op formaliteit en compleetheid van de requirements om zo weinig mogelijk informatie verloren te laten gaan bij overdracht. Traditionele methoden zijn sterk proces georiënteerd. Agile is meer op de persoon en klant georiënteerd. Agile methoden stellen directe (face-to-face) communicatie voorop en creëren nauwe samenwerking en korte communicatielijnen door alle disciplines binnen één team (bij voorkeur op één locatie) te verenigen. Zowel het proces, als de uiteindelijke specificaties zijn minder formeel dan binnen een traditioneel proces. Agile methoden stellen de conversatie met de klant centraal, niet de requirements.

- Voorbereiding: Requirements theorieën gaan uit van een uitgebreide analyse en requirements fase. Agile methoden streven er echter naar om zo snel mogelijk te starten met ontwikkelen en door middel van korte iteraties het product (en de requirements) incrementeel tot stand te laten komen. Men streeft daarbij naar *just-in-time* requirements. Dit beperkt de tijd die beschikbaar is voor de voorbereiding en totstandkoming van requirements. Echter doordat beslissingen zo laat mogelijk genomen worden kan optimaal van tot dan toe opgedane kennis worden geprofiteerd en sluit de oplossing beter aan bij de wensen en behoeften van de klant.

- Kwaliteit: Requirements theorieën en methoden streven in het algemeen naar hoge kwaliteit van de opgeleverde specificatie. Agile en Lean stellen echter de behoefte meer centraal en streven naar precies de goede kwaliteit.



Figuur 33 Quality - Costs (bron: J. Juran, opgehaald via www.qualityamerica.com)

Deze verschillen geven aan dat ook op het gebied van requirements engineering een andere aanpak nodig is om op een goede manier aansluiting te vinden met de iteraties en ontwikkelwerkzaamheden binnen een Agile context.

Het verschil in visie i.c.m. de grote verscheidenheid aan beschikbare practices en methoden maken het lastig om de juiste werkwijze te vinden.

6.3 Agile requirements

De volgende sectie beschrijft aan welke voorwaarden requirements en requirements engineering processen moeten voldoen om op een juiste manier aansluiting te vinden bij een Agile methode zoals Scrum en welke factoren daarbij van belang zijn.

6.3.1 Algemeen

Zoals geconcludeerd in de voorgaande sectie heeft het volgen van een Agile ontwikkelstrategie, en Scrum in het bijzonder, een duidelijke impact op het requirements proces. Scrum beschrijft het proces wat meer op project niveau en laat daarbij een groot deel van de invulling aan het team over. Wel stelt het bepaalde voorwaarden aan het proces en veronderstelt het een succesvol uitgevoerde voorbereiding voordat een iteratie kan starten. Het is daarom van belang om inzicht te hebben in de handelingen, rollen en verantwoordelijkheden voor wat betreft het omgaan met requirements in een Scrum project en zowel duidelijkheid, als eensgezindheid te creëren voor de Product Owner en het team.

6.3.2 Conclusies t.a.v. de specificatie

Binnen een Agile context is het van belang dat requirements precies voldoende beschreven zijn. Niet te veel, want overspecificatie is natuurlijk waste. Maar zeker ook niet te weinig. Onduidelijkheid introduceert namelijk fouten en extra communicatie en veroorzaakt dus vertraging en variatie in het proces. Omdat vrijwel iedere requirement uniek is, is het optimale niveau van diepgang en specificatie ook voor iedere requirement verschillend. Tevens is dit afhankelijk van de beschikbare kennis en ervaring van het team. Een eenzijdige definitie vaststellen of criteria handhaven biedt dus niet de meest efficiënte oplossing.

Wanneer gesproken wordt over “precies voldoende beschreven” dan zou de vraag (behoefte) meer centraal moeten staan. Wat heeft een ontwikkelaar precies nodig om efficiënt aan de slag te kunnen met bouwen? Dit kan slechts een korte, summiere beschrijving zijn, maar kan mogelijk ook bestaan uit uitgebreidere vorm, zoals modellen, of aanvullende technische specificatie. Er zou daarom ten minste een check moeten zijn met een ontwikkelaar om te verifiëren of de informatie in een user story toereikend is.

De User Story aanpak sluit goed aan bij deze flexibele aanpak. Een User Story wordt beschreven als placeholder voor de daadwerkelijke requirements. Deze placeholder kan uitstekend worden voorzien van uitgebreidere, of meer diepgaande requirements definities zoals User Cases of functionele modellen als daar behoefte aan is. Dit zal per User Story of team verschillend zijn.

De noodzaak om het ontwikkeltraject te starten met complete specificatie is onderzocht door (Mac Cormack, 97). Resultaten uit zijn onderzoek wezen uit dat de volledigheid van functionele specificatie een significante rol speelt in efficiëntie en productiviteit. Hij vond slechts een zwakke relatie tussen de volledigheid van een gedetailleerd ontwerp en het foutpercentage.

Hieruit wordt geconcludeerd dat: ontwikkelaars productiever zijn naarmate functionele specificatie volledig is afgerond voordat aan het coderen wordt begonnen, maar dat een deel van de technische analyse en ontwerp best mag overlappen met de ontwikkelfase. Dit is ook logisch te beredeneren aangezien ontwikkelaars weldegelijk een volledig plaatje van de exacte wensen moeten hebben om efficiënt de oplossing te ontwikkelen.

6.3.3 Conclusies t.a.v. het proces

Scrum hanteert een pragmatische aanpak als het gaat om requirements en requirementsprocessen. Daarbij wordt ernaar gestreefd om zo snel mogelijk te starten met ontwikkelen en niet te veel tijd aan voorbereiding te besteden. Hoewel het gebrek aan documentatie op de langere termijn weldegelijk voor problemen kan zorgen, kan deze pragmatische aanpak zeer effectief en doeltreffend zijn. Een goede betrokkenheid, samenwerking en efficiënte communicatie tussen het Team en de Product Owner zijn daarbij essentieel.

Volgens Scrum is de Product Owner verantwoordelijk voor een groot deel van de requirements activiteiten (zie hoofdstuk 4). Het team is daarbij sterk afhankelijk van de Product Owner en een juiste invulling van deze veelzijdige rol. Requirements die door de Product Owner niet goed zijn doordacht of omschreven veroorzaken problemen en vertraging in de ontwikkel- of planningsfase. Ook wanneer er veel onduidelijkheid is, of beslissingen lang op zich laten wachten, kan de efficiëntie van het team duidelijk worden geremd. Om de optimale efficiëntie in het ontwikkelteam te bereiken dient de Product Owner dus alle taken op een juiste manier in te vullen. Het is dus erg belangrijk om een goed gekwalificeerde persoon te kiezen voor deze rol. Voor het team is het van belang dat vragen snel en doeltreffend beantwoord kunnen worden en beslissingen ten aanzien van de functionaliteit snel genomen worden. In een ideale situatie is de Product Owner dan ook beslissingsbevoegd voor het gehele product en altijd bereikbaar voor het team.

Wanneer we kijken naar de persoon die normaliter een dergelijke rol toebedeeld krijgt, is dit meestal: de klant, opdrachtgever of (gedelegeerd) productverantwoordelijke. Dit zijn over het algemeen personen die op een heel ander gebied gespecialiseerd zijn en weinig ervaring hebben met productontwikkeling op team niveau (laat staan requirements engineering). Ook hebben zij niet altijd de tijd beschikbaar om voldoende betrokken te zijn bij het team of zich bezig te houden met kleine details in het product.

Een recente Agile Survey (Xebia, 2012) gaf aan dat ook veel Nederlandse organisaties moeite hebben met een juiste invulling van de Product Owner rol. Zaken als beschikbaarheid, het maken van keuzes en mandaat van de Product Owner worden daarbij aangegeven als belangrijkste oorzaken voor het niet slagen van een Agile project.

Je kunt je daarom afvragen of er daarom logischerwijs niet teveel van de Product Owner rol binnen Scrum wordt verwacht en of deze rol in die vorm wel realistisch is. Zeker in complexere constellaties waarbij het voor de Product Owner lastig kan zijn om het systeem op een doeltreffende manier te omschrijven. Er is dan meer initiatief en ondersteuning van het team nodig.

Ook binnen de software community wordt de problematiek rondom de Product Owner rol regelmatig geadresseerd. Verschillende alternatieven worden beschreven om dit knelpunt op een Agile manier op te vangen. Zo beschrijft onder andere Jan Grape in een blog een benadering waarbij de Product Owner rol wordt verdeeld over meerdere personen die gezamenlijk een Product Owner Team vormen. Een Product Owner Team kan bestaan uit een opdrachtgever, een requirements- of business analist, een productmanager, een projectmanager en een programmamanager. Afhankelijk van de behoeften. De veelzijdige Product Owner rol wordt op deze manier realistischer in te vullen en belemmeringen voor het team kunnen zo doeltreffend worden aangepakt. (blog.crisp.se, 2011)

Wanneer producten complexer worden en/of de projectorganisatie uitbreidt, neemt tevens het aantal deelnemers en stakeholders in het proces nemen toe en er ontstaat vanzelf meer behoefte aan het borgen van informatie en structurering van zowel het proces, als de opbouw van het product. Deze behoefte kan komen van externe bronnen zoals wetgeving of gebruikersdocumentatie, maar ook vanuit het team of de organisatie zelf. (Zie Figuur 3: input-output model.) Zaken als: technische documentatie, traceerbaarheid van requirements, afstemming wat betreft de architectuur, standaardisatie et cetera worden dan steeds belangrijker. Om goed in deze behoefte te kunnen voorzien is een meer gestructureerd proces van voorbereiding nodig. De pragmatische “Agile” aanpak is dan niet meer toereikend.

We richten ons in de verdere uitwerking op dit scenario.

6.3.4 Efficiëntie, productiviteit en value

Scrum beschrijft vele principes en richtlijnen om het team optimale efficiëntie te laten bereiken. Het werken in korte iteraties geeft echter de nodige beperkingen voor de voorbereiding van requirements. Deze Just-in-time eisen aan het proces vragen ook om optimale efficiëntie in dit proces en een juiste aansluiting op de engineering processen.

We maken gebruik van de Lean denkkaders om het requirements proces verder vorm te geven.

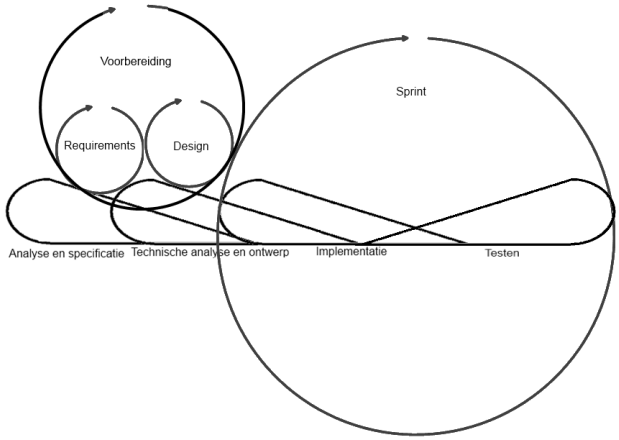
6.3.4.1 Flow

Bij Lean is het streven naar de ideale flow (happy flow), waarbij het proces in eens doorlopen wordt zonder terugkoppeling naar een eerdere fase of andere vormen van waste.

Agile processen veronderstellen dat een bepaalde hoeveelheid requirements is uitgewerkt alvorens een ontwikkeliteratie kan starten. Het requirementsproces kan niet binnen de ontwikkeliteratie plaatsvinden omdat het pas mogelijk is om een accurate inschatting te maken als relevante details voldoende helder zijn en de te kiezen oplossingsrichting bekend is. De requirements- en een deel van de ontwerp fase dienen hiertoe afgerond te zijn.

Met het oog op de voorbereiding van de sprint zou het proces van requirements engineering dus wat moeten voorijlen op het ontwikkel- en ontwerpproces en tegelijkertijd een iteratief en pragmatisch karakter hebben om met de juiste focus en instelling problemen te benaderen. Een deel van de requirements en design activiteiten overlappen met de ontwikkelfase. De ontwikkel-, test- en validatieactiviteiten worden binnen één iteratie (sprint) ten uitvoer gebracht. De requirements- en analyseprocessen dienen dus zowel binnen de sprint, als buiten de sprint plaats te vinden.

Figuur 34 geeft de overlap in de verschillende fases weer tezamen met het verloop van iteraties. De voorbereidende iteraties (requirements en design) zouden gestuurd moeten zijn door de behoefte aan specificatie van het team.



Figuur 34 Agile ontwikkelcycli

De figuur maakt duidelijk dat de iteraties als tandwielen op elkaar moeten aansluiten om de optimale flow te bereiken terwijl de werkzaamheden zelf overlappend zijn. De flow wordt gestart door de requirementsiteraties. Deze initiëren eventuele designiteraties en pas als voldoende requirements helder zijn kan een sprint iteratie worden gestart. Als de sprint eenmaal draait, dienen de requirementsprocessen door te gaan om ervoor te zorgen dat ook de volgende sprint voldoende is voorbereid en het team direct weer kan starten met een nieuwe sprint. Het gehele proces wordt in die zin gedreven en gevoed door requirements. We zouden dit de "*Agile requirements train*" kunnen noemen (zoals ook de "*Agile release train*" een belangrijke factor is in Agile projecten).

Zoals eerder vermeld is het inschatten van User Stories belangrijk voor het kunnen bepalen van een realistische omvang van de werkzaamheden binnen een iteratie en daarmee de variatie te beperken. Inschattingen binnen Agile methoden zijn in het algemeen abstracter en pragmatischer dan binnen traditionele methoden. Maar ook hier geldt: hoe vollediger de informatie, hoe nauwkeuriger de inschatting. Tijdens de sprintplanning zouden dus alle relevante details helder moeten zijn.

Wanneer we de big picture bekijken van het proces, dan hebben we geconcludeerd dat requirements activiteiten zich op verschillende niveaus afspelen en zowel binnen de sprint, als daarbuiten plaatsvinden. We hebben geconcludeerd dat Scrum de voorbereidingsfase onvoldoende in kaart brengt.

Dit zou in het verlengde van de Scrum methode opgelost kunnen worden door ook de activiteiten in de voorbereidingsfase op een Scrum/Kanban bord inzichtelijk te maken en eventueel de voortgang te bespreken tijdens de *Daily Standup*.

Om inzichtelijk te maken in welk stadium een requirement zich bevindt biedt de visie van Dean Leffingwell uitkomst door onderscheid aan te brengen in de omvang en abstractie niveau van individuele wensen (Epic, Feature, Story).

Vanzelfsprekend wordt volgens de Lean strategie gestreefd naar een ideale flow waarin niets fout gaat en alles precies op maat is. Om dit te bereiken bekijken we de bronnen van waste die kunnen optreden in het proces.

6.3.4.2 Waste

In sectie 5.5.3 zijn de verschillende bronnen van waste geïdentificeerd. Onder- en overspecificatie zijn daarbij afhankelijk gebleken van het vinden van de juiste balans in diepgang en uitwerking van de specificatie. Deze balans zal door het team gevonden moet worden en is van veel factoren afhankelijk.

Dit geldt tevens voor het voorkomen van fouten in de specificatie. Door al in een vroeg stadium een aantal nadrukkelijke controlemomenten in te voeren waarbij requirements zowel door de Product Owner, als een ontwikkelaar uit het team worden gevalideerd, komen foute of niet haalbare interpretaties snel aan het licht en kunnen tijdig worden bijgesteld zonder dat verderop in het proces geen hinder wordt ondervonden.

Het beperken van de hoeveelheid voorraad (of vooruitwerken) lijkt in deze context niet zo'n probleem te zijn. Scrum forceert het proces tot korte voorbereidingstijd, waardoor de aandacht alleen op het bovenste (belangrijkste) deel van de backlog wordt gevestigd. De totale hoeveelheid wensen is vaak veel groter. Slechts in eenvoudige context, of bij een overschot aan capaciteit, zou het een aandachtspunt zijn om niet te ver vooruit te werken.

6.3.4.3 Efficiëntie

Het uiteindelijke doel is het bereiken van zoveel mogelijk *customer value*. Ofwel, voor het Scrum team, een zo hoog mogelijke *business value*. Het in kaart brengen van deze *business value*, zoals beschreven in sectie 4.5.2, geeft inzicht van de toegevoegde waarde onder de streep en geeft aan hoe goed dit doel bereikt wordt. Deze gegevens zouden dan ook als de *key performance indicator* voor de ontwikkelorganisatie kunnen worden gebruikt.

Uiteindelijk kan software engineering worden gezien als een research & development activiteit waarbij (op detail niveau) naar de beste oplossingen wordt gezocht voor de totstandkoming van een product. Dit researchaspect is essentieel voor het maken van de juiste keuzes, maar zorgt inherent ook voor variatie in de scope en *triggert* opnieuw communicatie en verificatie cycli met de opdrachtgever indien nieuwe inzichten worden verkregen. Variatie in het proces is hierdoor niet helemaal uit te bannen.

De hectische omstandigheden waarbinnen Agile projecten zich vaak verkeren worden binnen Scrum in zekere zin geaccepteerd. De invloed hiervan op het team wordt zoveel mogelijk beperkt door binnen een duidelijk vastgestelde (beperkte) scope te werken en zoveel mogelijk een constant tempo te hanteren zodat rust en overzicht wordt bewaard. Wijzigingen binnen de scope worden in principe vermeden. Toch wordt het R&D aspect tijdens het uitwerken van de oplossing onderkend. De escalatiestappen van Scrum voorzien in de aanpak van mogelijke knelpunten die hieruit kunnen voortkomen, mits door het team op een juiste manier toegepast. Het bijstellen van de scope tijdens een sprint zou daarom niet per definitie moeten worden vermeden.

Met betrekking tot het aanpakken van knelpunten in het proces worden door Scrum verschillende Lean practices genoemd, zoals A3 *problem solving* en de PDCA cyclus. Net zoals het ontwikkelproces, zou ook hier het requirements proces nadrukkelijk geëvalueerd moeten worden en procesverbetering structureel moeten worden benaderd.

Andere suggesties voor het bereiken van efficiëntie door het volgen van de Lean filosofie is het standaardiseren of automatiseren van terugkerende handelingen. Binnen Agile methoden (met name XP) is het algemeen is veel aandacht voor het automatiseren van bijvoorbeeld test- en releaseprocessen en wordt dit als belangrijke *enabler* van velocity gezien. Door het eenmalig coderen of configureren van een aantal gestandaardiseerde handelingen wordt de effort beperkt en de kwaliteit van het product geborgd. Zoals in hoofdstuk 5 gesuggereerd zou met betrekking tot requirements processen het standaardiseren van non-functional requirements, of andere requirements, indien herbruikbaarheid van toepassing is.

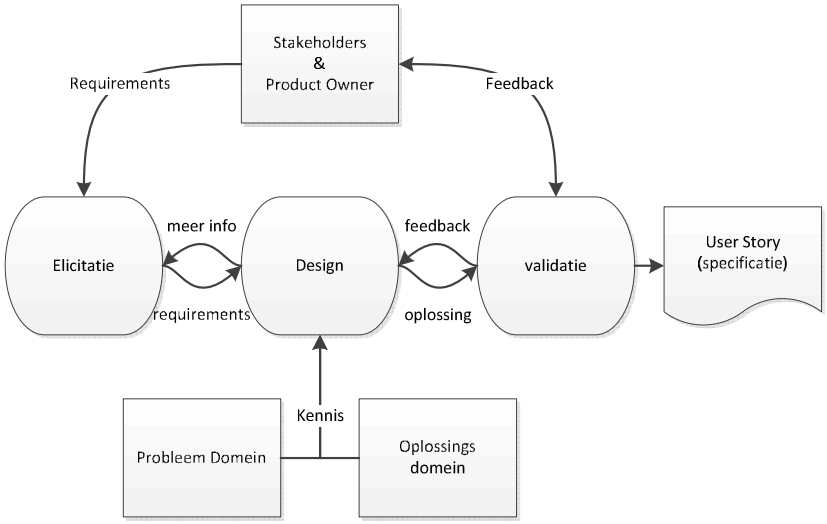
Om de aandacht op kwaliteit borgende maatregelen zoals: code reviews, unit tests, performance tests, functionele tests et cetera. te borgen zou het team doormiddel van een checklist tijdens de sprintplanning kunnen evalueren welke activiteiten van toepassing moeten zijn op een User Story en deze als “standaard” taken aan de User Story toekennen.

6.4 Concept geïntegreerd proces

Conclusies uit voorgaande secties zijn hier verwerkt en vertaald naar een model waarin de requirements processen binnen Scrum zijn geïntegreerd.

Zoals geconcludeerd in sectie 6.3.3 is het team sterk afhankelijk van de Product Owner en is deze rol weinig realistisch. Zoals (Takeuchi & Nonaka, 1984) beschrijven, maar ook door het Lean gedachtegoed wordt ondersteund, zou in dit geval optimaal gebruik gemaakt moeten worden van de expertise binnen het team. Het Team is opgebouwd uit experts die als de beste weten hoe het product zou moeten worden opgebouwd en welke informatie ze daarvoor nodig hebben. Door het team een grotere rol en verantwoordelijkheid te geven met betrekking tot het definiëren en voorbereiden van de requirements, wordt ruimte gecreëerd voor de Product Owner om zich te richten op de management aspecten zoals planning, roadmapping, prioritering, stakeholder management en product management. Activiteiten die tevens significant meer energie vergen in complexere situaties. We komen dan tot het volgende requirements model waarin de Product Owner de bron van informatie is en het eliciteren, specificeren

en valideren van klantwensen door het team wordt verzorgd. Zie Figuur 35. Het team neemt daarbij het initiatief voor het eliciteren en opstellen van de User Requirements en vertaalt deze direct door naar System Requirements.



Figuur 35 Requirements model - Team

Wanneer gekeken wordt naar de deelprocessen die zich binnen het Scrum proces afspelen dan wordt een vergelijkbare groepsactiviteit uitgevoerd tijdens de "*backlog refinement workshop*" (grooming sessie). In deze sessie zou echter het bespreken van de bestaande lijst met backlog items en de technische invulling (design) centraal moeten staan. Door uitgebreide discussies of ideeënuitwisseling over functionaliteit in deze sessie toe te laten zou deze zijn focus verliezen.

We kiezen er daarom voor om het team, in een aparte groepssessie de precieze systeemwensen bij zowel de klant als andere stakeholders boven water te krijgen en deze personen tevens actief te betrekken in de productopbouw (denk aan: *Joint Application Design*, of *Joint Application Development* - sessies).

Dergelijke sprint voorbereidende activiteiten worden meestal parallel aan een sprint uitgevoerd. De introductie van een nieuwe groepsactiviteit zal daarom capaciteit vergen in de sprint. Hiermee dient rekening gehouden te worden.

Doordat het team als geheel opereert kan het gemakkelijk afstemmen hoeveel informatie – en specificatie nodig is in de requirements definitie en hier beter een optimum in vinden.

Het model beschrijft een subtielere rol voor de Product Owner. Deze kan zich zo beter richten op de stakeholders en het vertalen van productwensen. Aangezien snelle - efficiënte besluitvorming en

opheldering van onduidelijkheid essentieel is voor de efficiëntie van het team, zal de Product Owner nog voldoende betrokken moeten zijn en aansluiting moeten hebben met het team.

Figuur 36 geeft het voorgestelde model weer binnen het Scrum procesmodel. In dit model is het van belang dat (zoals in sectie 6.3.2 geconcludeerd), requirements volledig moeten zijn bij aanvang van de sprint. De mate van detail en diepgang zal per requirement (en team) verschillen. Scrum hanteert hiervoor de “ready state”. Ontwikkelaars kennen deze status toe op het moment dat zijn voldoende informatie hebben om aan de ontwikkeling te kunnen starten (en tevens een betrouwbare inschatting is afgegeven). Om aan de “ready state” te voldoen hoeft het implementatie ontwerp (design) niet helemaal gereed te zijn. Wel moet het team een idee hebben over de oplossingsrichting om een goede inschatting te kunnen maken.
Zoals in sectie 2.3.3 duidelijk is geworden, zijn requirementsprocessen vaak niet in een enkele sequentie van handelingen te doorlopen. Om alle wensen goed helder te krijgen en met diverse partijen afgestemd te krijgen, zijn vaak meerdere communicatiecycli nodig. Door het herhaaldelijk uitvoeren van de workshops kan een iteratief requirements proces worden bewerkstelligd.

Om de status en voortgang van sprint voorbereidende activiteiten in kaart te brengen wordt voorgesteld om:

- Gebruik te maken van het model van Dean Leffingwell (sectie 5.6), waarbij backlog items worden beschouwd op verschillende niveaus. Dit helpt het team om eenvoudig onderscheid te maken in het abstractieniveau en omvang van bepaalde wensen.

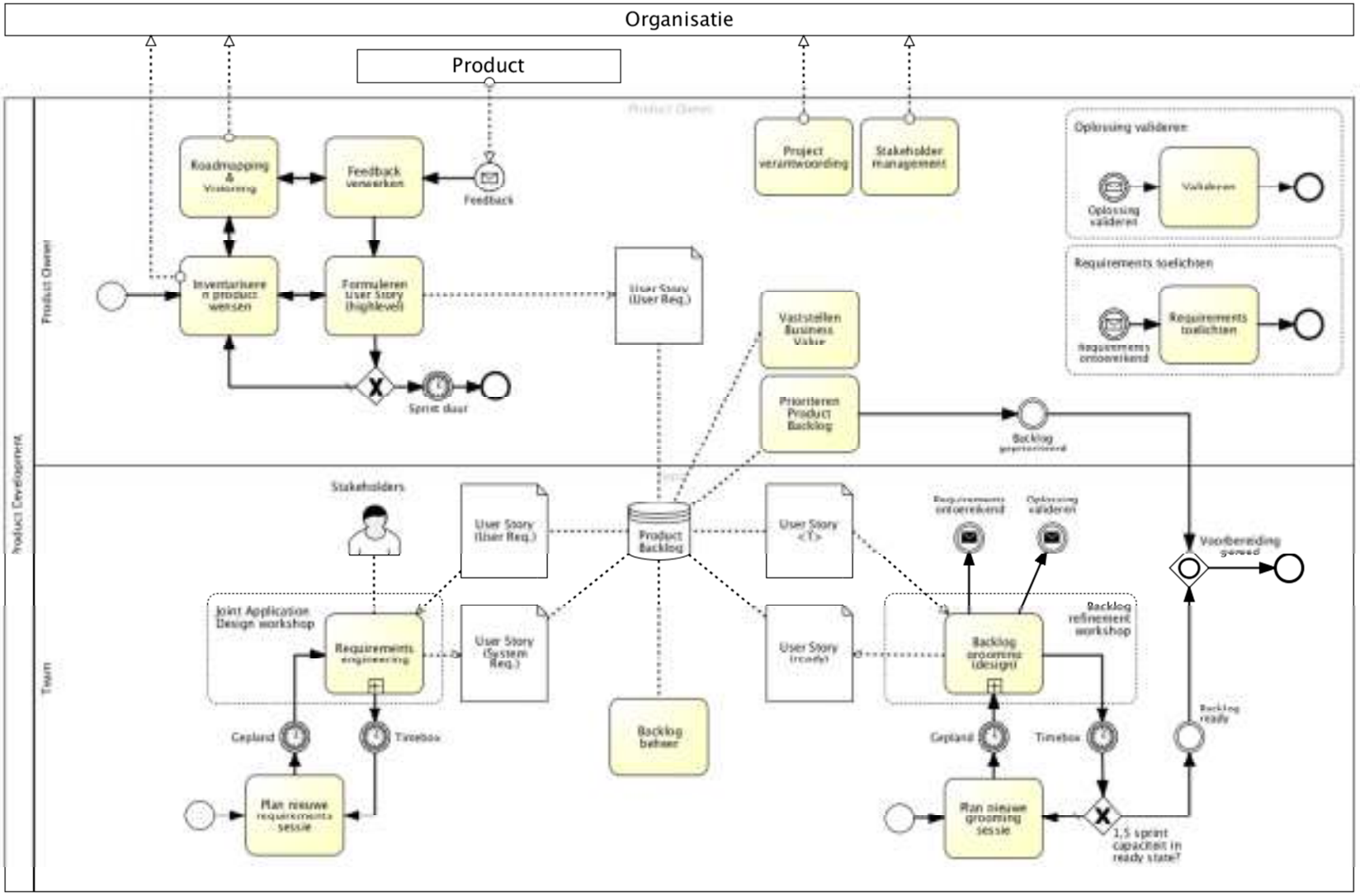
- Het bovenste deel van de backlog (c.a. 1,5 sprint capaciteit) inzichtelijk maken op een Scrum- of Kanban bord. Door de kolommen op het bord aan te passen naar de verschillende statussen die User Stories (en andere backlog items) kunnen hebben, wordt de voortgang in het voorbereidingsproces inzichtelijk gemaakt. ALM[4]-tools (zoals Microsoft Team Foundation Server, TFS2012) kunnen hierin faciliteren.

- Tijdens de *Daily Standup*s van het team naast de voortgang binnen de sprint, ook aandacht te besteden aan de status en voortgang van voorbereidende activiteiten voor de volgende sprint. Hiermee wordt voldoende aandacht op het voorbereidingsproces geborgd.

Wat betreft de aandacht op structurele procesverbetering, zou het sprintvoorbereidingsproces als vast agendapunt tijdens de sprint retrospective meetings moeten worden geëvalueerd.

Om hergebruik en centraal beheer van requirements mogelijk te maken zouden requirements waar mogelijk moeten worden gestandaardiseerd.

[4] Application Lifecycle Managment (ALM)



Figuur 36 Concept geïntegreerd procesmodel - sprintvoorbereiding

Deel 2: Empirisch onderzoek

DISCLAIMER: HOOFDSTUK VERWIJDERD UIT PUBLIEKE VERSIE WEGENS VERTROUWELIJK GEHALTE.

7 De case study

7.1 Inleiding

7.2 Scope en onderzoeksgebied

7.2.1 De organisatie

7.2.2 Technology NL

7.2.3 Product Development NL

7.3 Methode van onderzoek

7.3.1 Onderzoeksstrategie

7.3.2 Respondenten

7.3.3 Verwerken van resultaten

7.4 Resultaten

7.4.1 Bevindingen Team

7.4.2 Bevindingen Management

7.4.3 Bevindingen Opdrachtgevers

7.4.4 Overige bevindingen

7.5 Conclusie

Deel 3: Evaluatie

8 Conclusies en evaluatie

Dit hoofdstuk beschrijft de conclusies en evaluatie van het onderzoek. De relevante inzichten uit het praktijkonderzoek zijn verwerkt en tezamen met de conclusies uit het theoretisch kader verwerkt tot een definitief geïntegreerd model. Dit hoofdstuk wordt afgesloten met algemene conclusies, een korte reflectie op het onderzoek en suggesties voor vervolgonderzoek.

8.1 Conclusies t.a.v. de case

[redacted]

[redacted]

[redacted]

Het behandelen van requirements in de vorm van groepssessies, zoals in het concept model wordt voorgesteld, wordt echter als weinig efficiënt gezien. Zeker in grote projecten, waarbij de conceptualisatiefase moeizaam verloopt zou dit een te grote belasting voor het team vormen. Groepssessies als onder andere sprintplanning en backlog grooming zijn kostbare aangelegenheden. Alle resources zitten al snel enkele uren of dagdelen samen om alle voorbereidingen voor een sprint te treffen. Deze sessies zouden zich moeten concentreren op het bundelen van creativiteit voor het vinden van de beste oplossing (ofwel het design). Hierbij is input van meerdere teamleden essentieel. Het

proces van conceptualisatie en het vaststellen van de wensen zou daartoe al grotendeels afgerond moeten zijn.

Door een afvaardiging van het team, in samenwerking met de opdrachtgevers, het voorwerk te laten doen, kan het team als geheel efficiënt *groomen* (analyse, design), plannen en de implementatiefase doorlopen. Door de centrale rol die de analist heeft in het voorbereidingsproces, doet deze veel kennis op over de wensen, behoeften en achterliggende motieven van de opdrachtgevers. Kennis die bij het team in deze situatie minder aanwezig is. De analist zou daarom als eerste aanspreekpunt voor het team kunnen fungeren bij het ophelderen van klantwensen of implementatiekeuzes (zie rol Product Owner Team). De noodzaak van directe toegankelijkheid van de opdrachtgever (een veelgehoord knelpunt) kan hiermee worden beperkt. Toch blijft diens input en betrokkenheid cruciaal.

Tijdens de initiatiefase van een project is duidelijk behoefte aan een betere voorspelbaarheid van de omvang van werkzaamheden (aantal story points) en benodigde resources (velocity). De accuratie van de inschatting is daarbij van groot belang. Doordat de status van requirements (met betrekking tot compleetheid) niet inzichtelijk gemaakt wordt, is het vaak onduidelijk hoe betrouwbaar een inschatting is. Naarmate meer details bekend worden wordt de inschatting betrouwbaarder. Dat deze inschatting kan variëren is onvermijdelijk. Door de status met betrekking tot de mate van compleetheid aan te geven kan rekening worden gehouden met deze variatie. Geïnspireerd op het model van Kulak & Guiney zullen we hiervoor drie stadia van compleetheid definiëren:

	User Story	
Façade	Initiële definitie: User Requirements, Highlevel, oneliners	Eerste inschatting
Filled	Na eerste analyse: User Requirements uitgewerkt System Requirements initieel	Tweede inschatting
Focused	Na diepgaande analyse: User Requirements compleet System Requirements compleet Design Specification ready	Derde inschatting
(sprintplanning)	Analyse en opsplitsen in taken	**Definitieve** inschatting

Tabel 3 User Story fasering

Door backlogitems van bovenstaande status te voorzien, wordt inzicht verschaft in de betrouwbaarheid van een inschatting en tevens de status van het sprintvoorbereidingsproces. Door rekening te houden met afwijkingspercentages (per status) kunnen betrouwbaardere voorspellingen worden gedaan met betrekking tot iteratieplanning en roadmapping.

Scrum beschrijft een ideale situatie waarbij gewerkt wordt met een team van multi-skilled developers met een "*go where the work is*" mentaliteit. Er hoeft in dit geval weinig rekening gehouden te worden met expertise in de resourceplanning. Uit de conclusies van het praktijkonderzoek blijkt echter dat binnen de casus weldegelijk duidelijke verschillen in expertise en specialisatie aanwezig zijn. Teamleden proberen elkaar zoveel mogelijk te ondersteunen, maar zoals **Fout! Verwijzingsbron niet gevonden.** duidelijk maakt is dit slechts beperkt mogelijk.

Om een goede mix in expertise en resources per sprint te krijgen is daarom inzicht nodig in omvang van werkzaamheden per expertisegebied. Hiertoe zou het inschattingsproces kunnen worden uitgebreid door bijvoorbeeld per User Story aan te geven hoeveel inspanning per expertisegebied nodig is (bijv. in de vorm van een percentage). Binnen de casus zou dit vertaald kunnen worden naar een verdeling tussen:

- % Backend ontwikkelinspanning
- % Frontend ontwikkelinspanning
- % Testinspanning

8.2 Conclusies t.a.v. het model

Op basis van de verkregen praktijkinzichten zijn de conclusies uit sectie 6.4 geëvalueerd en op een aantal plaatsen herzien om beter op de praktijk te kunnen aansluiten.

Beperkingen die zijn geconstateerd ten aanzien van de rol van opdrachtgever zijn in het model verwerkt door de analist een centrale rol te geven in de requirementsprocessen. Dit om meer aandacht te vestigen op de voorbereidingsfase en de volgende doelen te bewerkstelligen:

- Dat de opdrachtgever naar behoefte kan worden ondersteund;
- Een minder ad-hoc proces van idee tot systeem requirements wordt gevolgd;
- Diverse andere (technische) bronnen in acht worden genomen bij het vaststellen van requirements;
- Een direct aanspreekpunt voor softwareontwikkelaars te hebben voor vragen ten aanzien van functionaliteit;
- Een meer realistische invulling van de Product Owner rol;
- Software ontwikkelaars hun focus behouden door minimale inspanning te vragen in de voorbereidingsfase.

We hebben geconcludeerd dat er een duidelijke behoefte is aan een dergelijke brugfunctie tussen opdrachtgever tot softwareontwikkelaars. Het ontwikkelen van een succesvol product vergt afstemming vanuit verschillende disciplines specialisten op ieder vakgebied. Hoewel de input van softwareontwikkelaars in de requirements - en conceptualisatiefase zeer bruikbaar is, wordt het betrekken van het gehele team in de voorbereidingsfase als weinig efficiënt gezien.

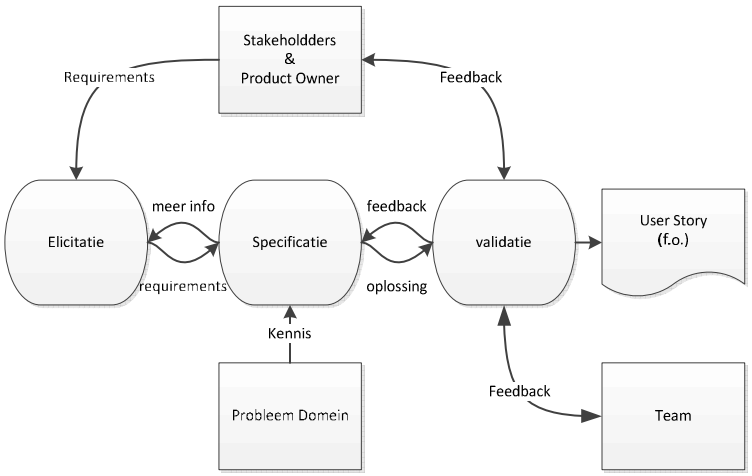
De analist en Product Owner zouden nauw moeten samenwerken om de ontwikkelaars van voldoende informatie te voorzien. Dit samenwerkingsverband zou verder kunnen worden bekrachtigd door een formeel Product Owner Team te vormen.

Op requirements niveau kan het proces worden weergegeven zoals Figuur 37 en Figuur 38. De invulling verloopt daarbij als volgt:

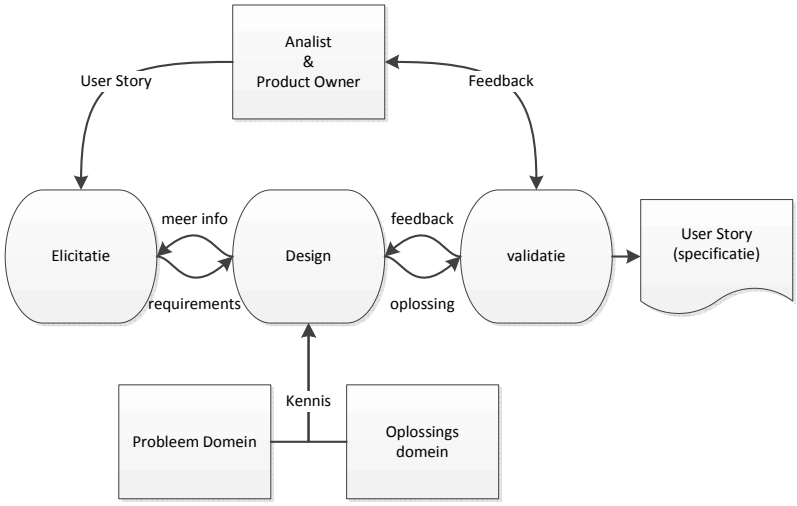
- De analist houdt zich bezig met zowel actieve, als reactieve elicitatie richting opdrachtgever en andere stakeholders. Dat wil zeggen, neemt zelf initiatief ten behoeve van elicitatie maar krijgt ook wensen of conceptuele User Stories aangeleverd die verder moeten worden uitgewerkt.
- De analist vertaalt deze wensen door naar concrete requirements en specificeert deze binnen de User Story. Hierbij wordt gebruikgemaakt van aangeleverde informatie, beschikbare kennis en informatie uit het probleemdomein.
- Om de balans in mate van detail en diepgang in de requirements te verbeteren is de validatiestap uitgebreid door een requirements review te laten plaatsvinden door één of meerdere teamleden (bij voorkeur ontwikkelaars die het waarschijnlijk gaan bouwen). Om deze validatiestap, met name in groepsverband, efficiënt te laten verlopen zou naast de reguliere *document review* ook gebruikgemaakt kunnen worden van een zogenaamde requirements *walk through.* Hierbij worden zowel de requirements, als de context mondeling toegelicht en doorgesproken;
- De analist verzorgt het beheer van de product backlog;
- Directe communicatie wordt waar mogelijk gebruikt om waste en trage besluitvorming te beperken;

- Een selectie User Stories bovenaan de product backlog wordt door het team geanalyseerd en geïnterpreteerd tijdens de *backlog refinement workshop*. Eventuele vragen worden direct beantwoord door de analist of de Product Owner;
- Het team stelt gezamenlijk een oplossingsrichting vast (technisch ontwerp), valideert deze oplossing en maakt een inschatting van de benodigde ontwikkelinspanning (of herziet de bestaande);
- Begrippen als functioneel - en technisch ontwerp worden zoveel mogelijk pragmatisch benaderd. Dat wil niet zeggen dat meer gedetailleerde of formele technieken worden geschuwd, maar dat deze weloverwogen worden ingezet om optimale informatieoverdracht te bewerkstelligen;
- Wanneer voldoende informatie beschikbaar is om aan de ontwikkeling te starten wordt de *ready state* aan een User Story toegekend. Door alleen User Stories in de ready state in de sprintplanning te behandelen kan worden voorkomen dat, bij voorbaat - onduidelijke requirement in een sprint worden ingepland;

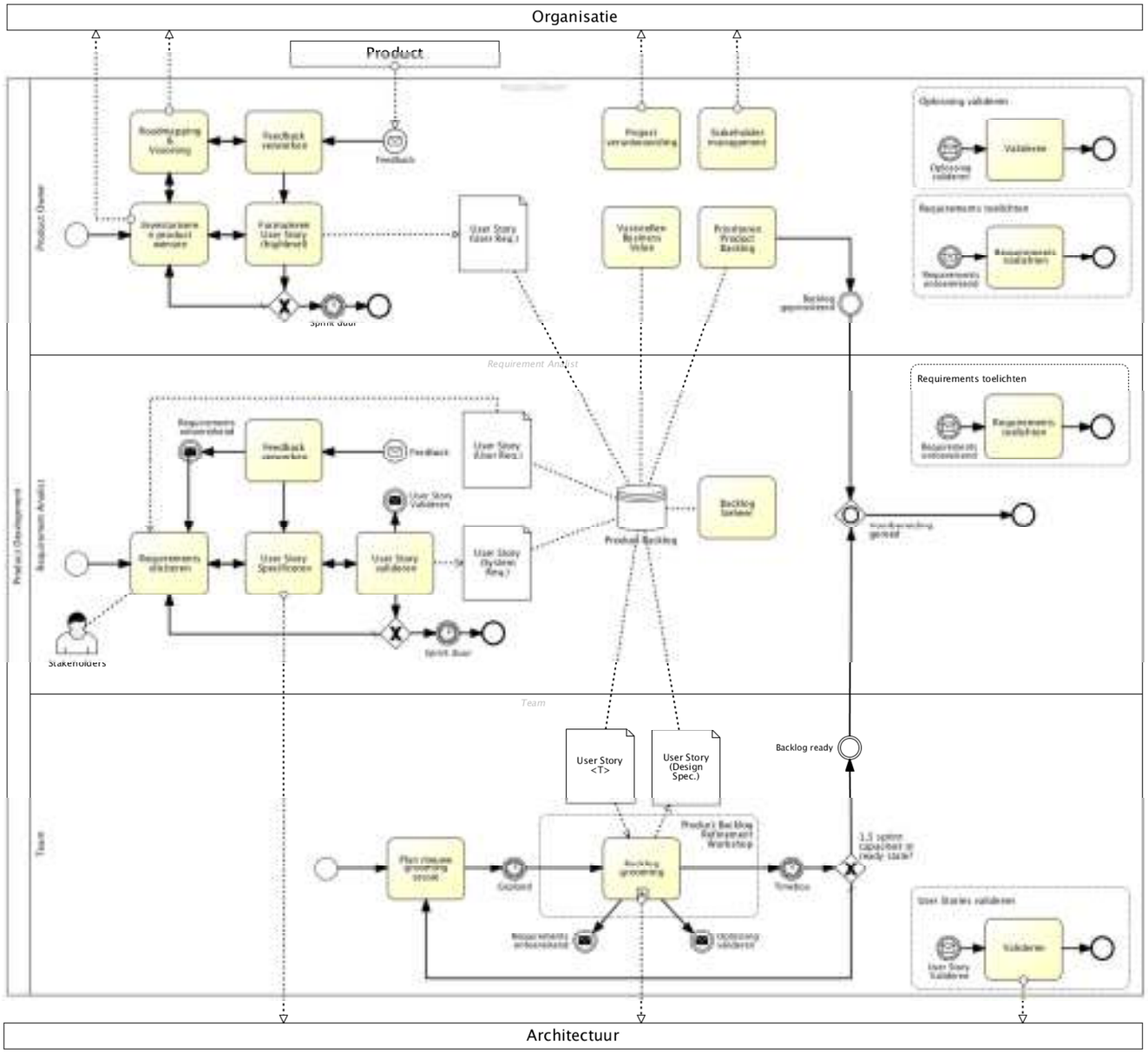
In Figuur 39 wordt de beschreven aanpassing weergegeven binnen het Scrum procesmodel van sprintvoorbereiding. Dit vormt de modellering van de definitieve integratie.



Figuur 37 Scrum requirements model - Analist



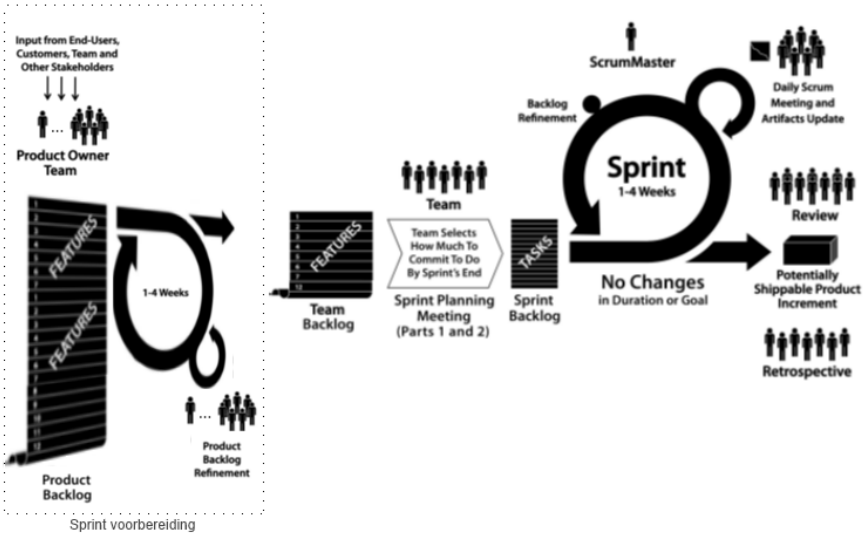
Figuur 38 Scrum design model - Team



Figuur 39 Geïntegreerd procesmodel – sprintvoorbereiding

Gezien de behoefte aan technische consistentie en eensgezindheid met betrekking tot de oplossingsrichting is in het model tevens de relatie met een centrale architectuur afdeling (of rol binnen het team) aangegeven.

Dit definitieve model kan worden beschouwd als aanvulling op Scrum. Om de integratie volledig te maken is daarom ook de procesweergave zoals gegeven in Figuur 13 aangepast om ook het voortraject van een sprint duidelijker te benadrukken.



Figuur 40 Scrum incl. aanvulling

8.3 Algemene conclusies

Requirements vormen de input van het ontwikkelproces en hebben daarmee directe invloed op de kwaliteit van het product, maar ook op de efficiëntie van het team en zouden derhalve voldoende aandacht moeten krijgen om een optimale flow in het gehele proces te bereiken.

Scrum biedt een goed doordacht scala aan richtlijnen en kaders om het proces van softwareontwikkeling optimaal vorm te geven. Daarbij wordt een groot deel ter invulling aan het team overgelaten. Dit is ook het geval voor requirements gerelateerde processen. Iedere organisatie (of zelfs ieder project) heeft te maken met unieke omstandigheden waardoor geen eenduidige aanpak kan worden gehanteerd, maar een proces op maat moet worden gecreëerd om optimaal in de behoeften te kunnen voorzien.

Het team heeft binnen de Scrum methode een grote mate van vrijheid om het proces naar eigen inzichten vorm te geven. De grote diversiteit aan methoden en best-practices op tal van expertisegebieden (die niet altijd met elkaar in overeenstemming zijn) maken het hierbij lastig om tot een optimale invulling te komen. Het is daarom van belang om de samenhang tussen individuele processen te zien en een goed beeld te hebben van het achterliggende gedachtegoed en principes.

Binnen Scrum wordt gestreefd naar een zgn. *Hyper Productive State* waarin een ideale flow wordt bereikt. Processen dienen hiertoe perfect op elkaar aan te sluiten en vormen van *waste* moeten zoveel mogelijk worden uitgesloten. De pragmatische benadering van requirements, die binnen Scrum wordt gehanteerd, is erop gericht om de waste in dit deel van de flow zoveel mogelijk te beperken. In een ideale situatie kan dit veel voordelen opleveren. Gebleken is echter dat een te pragmatische aanpak

weldegelijk een averechts effect kan hebben en zelfs veel waste in het proces kan introduceren. Met name wanneer producten complexer worden en de projectorganisatie uitbreidt zien we dat er meer behoefte ontstaat aan specificatie en documentatie ten behoeve van een efficiënte (en geborgde) informatieoverdracht. Ook de lastig in te vullen Product Owner rol speelt hierbij een belangrijke rol.

Het volgen van een Agile strategie heeft duidelijke consequenties voor het requirements proces. Het belang van een gedegen en gestructureerd requirements proces is afhankelijk gebleken van de complexiteit, informatiebehoeften van het team en beperkingen in de communicatie. Bestaande technieken en methoden kunnen nog weldegelijk worden toegepast, maar zouden naar behoefte en weloverwogen moeten worden ingezet om geen onnodige waste te introduceren. De User Story aanpak faciliteert in deze flexibele vorm van requirements specificatie.

Overal in het proces zullen nadrukkelijke afwegingen moeten worden gemaakt als het gaat om risicobeperking, kwaliteit en efficiëntie. Deze balans is overal terug te vinden. Lean denkkaders en technieken zijn zeer bruikbaar bevonden om deze afwegingen op een verantwoorde manier te maken door een brede visie en strategie op het proces te introduceren.

Deze top-down visie op het proces is in lijn- en complementeert de bottom-up visie van Scrum. Het combineren van beide methoden, zoals in deze thesis is uitgewerkt, kan dan ook worden beschouwd als een verantwoorde manier om het requirements proces vorm te geven binnen Scrum.

Wat betreft procesontwerp zou een brede oriëntatie moeten plaatsvinden op beschikbare practices en tools om het proces vorm te geven. Zoals (Zoet, Heerink, Lankhorst, Hoppenbrouwers, & Stokkum, 2012) beschrijft zou een Agile proces op maat vormgegeven moeten worden op basis van aanwezige behoeften. Daarbij is het van belang om niet op een specifieke methode te blijven hangen, maar een selectie te maken van beschikbare Agile practices (zgn. *Process Fragements*). (Esfahani & Yu, A Repository of Agile Method Fragments, 2010). Er zijn bibliotheken (repositories) beschikbaar waarin individuele deelprocessen worden beschreven in de vorm van *Proces Fragments*. Een selectie is daarmee vrij gemakkelijk samen te stellen. (Method Fragments [www.processexperience.org]) (Goal Oriented Repository of Method Fragments [cs.utoronto.ca])

Tot slot kan worden geconcludeerd dat het succes van een project staat of valt met een goede samenwerking tussen alle betrokken partijen. Een goede persoonlijke klik is daarbij cruciaal. Een optimale samenwerking die leidt tot de beste resultaten vereist een nauwe betrokkenheid van alle partijen. Teamleden en opdrachtgevers zullen daarom veelvuldig moeten samenwerken en vraagstukken gezamenlijk (en met inzicht vanuit meerdere disciplines) moeten tackelen om de meeste waarde uit een iteratie te halen.

8.4 Reflectie

In dit onderzoek is een model ontwikkeld waarmee het proces van requirements engineering op een verantwoorde manier wordt geïntegreerd binnen het Scrum framework. Daarnaast zijn diverse richtlijnen gegeven om dit proces in de praktijk vorm te geven en is tevens een Lean - Agile visie op requirements en requirements engineering geïntroduceerd. Deze visie kan organisaties helpen bij het

creëren van een proces op maat waarbij de aandacht op de juiste plaatsen wordt geconcentreerd om een optimale flow te bereiken. Het ontwikkelde model biedt een concrete invulling van het proces. Zoals geconcludeerd in hoofdstuk 4.7 is een dergelijke modellering niet universeel toepasbaar. Het model is van toepassing op organisaties waarbij:

- Scrum als procesmethodiek wordt toegepast;
- Knelpunten worden ondervonden met betrekking tot de Product Owner rol;
- Behoefte is aan een meer gedegen proces van sprintvoorbereiding en requirements engineering;
- Het team de beschikking heeft over een analist die zich voldoende op de voorbereidingsfase kan richten.

Het model geeft de requirements activiteiten weer binnen het Scrum proces en brengt tevens de rollen en verantwoordelijkheden in kaart. Impliciete aspecten zoals keuzes en afwegingen die binnen het proces moeten worden gemaakt zijn afhankelijk gebleken van vele omstandigheden. Inzichten zoals beschreven in de Lean-Agile visie op het proces kunnen worden gebruikt om deze keuzes op een verantwoorde manier te maken.

Graag had ik in dit onderzoek ook procesoptimalisatie- en volwassenheidframeworks betrokken zoals Lean Six Sigma en het CMMI (Capability Maturity Model Integration). Hoewel in de literatuurstudie grotendeels meegenomen, zijn deze onderwerpen in dit verslag buiten beschouwing gelaten wegens het noodgedwongen moeten reduceren van de scope. Toch zij beide onderwerpen zijn relevant gebleken als het gaat om gestructureerde procesoptimalisatie, institutionalisering en het bereiken van een hogere volwassenheidsniveaus en zouden een toevoeging kunnen zijn aan hetgeen in deze thesis is uitgewerkt.

Het CMMI zou bijvoorbeeld kunnen helpen om de huidige processen (procesgebieden) te toetsen en actiepunten te identificeren die nodig zijn om naar een hoger volwassenheidsniveau toe te werken. Ook hier wordt binnen Scrum wel aandacht aan besteed maar mist een gestructureerde aanpak en een concreet doel. Hoewel het CMMI wat stoffig aanvoelt (doordat het hoge eisen stelt aan de kwaliteit van processen en een daarmee een vrij traditioneel karakter heeft) kan het weldegelijk succesvol worden gecombineerd met Agile methoden. Sutherland, R. Jakobsen, & Johnson (2007), concludeert zelfs een beduidend hogere mate van effectiviteit, kwaliteit, klanttevredenheid en kortere levertijden wanneer Scrum wordt toegepast in een organisatie met een hoog volwassenheidsniveau (CMMI level 5) en waarbij tevens Lean als optimalisatie framework wordt toegepast. Deze combinatie wordt ook wel "*The magic potion for code warriors*" genoemd.

Hoewel het CMMI een norm kan bieden op organisatieniveau, is het vermoedelijk te abstract om te kunnen voorzien in een norm op team- en individueel niveau. De Lean denkkaders en technieken bieden uitkomst binnen dit deel van het spectrum door duidelijk te identificeren wat echt van belang is, het proces inzichtelijk te maken en stapsgewijs toe te werken naar een optimale flow en werkwijze. Lean Six Sigma is een methode of framework wat hierin nog wat verder gaat. Doormiddel van diverse procesindicatoren wordt het proces op een meer statistische manier in kaart gebracht en op basis van cijfers bijgestuurd. Het vermoeden bestaat dat deze benadering lastig is toe te passen binnen het proces

van softwareontwikkeling aangezien harde cijfers lastig zijn vast te stellen en bijna alles variabel is. Toch kan het een meer concrete invulling geven aan het Lean gedachtegoed en procesoptimalisatie in het algemeen.

8.5 Vervolgonderzoek

Zoals in voorgaande secties is geconcludeerd heeft iedere organisatie een unieke combinatie aan behoeften, mogelijkheden en beperkingen. Er is daardoor niet één beste oplossing of methode. De sleutel zit hem in een juiste combinatie van practices en het vinden van de balans in het proces. De Agile en Lean denkkaders en principes kunnen hier houvast bieden. Toch zou het bruikbaar zijn om organisaties wat meer concrete richtlijnen te geven voor het samenstellen van een dergelijk proces op maat.

In deze thesis is de rol van requirements engineering binnen Scrum beschreven vanuit het perspectief van een relatief complexe ontwikkelorganisatie. Dit model zou kunnen worden gebruikt als uitgangspunt voor een mogelijk abstractere modellering waarbinnen verschillende type organisaties worden onderscheiden. Men zou kunnen denken aan een classificatiemodel voor organisaties waarbij verschillende best-practices per situatie worden aangeboden, of bijvoorbeeld een matrixmodel waarin specifieke behoefte worden afgezet tegen beschikbare *Process Fragements*.

Zoals in sectie 8.4 wordt geconcludeerd zou verder onderzoek gewenst zijn naar de bruikbaarheid en mogelijke integratie van het CMMI en Six Sigma binnen het Scrum proces en requirements processen in het bijzonder.

Een ander punt waar verder onderzoek toegevoegde waarde zou bieden, is de integratie en rol van Product Management binnen het Scrum proces. Steeds meer organisaties hebben Product Managers in dienst. Zij hebben weldegelijk een invloedrijke rol in het ontwikkelproces. Hoewel Scrum wel ingaat op de rol van Project Management, laat het Product Management buiten beschouwing. Het zou daarom gewenst zijn om meer inzicht te krijgen in de mogelijkheden en best practices om deze rol optimaal vorm te geven binnen de context van Scrum.

De rol van Product Owner is binnen dit onderzoek problematisch gebleken. We hebben dan ook meerdere schikkingen gedaan om een meer realistische invulling van deze rol mogelijk te maken. Doordat echter maar één praktijksituatie is getoetst, kunnen deze bevindingen niet als algemeen geldend worden beschouwd. Echter signalen uit de branche doen vermoeden dat meer organisaties moeite hebben met dit aspect van Scrum. Kwantitatief onderzoek om in kaart te brengen in welke mate dit probleem speelt (en zich verhoudt tot andere knelpunten) zou daarom welkom zijn.

Scrum Alliance, de community van Scrum professionals (onder hoede van de auteurs van Scrum), staat open voor alle mogelijke vormen van feedback en voorstellen tot verbetering van de methode. Er zijn regelmatig evenementen en bijeenkomsten waar ervaringen worden uitgewisseld. Door het terugkoppelen van bevindingen uit individuele onderzoeken kan deze community worden geholpen om de methode zelf, maar ook educatie van de methode (certificeringstrajecten), nog beter vorm te geven.

9 Bronnen

Abrahamsson, P. (2002). *Agile Software Development Methods - review and analysis.* VTT.

Annosi, Esfahani, & Yu. (2011). *Strategically Balanced Process Adoption.* ICSSP.

Barton, B., Schwaber, K., & Rawsthorne, D. (2005). *Reporting Scrum Project Progress to Executive.* Scrum Alliance.

Birkhölzer. (2011). *Goal-Driven Evaluation of Process Fragments Using Weighted Dependency Graphs.* ICSSP.

blog.crisp.se. (2011). *The Product Owner team*. Opgehaald van Grape, Jan: http://blog.crisp.se/2011/06/08/jangrape/1307557795354

Braafhart, L. (2010). *Richtlijn voor professionele productontwikkeling - CMMI1.3.* Academic Service.

Cohen, D., Costa, P., & Lindvall, M. (2004). An Introduction to Agile Methods. *Advances in Computers, vol. 62*.

Decreus, K., Kharbili, M., Poels, G., & Pulvermueller, E. (2009). *Bridging Requirements Engineering and Business Process Management.* Universiteit van Gent.

ESA. (1995). *Guide to the software requirements definition phase.* Parijs: European Space Agency.

Esfahani, & Yu. (2010). *A Repository of Agile Method Fragments.* Springer-Verlag.

Esfahani, Yu, & Cabot. (2010). *Situational Evaluation of Method Fragments: an Evidence-Based Goal-Oriented Approach.* CAiSE.

Gallardo-Valencia, R., Olivera, V., & Elliott Sim, S. (sd). *Are Use Cases Beneficial for Developers Using Agile Requirements?* IEEE.

Goal Oriented Repository of Method Fragments [cs.utoronto.ca]. (sd). Opgeroepen op 11 4, 2012, van cs.utoronto.ca: http://www.cs.utoronto.ca/~hesam/MFR/MF.htm

Interview: Jim Johnson of the Standish Group. (sd). Opgeroepen op 7 12, 2012, van infoq.com: http://www.infoq.com/articles/Interview-Johnson-Standish-CHAOS

Johnson, J. (2002). XP2002. *Standish Group Study Report.*

Kulak, D., & Guiney, E. (2004). *Use Cases - Requirements in context.* Pearson Education, Inc.

Laanti, M., Salo, O., & Abrahamsson, P. (2010). *Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation.* Elsevier.

Leffingwell, D. (2011). *Agile Software Requirments.* Addison-Wesley.

Martin, S., Aurum, A., Jeffery, R., & Paech, B. (2002). *Requirements Engineering Process Models in Practice.*

Melo , C., Cruzes, D., Kon, F., & Conradi, R. (2011). *Agile Team Perceptions of Productivity Factors.* IEEE.

Method Fragments [www.processexperience.org]. (sd). Opgeroepen op 11 4, 2012, van processexperience: http://www.processexperience.org/fragments

Middleton, P., & Sutton, J. (2005). *Lean Software Strategies.* New York: Productivity Press.

Orr, K. (2004). *Agile Requirements - Opportunity or Oxymoron.*

Paetsch, F., Eberlein, A., & Maurer, F. (2003). Requirements Engineering and Agile Software Development. *Computer Society IEEE*.

Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development - An Agile Toolkit.* Addison-Wesley Professiona.

Poppendieck, T., & Poppendieck, M. (2006.). *Implementing Lean Software Development: From Concept to Cash.* Addison-Wesley.

Reel, J. (1999). Critical Success Factors In Software Projects. IEEE Software.

Rinsing, L., & Janoff, N. (2000). *The Scrum Software Development Process for Small Teams.* IEEE Software.

Rising, L., & Janoff, N. (2000). *The Scrum Software Development Process for Small Teams.* IEEE.

Rolland, C. (1993). *Modeling the Requirements Engineering Process.*

Scrum Inc. (2012, Juni 2). *Enabling Specifications: The Key to Building Agile Systems*. Opgeroepen op Februari 16, 2013, van scrum.jeffsutherland.com: http://scrum.jeffsutherland.com/2009/11/enabling-specifications-key-to-building.html

Shams-Ul-Arif, Khan, Q., & Gahyur, S. (2010). *Requirements Engineering Process Tools/Technologies & Methodologies.*

Sutherland, J. (2011). *Scrum Handbook.* Scrum Training Institute Press.

Sutherland, J., & Schwaber, K. (2011). *The Scrum Papers: Nuts, Bolts, and Origins of an Agile Framework.* Paris: Scrum Inc.

Sutherland, J., & Schwaber, K. (2012). The Crisis in Software: The Wrong Process Produces the Wrong Results. In *Software in 30 days.* John Wiley & Sons.

Sutherland, J., R. Jakobsen, C., & Johnson, K. (2007). *Scrum and CMMI Level 5: The Magic Potion for Code Warriors.* Washington, DC: Agile Conference.

Szalvay, V. (2004). *An Introduction to Agile Software Development.* Danube.

Takeuchi, H., & Nonaka, I. (1984). *The new new development game.* Harvard Business School.

VersionOne. (2011). *State Of Agile Survey.* Version One Inc.

Womack, J., & Jones, D. (1996). *Lean Thinking.* London: Simon and Schuster.

Xebia. (2012). *Agile Survey Nederland.* Xebia Nederland.

Zoet, M., Heerink, A., Lankhorst, M., Hoppenbrouwers, S., & Stokkum, W. v. (2012). An Agile Way of Working. In M. Lankhorst, *Agile Service Development - Combining Adaptive Methods and Flexible Solutions.* Berlin Heidelberg London New York Johannesburg Mumbai Punta Arenas Wlandiwstok: Springer Verlag.

10 Bijlagen

Bijlage 1: BPMN2 overzicht toelichting syntax

Bijlage 2: Value Stream Map

Bijlage 3: Kwaliteitscirkel van Deming

Bijlage 4: A3 Problem Solving

Bijlage 5: Non-functional requirements

Bijlage 6: Use Case Diagram

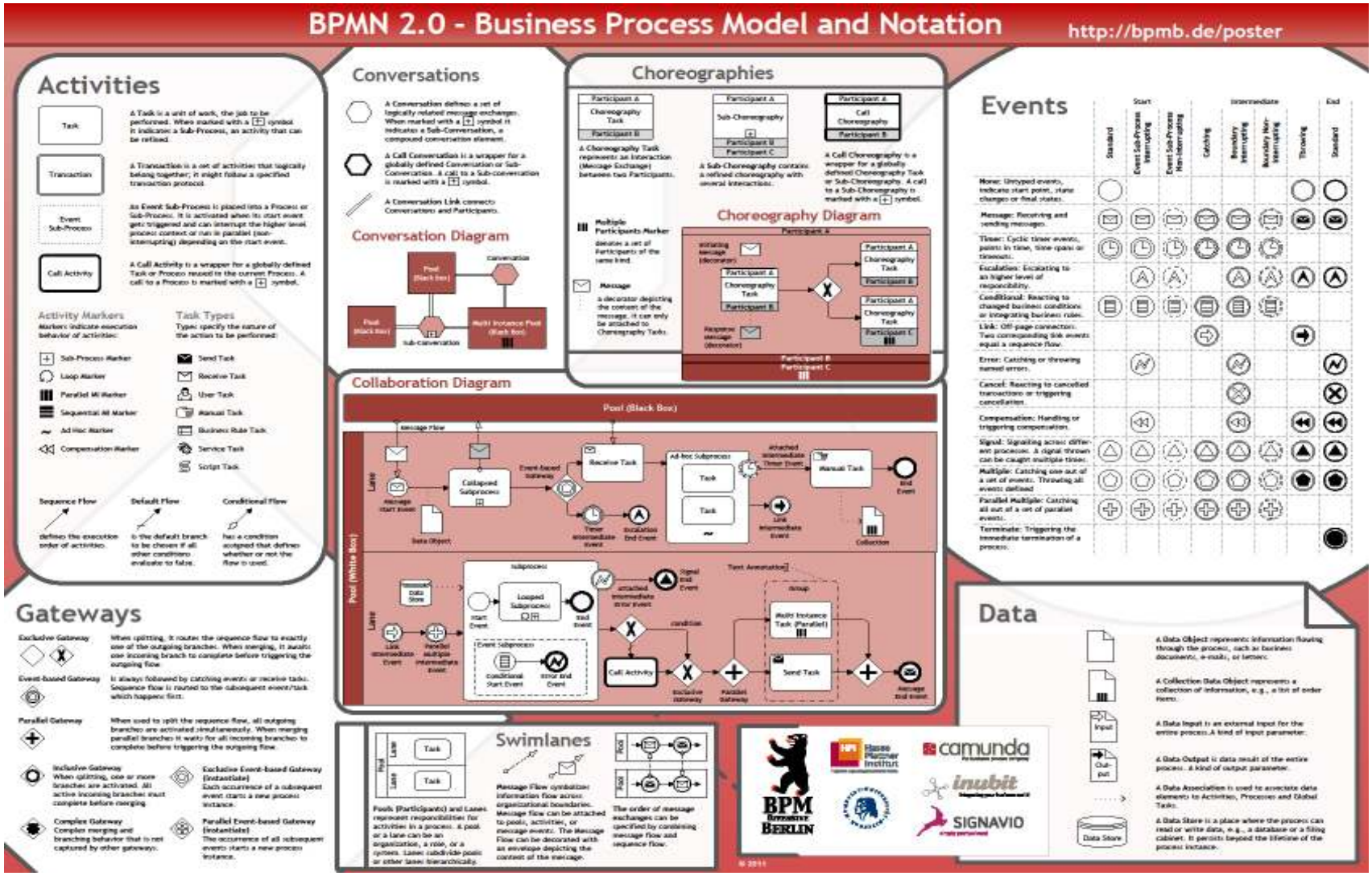
Bijlage 7: Definition of Done

(aanvullende bijlagen: separaat document)

Bijlage 8: Sprint Burn Down Charts

Bijlage 9: Sprint Retrospectives

Bijlage 1: BPMN2.0 – reference guide



Figuur 41 BPMN2.0 (bron: http://www.signavio.com/)

Bijlage 2: Value Stream Map

Om de stroom van waarde in de organisatie in kaart te brengen wordt binnen het *Lean Production* proces gebruikgemaakt van een techniek genaamd een *value stream mapping*. Een *value stream map* is een schematische voorstelling (momentopname) van de productieflow binnen een organisatie. Het beschrijft alle stappen die nodig zijn voor de totstandkoming van een product of dienst en de productie- en wachttijden binnen het proces alsmede de voorraadaantallen. Een value stream map is met name nuttig om de *big picture* te zien van de processen in een organisatie. Tevens breng het ook de waste in het proces in kaart. Volgens Lean principes zou de focus zou moeten liggen op het optimaliseren van de gehele value stream om de *order-to-cash* tijd te verkorten. Bij het opstellen van een value stream map worden de volgende vier stappen doorlopen:

1. Bepaal de verschillende productgroepen;
2. Breng de huidige situatie in kaart (*value stream map* van de huidige situatie, focus op één productgroep);
3. Breng de ideale situatie in kaart (*value stream map* van ideale situatie zonder *waste*);
4. Voer een implementatie plan in voor optimalisatie van de *value stream*.

Vervolgens wordt een stap-voor-stap verbeterplan (*Kaizen*) uitgevoerd waarin de *waste* in het proces in kaart wordt gebracht en wordt geëlimineerd om de uiteindelijke flow te verbeteren.

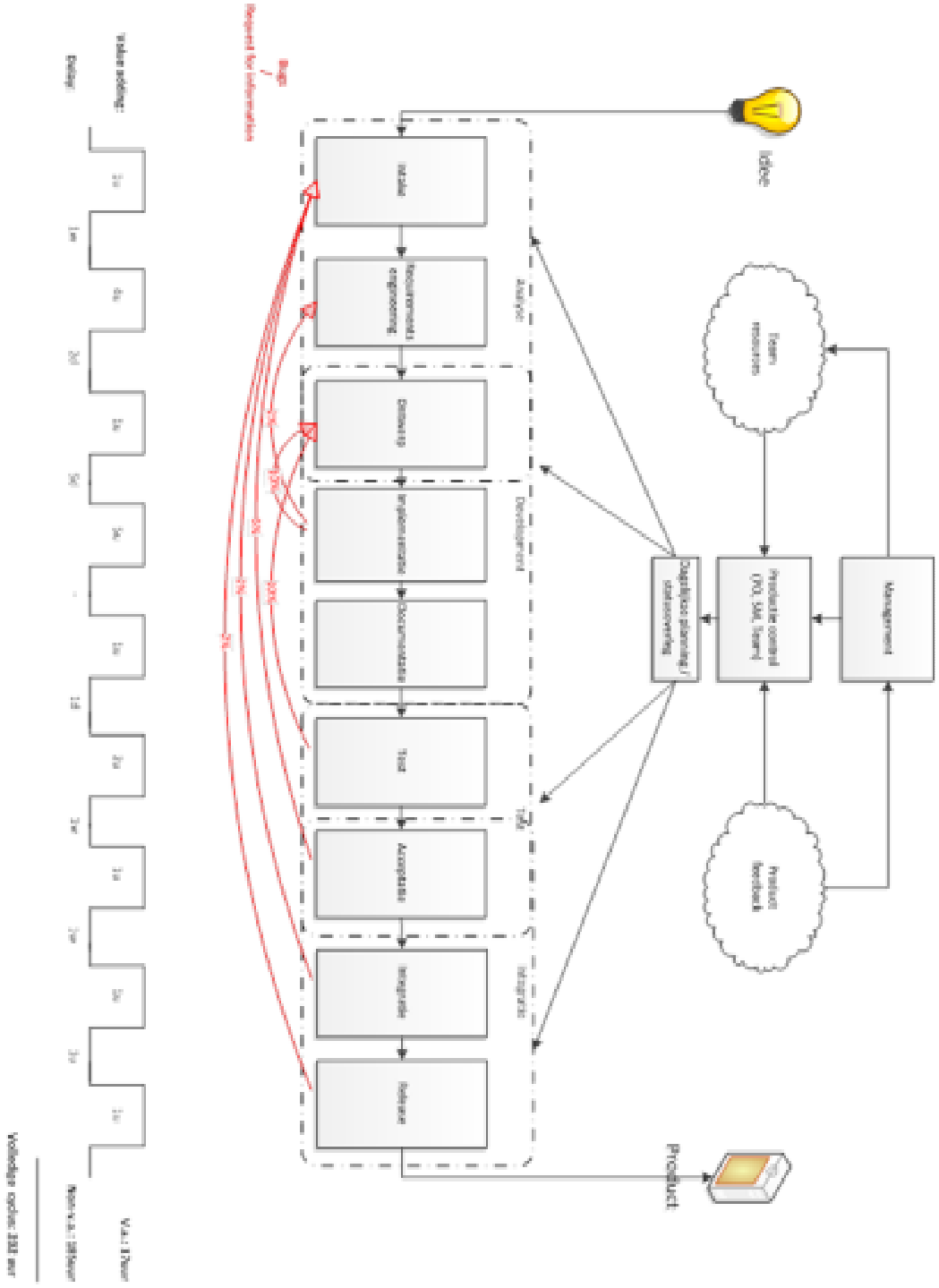
Value stream mapping is binnen softwareontwikkeling niet erg gebruikelijk. Toch kan het een goed middel zijn om het proces inzichtelijk te maken en waste te identificeren. Het rekenen met exacte tijdseenheden en voorraadaantallen binnen de value stream is echter lastig. Dit komt met name doordat telkens unieke producten van verschillende omvang en complexiteit worden geproduceerd. Het tot in detail vastleggen van de value stream is hierdoor lastig en een abstracter niveau zal moeten worden gehanteerd.

Figuur 42 geeft een voorstelling van een value stream map van een (fictieve) software organisatie:

In de figuur wordt direct duidelijk waar de feitelijke waste in het proces zit. Tevens is duidelijk zichtbaar in hoeverre bugs of informatieverzoeken de doorlooptijd beïnvloeden doordat delen van de cyclus opnieuw doorlopen moeten worden. Aan de hand van proces indicatoren kan de hoeveelheid waste wat hierdoor veroorzaakt wordt worden berekend en tevens het effect op de cyclustijd en efficiëntie onder de streep.

Het verschil tussen de doorlooptijd en *value adding time* geeft de *delay* in het proces weer. De zogenaamde *process cycle efficiency* is een kengetal wat als volgt kan worden berekend:

$$Process\ Cycle\ Efficiency\ (\%) = \frac{Value\ added\ time}{Cycle\ Time}$$



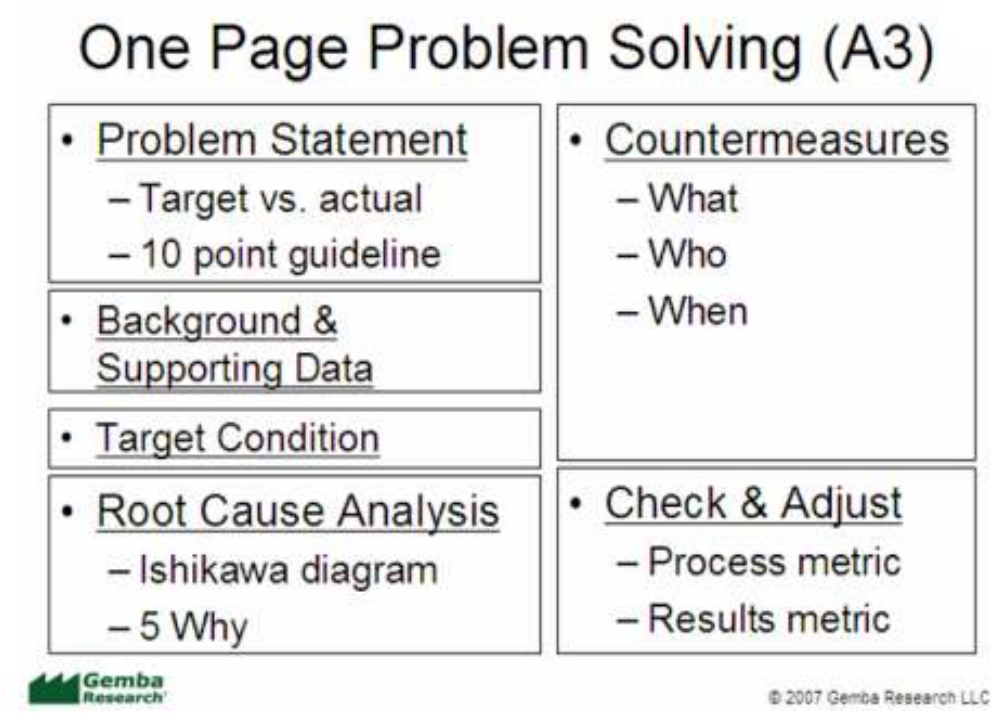
Figuur 42 Value stream map software organisatie

Een *value stream map* is een momentopname. Omdat de omvang van de werkzaamheden, foutpercentage etc. voor iedere requirement verschillend is, kunnen exacte waarden niet gemakkelijk worden bepaald. Het werken met gemiddelden kan echter relatief eenvoudig een indicatie geven zonder dat alle individuele handelingen afzonderlijk hoeven te worden gemeten.

Bijlage 3: A3 Problem Solving

Een probleemoplossingsstrategie uit het Lean gedachtegoed is A3. Een A3 formulier dient daarbij als communicatie placeholder voor het beschrijven en structureel oplossing van problemen en knelpunten in het proces. De naam A3 is afkomstig van het papierformaat dat voor de methode werd gebruikt. A3 en *A3 thinking* – zoals de manier van werken ook wel genoemd wordt – volgt de kwaliteitscirkel van Deming (zie bijlage vier) en beschrijven een transparante en concrete manier om deze cyclus te doorlopen.

Ieder probleem krijgt een eigen vel papier. De linkerzijde van het papier beschrijft het probleem, achtergrondinformatie, meetbare doelstellingen en een oorzaakanalyse. Dit zijn typisch de resultaten uit de plan fase van de Deming-cirkel. De rechterzijde van het papier beschrijft de concrete actiepunten (do), de bevindingen (check) en de daarop te nemen maatregelen (act). De onderstaande figuur illustreert de indeling en opbouw van een A3 formulier.



Figuur 43 One Page Problem Solving (bron: Gemba Research)

A3-*problem solving* kan dus worden gebruikt om het proces van probleemoplossing te structureren en transparant te maken. De grondigheid waarmee A3 het probleem aanpakt leent zich niet voor alle type problemen, maar is met name geschikt voor het nemen van beslissingen waaraan grotere risico's verbonden zijn of waarbij meerdere mensen in de organisatie geraakt worden bij het doorvoeren van een maatregel. Kleinschalige optimalisaties worden *Kaizen* genoemd. *Kaizen* staat letterlijk voor "het verbeteren en goedmaken" en wordt binnen Lean beschreven als het continue proces van kleine optimalisaties. Kaizen zijn de maatregelen tegen lokale problemen die snel opgelost kunnen worden. *Hansei* staat voor het kritisch kijken naar het proces en opsporen van knelpunten. Beide staan centraal in het Lean raamwerk.

Bijlage 4: Kwaliteitscirkel van Deming

Ook wel bekend als de plan-do-check-act cyclus. Beschrijft een eenvoudig model waarbij een continu proces van kwaliteitsverbetering plaatsvindt. De volgende fases worden onderscheiden:

- Plan:
  Dit omvat het plannen en voorbereiden van alle doelen ter verbetering. Hierbij worden o.a. de scope, mijlpalen, risico's, resources, rollen en verantwoordelijkheden vastgelegd zodat te nemen acties duidelijk zijn. Ook wordt vastgesteld hoe de doelen meetbaar kunnen worden gemaakt.

- Do:
  Het uitvoeren van de geplande acties.

- Check:
  De effecten van de acties worden gemonitord, gerapporteerd en geëvalueerd.

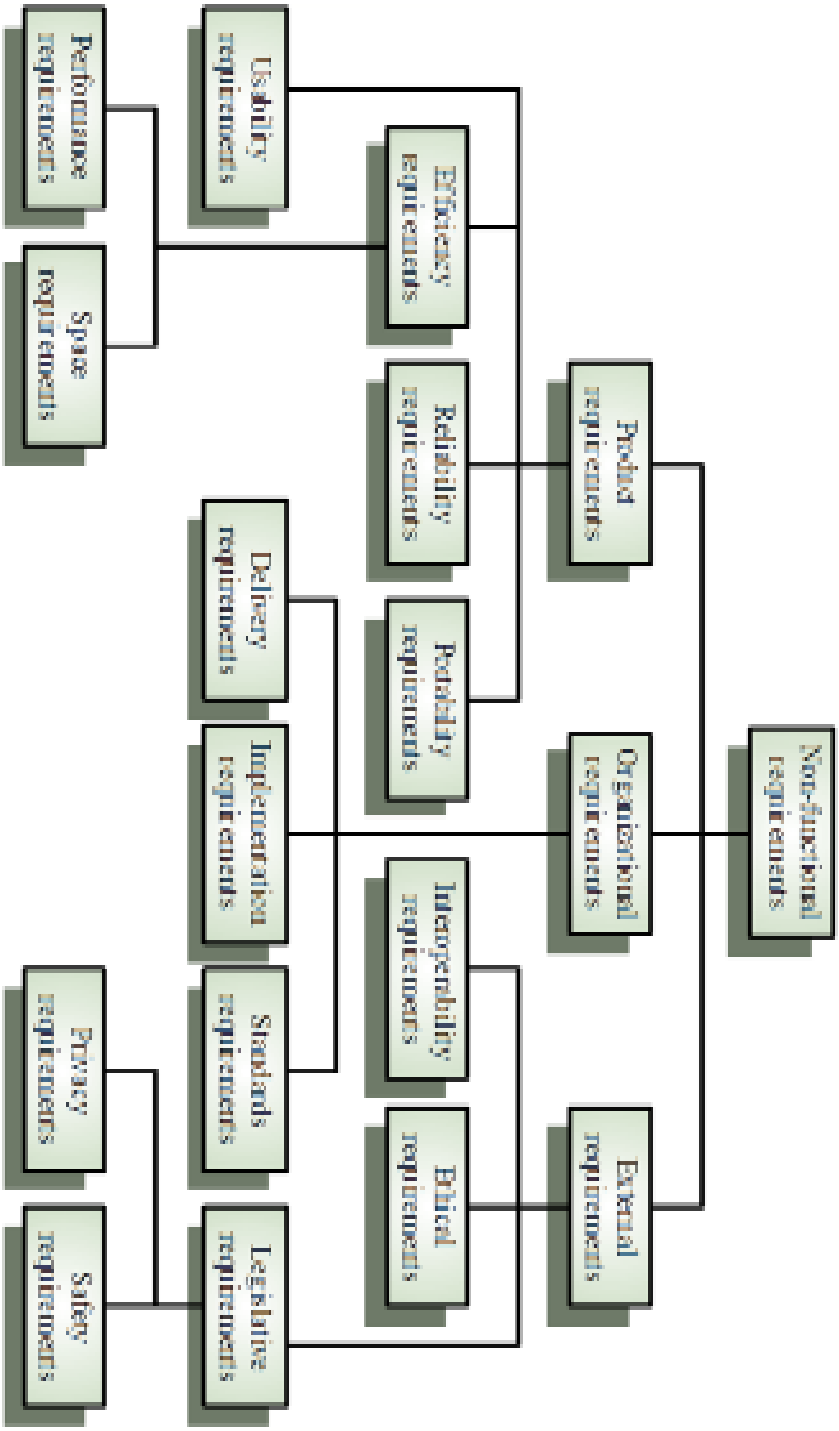
- Act:
  Op basis van input uit de 'check' fase wordt sturing gegeven en actie ondernomen.

Figuur 44 PDCA cyclus (bron: www.pdcacyclus.nl

- Q.A.:
  Quality Assurance (Q.A) zorgt ervoor dat behaalde verbeteringen worden geborgd en niet wordt teruggevallen op het oude gedrag.

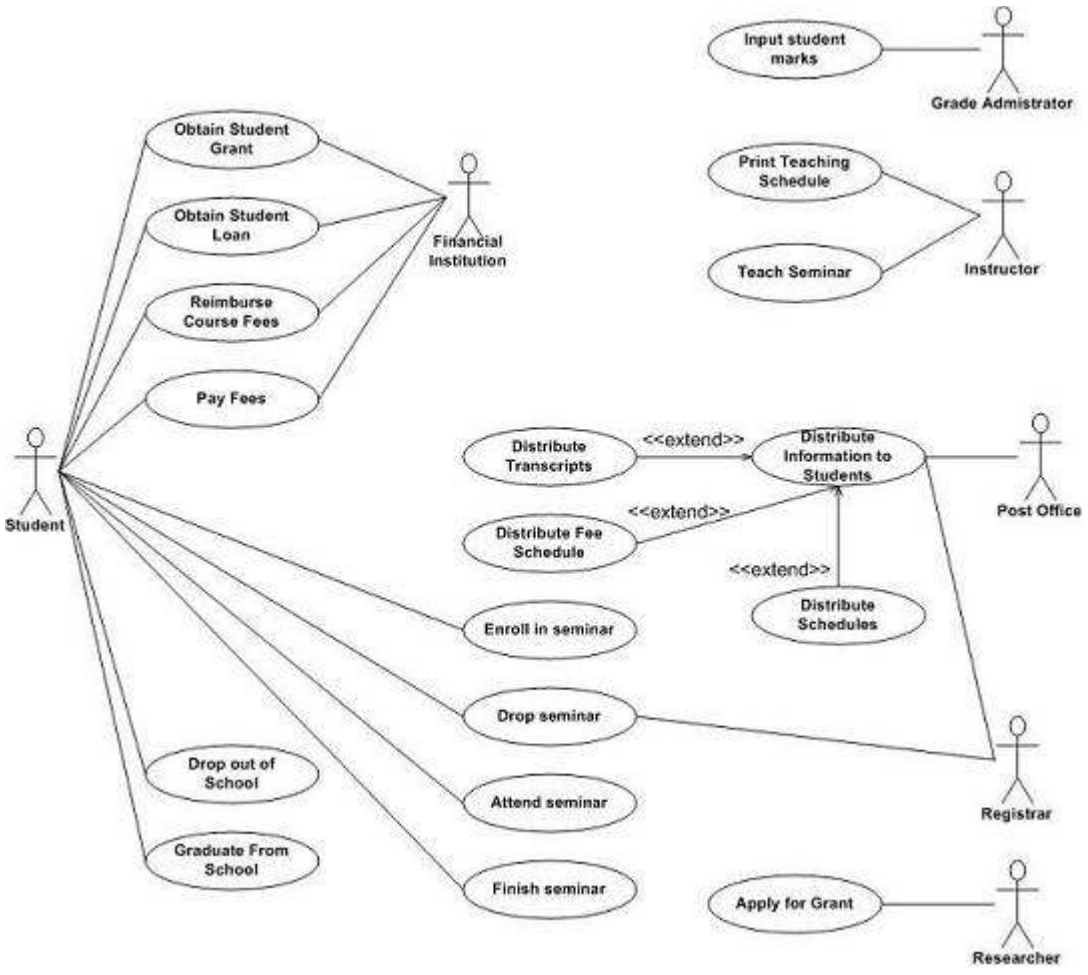
Bijlage 5: Non-functional requirements

De volgende figuur toont diverse types non-functional requirements:



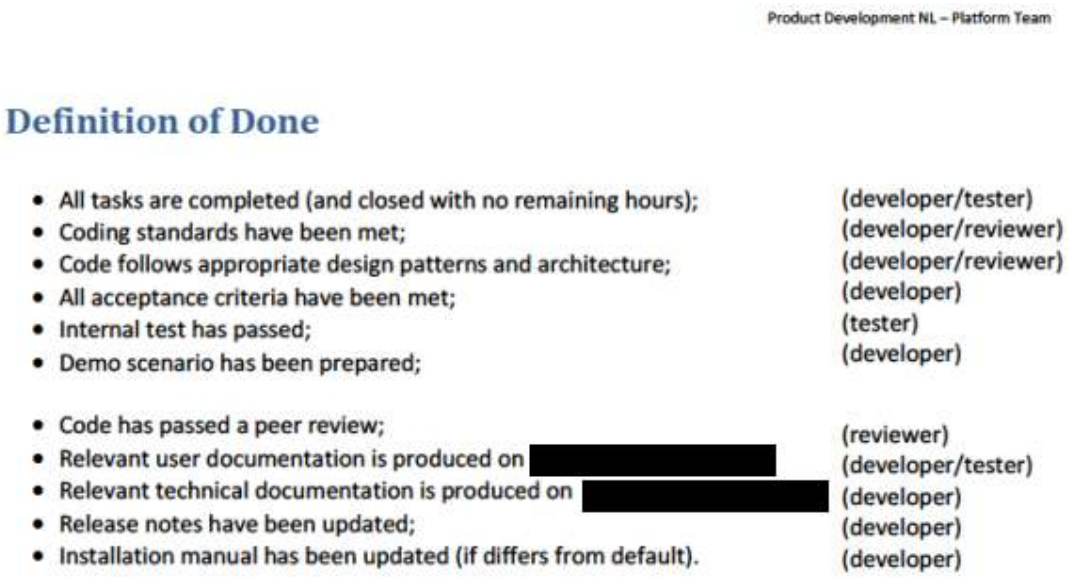
Figuur 45 Non-functional requirements (Bron: Ian Sommerville)

Bijlage 6: Use Case Diagram



Figuur 46 Voorbeeld - Use Case Diagram (bron: www.agilemodeling.com)

Bijlage 7: Definition of Done



Product Development NL – Platform Team

Definition of Done

• All tasks are completed (and closed with no remaining hours);	(developer/tester)
• Coding standards have been met;	(developer/reviewer)
• Code follows appropriate design patterns and architecture;	(developer/reviewer)
• All acceptance criteria have been met;	(developer)
• Internal test has passed;	(tester)
• Demo scenario has been prepared;	(developer)
• Code has passed a peer review;	(reviewer)
• Relevant user documentation is produced on [redacted]	(developer/tester)
• Relevant technical documentation is produced on [redacted]	(developer)
• Release notes have been updated;	(developer)
• Installation manual has been updated (if differs from default).	(developer)

Figuur 47 Voorbeeld Definition of Done