Radboud Universiteit Nijmegen

REQUIREMENTS SPECIFICATIE IN SOFTWARE-ONTWIKKELPROCESSEN

Een beslissingsondersteunend model ten behoeve van de te kiezen requirements-specificatie-aanpakken in software-ontwikkelprocessen

MASTERSCRIPTIE INFORMATIEKUNDE
RADBOUD UNIVERSITEIT NIJMEGEN

S. VAN OOSTENBRUGGE

Titel:	Requirements specificatie in software-ontwikkelprocessen
Ondertitel:	Een beslissingsondersteunend model ten behoeve van de te kiezen requirements-specificatie-aanpakken in software-ontwikkelprocessen
Datum:	oktober, 2014
Plaats:	Nijmegen
Onderwijsinstelling:	Radboud Universiteit Nijmegen
Faculteit:	Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Opleiding:	Informatiekunde
Master:	Informatiekunde
Scriptie:	Masterscriptie
Studentennummer:	4159713
Naam:	Sjoerd van Oostenbrugge
E-mailadres:	svan.oostenbrugge@student.ru.nl

Naam eerste begeleider:	Dhr. Dr. S.J.B.A. Hoppenbrouwers
Naam tweede begeleider:	Dhr. Dr. P. van Bommel

Begeleiders Info Support:

Opdrachtgever	Dhr. H. Nieboer
Technisch begeleider	Dhr. S. Jansen
Procesbegeleider	Mevr. P. Hijl

Radboud Universiteit Nijmegen

SAMENVATTING

De samenleving digitaliseert in een snel tempo, door innovatie en toenemende concurrentie willen organisaties en bedrijven steeds sneller over nieuwe software kunnen beschikken. Dit, in combinatie met het snel in willen kunnen spelen op marktveranderingen, heeft er toe geleid dat afgelopen decennia agile ontwikkelmethodieken steeds populairder zijn geworden. In agile ontwikkelprocessen wordt er, in tegenstelling tot de traditionele ontwikkelmethodieken, minder tijd besteedt aan het requirements engineering proces en is de rol van requirements fundamenteel anders. Dit resulteert in het feit dat veel bedrijven moeite hebben met het beheren en specificeren van de requirements in een project. In deze scriptie is onderzocht welke mogelijkheden er zijn om de requirements te specificeren en te beheren in een software-ontwikkelingsproject.

Aan de hand van de transitie, van traditionele ontwikkelprocessen naar de hedendaagse ontwikkelprocessen, is de rol van de requirements beschreven en zijn de keuzefactoren bij het kiezen van de juiste aanpak geïdentificeerd. Dit resulteerde, in samenspraak met experts uit de praktijk, in de creatie van een beslissingsondersteunend model voor de mogelijk te kiezen requirements-specificatie-aanpakken in software-ontwikkelingsprocessen. Het model is, aan de hand van casestudies en beoordelingsvragen, in de vorm van een digitale vragenlijst voorgelegd aan experts in het vakgebied. Het resultaat van deze scriptie is een beslissingsondersteunend model dat IT-professionals ondersteunt bij de maken keuzes op het gebied van requirements specificatie-aanpakken in software-ontwikkelingsprojecten en dat kan dienen als naslagwerk om meer te weten te komen over de hedendaagse beschikbare requirements specificatie-aanpakken.

VOORWOORD

Voor u ligt het resultaat van mijn afstudeeronderzoek ter afsluiting van de opleiding Master Informatiekunde aan de Radboud Universiteit te Nijmegen. Het schrijven van de scriptie was niet altijd even makkelijk en zonder de hulp van anderen was dit nooit gelukt. Via deze weg wil ik iedereen bedanken die heeft bijgedragen tot het behalen van dit resultaat.

De volgende mensen zou ik graag willen bedanken: Als eerste, dr. S.J.B.A. Hoppenbrouwers, in de rol van begeleider gedurende het schrijven van mijn masterscriptie. Bedankt voor de goede begeleiding, hulp, ideeën en sturing gedurende het traject. Als tweede, dr. P. van Bommel, als tweede-lezer. Ten derde, het bedrijf Info Support B.V., voor het mogelijk maken van dit afstudeeronderzoek. Een speciaal dankwoord gaat uit naar de medewerkers H. Nieboer (opdrachtgever), S. Jansen (technisch begeleider) en P. Hijl (procesbegeleider). Bedankt voor jullie betrokkenheid, sturing, ideeën, feedback, brainstorming en het delen van jullie ervaringen gedurende het traject.

Als laatste wil ik mijn ouders bedanken voor alles wat ze voor mij hebben gedaan tijdens mijn studieperiode. Jullie hebben mij altijd gesteund en daar ben ik erg dankbaar voor.

Veel leesplezier gewenst.

Sjoerd van Oostenbrugge
Veenendaal, september 2014

DISCLAIMER: DIT BETREFT EEN PUBLIEKE VERSIE VAN HET DOCUMENT. DE INTERVIEWS ZIJN ALS VERTROUWELIJK AANGEMERKT EN ZIJN DAAROM IN DEZE VERSIE WEGGELATEN.

INHOUDSOPGAVE

1. INLEIDING

Softwareontwikkeling is uitgegroeid tot een van de belangrijkste technologieën in de wereld, experts zijn zelfs van mening dat de huidige moderne samenleving afhankelijk is van software (Leffingwell, 2010). In bijna elk product dat tegenwoordig in de winkel te koop is, zit een 'eigen' stukje software verwerkt wat het product 'slim' maakt. Elk stukje software dient ontwikkeld te worden voor het doel waarvoor het gebruikt gaat worden, het proces van het tot stand komen van de software wordt softwareontwikkeling genoemd. In dit hoofdstuk wordt het onderzoek ingeleid, worden de onderzoeksvragen besproken en wordt de opbouw van het onderzoek beschreven.

1.1 AANLEIDING EN PROBLEEMSTELLING

Een belangrijk aspect binnen het softwareontwikkelingsproces, is het verkrijgen en vastleggen van de klantwensen voor het systeem. Deze wensen, ook wel requirements genoemd, beschrijven waaraan het product moet gaan voldoen. Het proces van het verkrijgen van de requirements vormt de input[1] voor het daadwerkelijke ontwikkelproces en is daarom essentieel van het ontwikkelen van een product. Waar in de traditionele ontwikkelmethode het verkrijgen en vastleggen van de requirements, voorafgaand aan de realisatiefase, in een aparte fase wordt uitgevoerd, wordt dit proces bij hedendaagse agile ontwikkelmethodieken gezien als onderdeel van het ontwikkelproces. Tevens is er een duidelijk verschil waarneembaar in de mate van detaillering die in de requirements worden aangebracht. In traditionele softwareontwikkeling wordt gestreefd naar een hoge mate van detaillering van de requirements in de requirements-fase, dit resulteert in volledig uitgewerkte requirements voor het te bouwen product. Dit is tegenstrijdig met de principes in de agile methodieken waar, in de requirements, een beperkt niveau van detaillering wordt nastreeft om zodoende meer flexibiliteit te introduceren en *waste*[2] te reduceren. Details in agile ontwikkelmethodieken worden middels het *Just in time en just enough* principe gespecificeerd wat inhoudt dat detail pas wordt toegevoegd indien dit noodzakelijk is in het proces.

Door de behoefte van organisaties om snel en effectief op veranderingen in de markt in te kunnen spelen, om zodoende de *business-value* van het product te verhogen, zijn de laatste jaren agile software-ontwikkelmethodieken sterk in opkomst. Uit recent onderzoek (VersionOne, 2014) blijkt dat wereldwijd in 88 procent van de gevallen een agile methodiek wordt toegepast om software te ontwikkelen. Deze verschuiving op het gebied van software-ontwikkelmethodieken heeft invloed op de manier hoe men met requirements omgaat gedurende het ontwikkelproces. Uit onderzoek blijkt dat veel organisaties moeite hebben met het proces van het vastleggen en het beheren van de requirements in hedendaagse projecten (Cao & Balasubramaniam, 2008). Dit doordat, in tegenstelling tot de traditionele ontwikkelmethodieken, waar exact stond beschreven hoe er met requirements moest worden omgegaan gedurende het proces, er in de agile methodieken geen standaard manier wordt voorgeschreven hoe met requirements om te gaan gedurende het ontwikkelproces.

Het gebrek aan sturing, in combinatie met het vast willen houden aan (oude) gewoontes, leidt er toe dat in veel bedrijven de software wel op een agile manier wordt ontwikkeld, maar de requirements nog op een traditionele manier worden vastgelegd en beheerd. Tevens worden in veel organisaties de requirements op de verkeerde manier vastgelegd en/of beheerd, of worden requirements artefacten ten behoeve van het verkeerde doel gebruikt. Dit leidt tot miscommunicatie, onnodige verspilling van tijd en resources en vermindert de slagingskans van een project.

[1] Traditioneel gezien (Benington, 1956)
[2] Verspilling van tijd

1.2 HET ONDERZOEK

De doelstelling van het onderzoek is om de verschuiving van traditionele ontwikkelprocessen naar agile ontwikkelprocessen, en de rol van requirements in deze processen, in kaart te brengen om uiteindelijk tot een beslissingsondersteunend model te komen waarin de mogelijke aanpakken, ten behoeve van de requirements specificatie in software-ontwikkelprocessen, worden beschreven.

Het model dat in deze scriptie wordt ontworpen is een beslissingsondersteunend systeem, ook wel een *Decision Support System* (DSS) genoemd. Een DSS is een model of een systeem dat kan worden ingezet ter ondersteuning van een te maken keuze, in dit geval het kiezen van de juiste requirements specificatie-aanpak in een project. In tegenstelling tot een beslissingssysteem, ook wel *Decision System* (DS) genoemd, is het doel van het te ontwerpen model niet om op basis van eigenschappen een geschikte aanpak voor te schrijven, maar de gebruiker te ondersteunen bij het kiezen van een geschikte requirements specificatie-aanpak in een project. Het beslissingsondersteunend model wordt niet gerealiseerd in de vorm van een interactieve softwareapplicatie, maar in de vorm van een 'platte' beschrijving van de mogelijke aanpakken, om deze reden wordt er daarom ook gesproken over een model in plaats van een systeem.

Het beslissingsondersteunende model, voor het kiezen van de juiste requirements specificatie-aanpakken in software-ontwikkelingsprocessen, kent twee toepassingen:

1. Het ondersteunen van IT-professionals, in het bijzonder projectmanagers en requirements-engineers, bij de te maken keuzes op het gebied van requirements specificatie-aanpakken in software-ontwikkelingsprojecten.
2. De in het model beschreven requirements specificatie-aanpakken dienen als naslagwerk om meer te weten te komen over de hedendaagse beschikbare requirements specificatie-aanpakken, dit om bijvoorbeeld gemaakte keuzes te toetsen of risico's in een gekozen aanpak te identificeren.

Om aan het eind van het onderzoek tot het beslissingsondersteunende model te komen, zijn de volgende onderzoeksvragen gedefinieerd:

HOOFDVRAAG

Welke requirements specificatie-aanpakken, en bijbehorende requirements artefacten, kunnen ten behoeve van welke doel en in welke (project)situatie worden toegepast in software-ontwikkelingsprocessen?

DEELVRAAG

1. *Wat zijn requirements en waarvoor worden requirements gebruikt in de softwareontwikkeling?*
2. *Welke type ontwikkelprocessen zijn er, en welke rol spelen requirements in deze ontwikkelprocessen?*
3. *Wat betekent de transitie, van de traditionele ontwikkelmethodieken naar de agile ontwikkelmethodieken, voor het ontwikkelproces en voor het gebruik van requirements binnen de softwareontwikkeling?*
4. *Welke factoren zijn van belang bij de keuze van de juiste requirements specificatie-aanpak in software-ontwikkelprocessen en welke invloed heeft een aanpak op de uiteindelijke kwaliteit van het product?*
5. *Hoe moet het beslissingsondersteunend model, ter ondersteuning van de te kiezen requirements-specificatie-aanpakken in software-ontwikkelprocessen, worden geconstrueerd?*

1.3 LEESWIJZER

De opbouw van deze scriptie is als volgt: In hoofdstuk 2 wordt er een korte introductie gegeven op het gebied van softwareontwikkeling en requirements engineering. In hoofdstuk 3 wordt er dieper ingegaan op de verschillende type processen om software te ontwikkelen, hierin zal de transitie van traditionele naar hedendaagse ontwikkelmethodieken worden toegelicht. Vervolgens wordt er in hoofdstuk 4 beschreven welke factoren een rol spelen bij het kiezen van de juiste requirements specificatie-aanpak en wordt het begrip softwarekwaliteit nader toegelicht.

Hoofdstuk 5 beschrijft het onderzoeksmodel, beschrijft de verschillende fasen die gedurende het onderzoek zijn doorlopen en licht de toegepaste onderzoeksvormen toe. In hoofdstuk 6 wordt het beslissingsondersteunende model daadwerkelijk geconstrueerd en worden de achterliggende concepten en technieken nader toegelicht.

Vervolgens worden in hoofdstuk 7 en in hoofdstuk 8 de resultaten van het onderzoek besproken en worden in de conclusie de deelvragen beantwoordt. Ten slot worden de resultaten in hoofdstuk 10 uitvoerig besproken en zullen er aanbevelingen worden gedaan voor eventueel vervolgonderzoek.

2. SOFTWARE ONTWIKKELING IN CONTEXT

In dit hoofdstuk wordt een introductie gegeven op het gebied van softwareontwikkeling, het proces van het verkrijgen van de requirements en de rol van requirements gedurende de ontwikkeling van een product.

2.1 SOFTWARE-ONTWIKKELPROCESSEN

Een software-ontwikkelproces is een verzameling van activiteiten die resulteren in de productie van een software product (Sommerville, 2011). De stappen en procedures die gerelateerd zijn aan softwareontwikkeling kunnen zowel een zelf gedefinieerd pad volgen, of meer universeel gedefinieerd pad.

Er bestaan veel verschillende soorten processen voor het ontwikkelen van software, toch bevatten al deze processen de volgende vier fundamentele activiteiten (Kulak & Guiney, 2003):

1. ***Software specificatie**:* De functionaliteit van de software en de omgevingsbeperkingen worden vastgelegd.
2. ***Software ontwerp en implementatie**:* De software wordt op basis van de specificaties ontworpen of geproduceerd.
3. ***Software validatie**:* De software wordt gevalideerd om ervoor te zorgen dat het functioneert zoals de klant het voor ogen heeft.
   Software evolutie: De software evolueert aan de hand wijziging in omgevingsvariabelen en/wijziging in behoefte van de klant.

De activiteiten, zoals hierboven beschreven, zijn geen fasen in een ontwikkelproces maar zijn activiteiten die gedurende een ontwikkelproces uitgevoerd dienen te worden. Het ontwikkelproces zelf kan dus op een andere manier gestructureerd worden middels fasen, afhankelijk van het type ontwikkelproces. De aandacht dat een team geeft aan elke activiteit bepaald de richting en de kwaliteit van het uiteindelijke systeem. Indien één of meerdere activiteiten onvoldoende aandacht krijgen, zal dit resulteren in problemen in het project en in het uiteindelijke product (Kulak & Guiney, 2003).

Activiteiten in een het ontwikkelproces kunnen erg complex zijn, om deze reden zijn er vaak ook meerdere disciplines betrokken bij het ontwikkelen van de software. Een voorbeeld van een discipline kan zijn: functioneel analist, architect, ontwikkelaar of tester. Deze disciplines kunnen vertegenwoordigd worden door steeds andere (teams) specialisten, maar kunnen ook vertegenwoordigd worden middels multidisciplinaire teams (Cockburn, 2002). In de komende hoofdstukken zal dit verschil in type proces nader worden toegelicht.

2.2 SOFTWARE REQUIREMENTS

Een software requirement is ‘iets’ dat een systeem moet kunnen uitvoeren voor de gebruikers (Kulak & Guiney, 2003). Een requirement beschrijft dus een specifieke functionaliteit, principe of een kwaliteitseis dat het systeem moet bevatten. Tevens kunnen requirements ook worden gebruikt om de interactie tussen het systeem en de gebruiker vast te leggen. De requirements gezamenlijk vormen de scope van een software-ontwikkelingsproject.

Een requirement kan vele vormen aannemen, daarom wordt er onderscheid gemaakt tussen verschillende soorten requirements. Op het hoogste niveau worden requirements onderverdeeld in twee categorieën:

> ***Functionele requirements***: Functionele requirements zijn eisen die beschrijven welke functionaliteiten het systeem moet bevatten, hoe het systeem moet reageren op een bepaalde invoer en hoe het systeem zich moet gedragen in bepaalde situaties.
>
> ***Niet -functionele requirements***: Niet-functionele requirements zijn eisen die worden gesteld aan de eigenschappen van het systeem. Voorbeelden van dergelijken eigenschappen zijn: betrouwbaarheid, gebruiksvriendelijkheid of de performance van het systeem.

Naast het onderscheid tussen functionele en niet-functionele requirements, kan er ook onderscheid worden gemaakt in de verschillende stadia waarin een requirement zich bevindt met betrekking tot de gedetailleerdheid en het doel van de requirement. De drie belangrijkste stadia zijn (Kleine Staarman, 2013):

> ***Gebruiker requirements***: Gebruiker requirements zijn eisen aan het systeem die in de eerste fasen van het project, vaak middels natuurlijke taal, worden beschreven. Deze gebruiker requirements bevatten de functionele en niet-functionele eisen van het systeem vanuit de visie van de klant of gebruiker. Het doel van deze *high-level* requirements is voornamelijk het ondersteunen van de communicatie over de functionaliteiten van het toekomstige systeem.
>
> ***Systeem requirements:*** Gebruiker requirements bieden niet de juiste mate van gedetailleerdheid om direct te kunnen worden omgezet naar de technische implementatie van een functionaliteit. De gebruiker requirements worden daarom uitgewerkt naar systeem requirements, deze bevatten een meer technische beschrijving van wat het systeem moet kunnen. Systeem requirements houden rekening met de meer technische opbouw van het systeem, afhankelijkheden met externe systemen en andere externe geldende beperkingen.
>
> ***Ontwerp specificatie:*** Nadat de requirements op gebruiker en systeem niveau zijn beschreven, worden deze, middels de ontwerp specificatie, vertaald naar een technisch ontwerp. In de ontwerp specificatie worden de requirements verder geanalyseerd en wordt er beschreven hoe de desbetreffende het beste (technisch) geïmplementeerd kan worden.

Requirements engineering

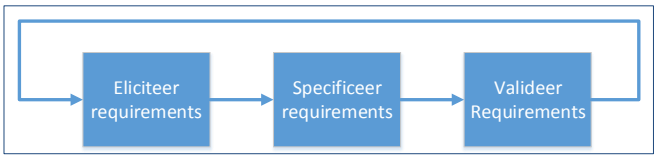
Requirements engineering is het proces van de totstandkoming, documenteren en onderhouden van de requirements. Het doel van requirements engineering is om het gat, tussen de oorspronkelijke behoefte van een klant of de gebruiker en de uiteindelijk gerealiseerde oplossing, te overbruggen (Nehru, Jagityal, & Munassar, 2010) .

Verschillende organisaties en experts hebben de afgelopen decennia het proces van requirements engineering op verschillende manieren proberen vorm te geven, deze aanpakken verschillen in de level van gedetailleerdheid en formaliteit. Alle type requirements engineering processen bevatten de volgende drie stappen:

> ***Eliciteren:*** Het eliciteren van de requirements is de eerste stap in het requirements engineering proces en behelst het verkrijgen en het ophelderen van informatie. Het doel van dit deelproces is het achterhalen van de exacte wensen met betrekking tot het systeem om zo uiteindelijk aan de hand van deze wensen het systeem te realiseren.
>
> ***Specificatie:*** In de specificatiestap worden de eisen, verkregen in het elicitatieproces, geanalyseerd en op een (gestructureerde) manier vastgelegd. Het doel van dit deelproces is het overbrengen van de klantwensen naar de ontwikkelaars. Tevens dient de specificatie ervoor om de verkregen wensen (uit de elicitatieproces) met de stakeholders te evalueren en te valideren.
>
> ***Validatie:*** In het laatste deelproces worden de requirements gevalideerd. Het doel van deze validatiestap kan zijn om goedkeuring te ontvangen van de opdrachtgever of om de requirements op juistheid en/of correctheid te controleren.

Het correct en zo precies helder mogelijk specificeren van de requirements is van essentieel belang gedurende een ontwikkelproces. Traditioneel gezien is het, naarmate het project vordert, steeds kostbaarder om wijzigingen in requirements door te voeren. In hedendaagse projecten wordt er, mede door deze reden, op een andere manier omgegaan met wijzigingen in de requirements. In de komende hoofdstukken worden de voor- en- nadelen van beide aanpakken uitgebreid besproken.

Het requirements engineering proces kan, afhankelijk van de situatie, op verschillende manieren worden doorlopen. Zo kan het requirements engineering proces lineair worden doorlopen, maar ook iteratief of incrementeel. De verschillende manieren om een proces te doorlopen, worden later in de scriptie behandeld. In figuur 1 is het lineair requirements engineering proces geïllustreerd:



FIGUUR 1: HET LINIAIR REQUIREMENTS ENGINEERING PROCES

Het vastleggen en beheren van de requirements, in het specificatieproces, gebeurt met behulp van requirements artefacten. Wat requirements artefacten zijn en waarvoor ze worden gebruikt, wordt uitgelegd in de onderstaande sectie.

REQUIREMENTS ARTEFACTEN

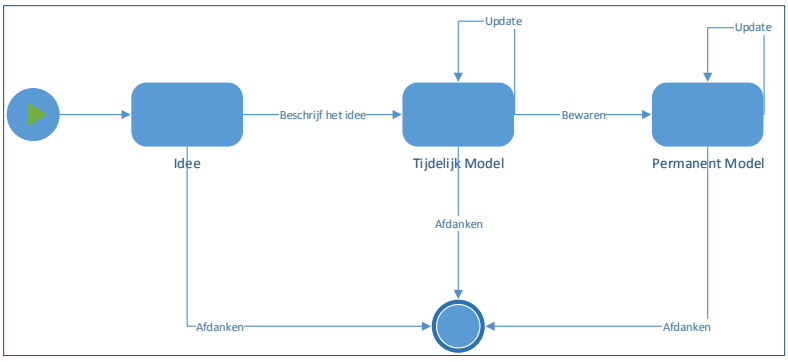
Een requirements artefact is een permanent of tijdelijk product dat wordt geproduceerd, of wordt gebruikt, tijdens een software ontwikkelingsproces. Requirements artefacten worden gebruikt om informatie vast te leggen en/of vastgelegde informatie over te kunnen dragen. Een requirements artefact kan de volgende vormen aannemen (University of Houston, 2012):

> ***Document:*** Een document zoals een *business case* of een software architectuur document.
>
> ***Model***: Een model zoals een *use case model* of een database-model.

Een artefact dient niet alleen ter specificatie van de wensen van de klant maar kan ook dienen ter ondersteuning voor het gebruik en ter validatie van het opgeleverde systeem, de documentatie van en systeem kan daarom ook gezien worden als een artefact (University of Houston, 2012).

Een requirements artefact kan verschillende levensvormen aannemen. Artefacten komen tot stand bij het ontstaan van een idee zoals bijvoorbeeld: '*Hoe willen de gebruikers dat deze functionaliteit werkt?*'. Het idee wordt vervolgens uitgewerkt door gebruik te maken van een requirements artefact. Een requirements artefact kan, afhankelijk van het doel, van tijdelijk of permante aard zijn. Een tijdelijk artefact heeft een tijdelijk doel en wordt, nadat het doel behaald is, niet meer gebruikt en dus weggegooid. Een voorbeeld van een tijdelijk requirements artefact is een schermontwerp, dit artefact ondersteund de ontwikkelaar in de te maken ontwerpkeuzes maar is overbodig nadat het desbetreffende scherm is gerealiseerd.

Een permanent artefact heeft een doel dat van belang is gedurende de gehele levenscyclus van het product en moet daarom ook onderhouden worden. Wijzigt het gekoppelde item, dan zal ook het desbetreffende artefact bijgewerkt moeten worden, dit om de artefacten consistent te houden. Een permanent artefact wordt ook wel documentatie genoemd. Een voorbeeld van een permanent artefact kan, afhankelijk van de projectsituatie, een use case model zijn. De levenscyclus van een requirements artefact is geïllustreerd in figuur 2.



FIGUUR 2: LEVENSCYLUS REQUIREMENTS ARTEFACT (AMBLER S. , 2002)

3. SOFTWARE-ONTWIKKELPROCESSEN

Globaal gezien bestaan er 2 typen softwareprocessen, voorschrijvende en beschrijvende softwareprocessen (Saarnak & Gustafsson, 2003). De traditionele software ontwikkelmethodieken hebben een meer voorschrijvende insteek waarin alle stappen in het proces zijn gespecificeerd, de doelen zijn vastgelegd en het gehele proces is gestructureerd middels fasen. Daartegenover staan de beschrijvende softwareprocessen, dit zijn relatief nieuwe methodieken, zoals *Scrum*, waar de stappen in het proces flexibel zijn en de doelen dynamisch worden bepaald.

In dit hoofdstuk wordt, aan de hand van evolutie van software-ontwikkelprocessen, de verschillende voorschrijvende en beschrijvende software processen beschreven. De softwareprocessen worden onderverdeeld in drie categorieën, gebaseerd op de softwareproces-categorisatie van *Dean Leffingwel* (Leffingwell, 2010). De categorieën zijn:

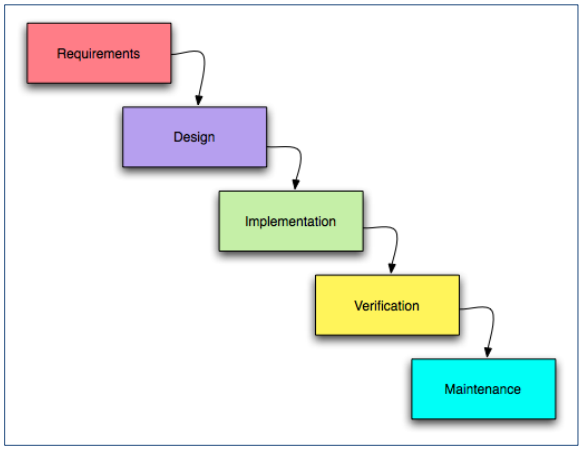
1) Lineaire (sequentiële) processen.
2) Iteratieve en incrementele processen.
3) Adaptieve (agile) Processen.

Per categorie zullen in de onderstaande hoofdstukken de kernprincipes, de gebruikte methodieken en de rol van de requirements in het proces worden toegelicht.

3.1 LINEAIRE (SEQUENTIËLE) PROCESSEN

Sinds de opkomst van de informatietechnologie (IT) worstelen organisaties met het beheersen van software ontwikkelingsprojecten. Producten kosten vele male meer dan oorspronkelijk begroot en worden vele male later opgeleverd dan afgesproken (Benington, 1956). Om meer grip te krijgen op dit proces werd in 1956 door *H.D. Benington* (Benington, 1956) een sequentiële methode geïntroduceerd om software te ontwikkelen. Dit model, later bekend als het waterval model, introduceerde voor het eerst een gestructureerde aanpak voor het ontwikkelen van software.

Binnen deze aanpak worden de volgende stappen doorlopen: Er wordt gestart met het verkrijgen en definiëren van de requirements. De requirements worden vervolgens zeer gedetailleerd uitgewerkt en geaccordeerd door de opdrachtgever. Op basis van deze requirements wordt het systeem volledig ontworpen en vervolgens volledig gerealiseerd, geïmplementeerd en getest. Na elke fasen worden de resultaten uit de voorgaande fasen besproken en geaccordeerd. Het lineaire ontwikkelproces is geïllustreerd in figuur 3.



FIGUUR 3: HET WATERVALMODEL (THE WATERFALL DEVELOPMENT METHODOLOGY, N.D.)

Het is dus belangrijk dat eerst de fase volledig wordt afgerond voordat men start met de volgende fase. Er wordt vanuit gegaan dat de wensen van de klant duidelijk zijn en dat deze middels requirements volledig kunnen worden gespecifieerd. De opgeleverde documenten, aan het eind van elke fase, worden ondertekend

door de opdrachtgever en vervolgens 'bevroren'. Er is geen overlap tussen de fasen en wijzigingen van de requirements zijn *by default* niet mogelijk. De kernprincipes van lineaire softwareontwikkeling zijn als volgt gedefinieerd (Benington, 1956):

> ***Gefaseerde ontwikkeling**:* De gehele ontwikkeling van het systeem wordt gestructureerd middels de voor gedefinieerde fasen. Het proces start vanaf het 'idee' van het te ontwikkelen product, tot en met de uitfasering van het product.
>
> ***Fase review**:* Elke fase eindigt met een review van de opgeleverde documenten die gerelateerd zijn aan de desbetreffende fase. Deze documenten worden ondertekend en vervolgens vindt er een transitie plaats, middels een (formele) overdracht, naar de volgende fase in het proces.
>
> ***Gedetailleerde documentatie**:* Alle activiteiten die gedurende het proces worden uitgevoerd moeten zijn gedocumenteerd. Hoe gedetailleerder een functionaliteit staat beschreven hoe beter de functionaliteit gerealiseerd wordt (Benington, 1956).

Requirements in lineaire processen

Methodieken in de lineaire en sequentiële softwareontwikkeling, zoals het V-Model en het watervalmodel, hebben gemeen dat er van uit wordt gegaan dat alle requirements *Up Front* kunnen worden gespecificeerd. In lineaire en sequentiële ontwikkelprocessen wordt dit gedaan middels requirements artefacten zoals een software requirements document, een functioneel en een technisch ontwerp (Royce, 1970). De reden waarom deze mate van specificatie noodzakelijk is, wordt door *W. Royce* als volgt beschreven (Royce, 1970):

1) ***Communicatie en contract***: Een ontwikkelaar moet kunnen communiceren met de grafisch ontwerper, het management en wellicht ook met de klant. Een verbale requirement voorziet niet in een adequate basis voor een te ontwikkelen functionaliteit of een management beslissing. Een geschreven requirement forceert de betrokkenen tot een eenduidige positie en een tastbaar bewijs na afronding.
2) ***Planning en kostenschatting***: De gespecificeerde functionaliteiten dienen als basis voor de schatting voor de kosten en de tijd dat het gaat kosten om het systeem te ontwerpen.
3) ***Realisatie, implementatie en productie***: Door de uitgebreide specificatie weten de ontwikkelaars exact wat er gerealiseerd moet worden. Tevens zorgt de documentatie ervoor een dat het systeem kan worden onderhouden door een partij die hier gespecialiseerd in is, dit komt de kwaliteit ten goede en is bovendien goedkoper.
4) ***Wijzigingsbeheer***: Doordat het systeem volledig is beschreven kunnen wijzigingen in het systeem makkelijker worden doorgevoerd. Indien de documentatie niet bestaat moet, zelfs voor de kleinste wijziging, de werking van het gehele systeem worden doorlopen.
5) ***Testen***: Met goede documentatie kunnen de testwerkzaamheden worden uitgevoerd door een aparte partij, dit komt ten goede van de kwaliteit van het testen. Zonder uitgebreide documentatie kan alleen het desbetreffende ontwikkelteam de gerealiseerde functionaliteit testen.

Requirements worden in lineaire processen vastgelegd als mogelijkheden en beperkingen van het systeem, de requirements beschrijven dus wat het systeem moet kunnen, of niet moet kunnen. De (meeste) requirements in lineaire en sequentiële processen worden gespecificeerd middels de volgende requirements artefacten:

Software requirements document (functioneel ontwerp)

Het software requirements document is een verzameling van geschreven statements over wat de software moet doen. Het document bevat een beschrijving van de gewenste functionaliteiten, externe koppelingen, niet-functionele eisen als performance en security en ontwerp beperkingen zoals standaarden waar rekening mee gehouden moet worden. De kern van deze documenten is dat er beschreven wordt wat er gerealiseerd dient te worden en niet hoe het gerealiseerd dient te worden.

TECHNISCH ONTWERP (SOFTWARE ARCHITECTUUR DOCUMENT)

Het technisch ontwerp beschrijft hoe de gespecificeerde functionaliteiten in het software requirements document moet worden gerealiseerd. Het technisch ontwerp bevat technische afwegingen, zoals de te kiezen programmeertaal, de te ontwerpen database en andere noodzakelijk informatie voor de ontwikkelaars.

3.2 ITERATIEVE EN INCREMENTELE PROCESSEN

Het watervalmodel, populair in de jaren voor 1980, kwam steeds meer onder druk te staan door de toenemende behoefte aan flexibiliteit, innovaties in software ontwikkeltools en vernieuwde (ontwikkel)technologieën. Dit resulteerde in de behoefte naar een meer innovatieve, ontdekking en resultaat gebaseerde processen, wat uiteindelijk leidde tot de introductie van de iteratieve en incrementele processen begin jaren 80 en 90 (Leffingwell, 2010).

Om te beschrijven wat iteratief en incrementeel ontwikkelen inhoudt, worden eerst de twee begrippen los van elkaar toegelicht:

> **Incrementeel**: Het incrementeel ontwikkelen van software houdt in dat de gewenste functionaliteit wordt opgedeeld in diverse blokken (incrementen) en deze blokken vervolgens één voor één gerealiseerd. Het is nog wel noodzakelijk dat het gehele systeem *Up Front* is gedefinieerd.
>
> **Iteratief**: Het iteratief ontwikkelen van software houdt in dat het systeem niet in blokken, maar in verschillende delen (iteraties) wordt ontwikkeld. Het is dus niet noodzakelijk dat het gehele systeem *Up Front* is gedefinieerd.

In iteratieve en incrementele (I & I) processen worden deze twee type processen gecombineerd tot één proces. In tegenstelling tot lineaire processen, wordt er bij een I & I proces niet gestart met een volledige lijst van gedetailleerde requirements maar worden de requirements, naarmate het proces vordert, steeds vollediger en gedetailleerder. Tevens wordt het systeem opgesplitst in meerdere incrementen die gezamenlijk de complete gewenste functionaliteit bevatten. Elk increment wordt vervolgens ontwikkeld in een iteratie, wat het team in staat stelt om het product eerder te demonstreren en zodoende sneller feedback te kunnen ontvangen van de betrokkenen. Aan de hand van de feedback van de opdrachtgever, kan desgewenst in de volgende iteratie het increment verder verfijnd worden. Dit proces wordt vervolgens herhaald, wat uiteindelijk leidt tot een compleet product (Nehru, Jagityal, & Munassar, 2010).

De kernprincipes van I & I processen zijn als volgt gedefinieerd:

> ***Korte iteraties***: Door de iteraties in het ontwikkelproces wordt er eerder functionaliteit opgeleverd en geïntegreerd, dit resulteert in een snellere leercurve en verminderd het risico (Korsaa, Olesen, & Vinter, 2002).
>
> ***Snelle feedback***: Doordat het systeem iteratief en incrementeel wordt ontwikkeld kan de gerealiseerde functionaliteit sneller worden geverifieerd met de opdrachtgever. Zodoende kan de opdrachtgever sneller feedback geven en bijsturen waar nodig (Korsaa, Olesen, & Vinter, 2002).
>
> ***Communicatie:*** Door de periodieke communicatie wordt de opdrachtgever meer betrokken bij het proces en kunnen beslissingen gezamenlijk worden genomen (Korsaa, Olesen, & Vinter, 2002). Dit resulteert in een functionaliteit die voldoet aan de wensen van de opdrachtgever (Thomas, 2003).

REQUIREMENTS IN ITERATIEVE EN INCREMENTELE PROCESSEN

In iteratieve en incrementele (I & I) processen is er duidelijk een verschuiving waarneembaar van de traditionele requirements *Up Front* methode, naar een meer ontdekking achtige manier van software ontwikkelen (Leffingwell, 2010). De 'zwaargewicht' requirements artefacten, zoals het functioneel ontwerp, worden vervangen door meer 'lichtgewicht' artefacten, zoals het *use case model* en *use cases*. Het voordeel van deze artefacten, is de uitbreidbaarheid gedurende het proces, zodoende kunnen de artefacten, middels het iteratieve karakter van het proces, steeds gedetailleerder worden uitgewerkt (Leffingwell, 2010). In I & I processen (RUP) wordt er nog steeds gebruik gemaakt van een *software requirements document (*zie lineaire processen), maar staan *use cases* centraal binnen het proces, om deze reden wordt RUP ook vaak het *Use Case-driven* proces genoemd. De requirements worden in I & I processen beschreven als interacties tussen de gebruiker en zijn gelijkwaardig, qua mate van gedetailleerdheid, met artefacten in traditionele ontwikkelprocessen. De doelen van de requirements artefacten blijven echter hetzelfde als in de lineaire en sequentiële ontwikkelprocessen. In de onderstaande paragraaf wordt het artefact *use case* verder toegelicht.

USE CASE

Use cases richten zich op de communicatie tussen de gebruiker en het systeem en zijn vanuit gebruikersperspectief geschreven. Met behulp van actoren en events wordt elke stap van de communicatie tussen de gebruiker en het systeem gedefinieerd. Binnen een use case wordt de term *Basic Course of Events* (BCoE) gebruikt, dit is het primaire pad wat gevolgd wordt als alles succesvol verloopt. De extensie en de exceptie sectie beschrijft de alternatieve en het pad wat gevolgd moet worden indien er een fout optreedt in het proces. Elk pad wat mogelijk doorlopen kan worden, wordt een scenario genoemd. Bij het opstellen van Use Cases wordt typisch gebruikgemaakt van een *Use Case Template.* Use Cases kenmerken zich door een vrij gedetailleerde benadering en uitwerking van de user requirements en richten zich met name op gebruikersinteractie (Kleine Staarman, 2013).

3.3 ADAPTIEVE (AGILE) PROCESSEN

De geïntroduceerde methodieken, zoals besproken in de voorgaande hoofdstukken, hebben als doel om softwareontwikkeling voorspelbaarder en efficiënter te maken. Methodieken zoals RUP proberen dit te bewerkstellingen door nadrukkelijk de focus te leggen op requirements en de planning van een project. Ondanks de pogingen om de processen 'minder zwaar' te maken, qua requirements en beslissingsstructuren, worden deze methodieken veelal bekritiseerd om het bureaucratische aspect en het vertragen van de daadwerkelijk softwareontwikkeling (Saarnak & Gustafsson, 2003).

Als een reactie op deze 'zwaargewicht' methodieken werden er gedurende de jaren 1990 een aantal 'lichtgewicht' methodieken bedacht. Vanuit dit perspectief is er een nieuw gedachtegoed ontstaan waarin wordt beweert dat betere resultaten worden bereikt met een aanpak waarbij meer rekening wordt gehouden met het menselijke aspect, meer aandacht is voor het leeraspect, het stimuleren van innovatie en waarbij de *business-value* en het aanpassingsvermogen centraal staan. Vanuit dit gedachtegoed zijn verschillende methoden voortgekomen die met de parapluterm 'Agile' worden aangeduid. Agile is geen methodiek maar een manier van werken.

AGILE MANIFESTO

In februari 2011 kwamen 17 specialisten en wetenschappers bij elkaar in Utah (V.S.) om de nieuwe ontwikkelingen te bespreken op het gebied van softwareontwikkeling en een alternatief te vinden op de 'zwaargewicht' methodieken (Saarnak & Gustafsson, 2003). Centraal stond de vraag: "*Hoe kunnen we het ontwikkelproces beter vormgeven zodat we succesvoller worden in het leveren van software*?". Vanuit deze vraag 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 verschillende disciplines en vakgebieden (Kleine Staarman, 2013).

Er werd overeenstemming bereikt op de volgende vier gebieden (Beck, et al., 2001):

Mensen en hun onderlinge interactie	*boven*	*processen en hulpmiddelen*
Werkende software	*boven*	*allesomvattende documentatie*
Samenwerking met de klant	*boven*	*contractonderhandelingen*
Inspelen op verandering	*boven*	*het volgen van een plan*

Het Agile Manifesto legt de nadruk op mensen in plaats van op het proces. In tegenstelling tot dat elk teamlid aan een 'eigen' stukje software werkt, dwingt agile de teamleden tot gezamenlijke verantwoordelijkheid en intensieve communicatie. Communicatie wordt daarom aangemerkt als belangrijkste factor binnen een agile proces (Cockburn, 2002). Een ander kernpunt uit het *Agile Manifesto* is dat er wordt gestreefd naar het snel opleveren van belangrijke functionaliteit en werkende software middels korte iteraties. Dit omdat alleen een stuk werkende software kan vertellen wat het team daadwerkelijk heeft gebouwd. Documenten worden ter ondersteuning van het proces gebruikt en dienen niet als leidraad voor het proces (Saarnak & Gustafsson, 2003). Het derde statement heeft betrekking op de communicatie tussen het ontwikkelteam en de opdrachtgever. In de Agile werkwijze wordt de opdrachtgever nauw betrokkenen gedurende het ontwikkelproces, zodoende kan de opdrachtgever het project sturen door bijvoorbeeld requirements, indien noodzakelijk, toe te voegen of te wijzigen. Het van te voren vastleggen van requirements en contracten is nadelig voor het uiteindelijke product (Nehru, Jagityal, & Munassar, 2010). Het laatste statement is het inspelen op verandering boven het volgen van een plan. Doordat het systeem iteratief en incrementeel wordt gerealiseerd, kan er continue feedback van de opdrachtgever worden verkregen en wordt het mogelijk gemaakt om snel in te kunnen spelen op wijzigingen, vanuit de opdrachtgever of de markt (Leffingwell, 2010).

REQUIREMENTS IN AGILE PROCESSEN

Door de behoefte om in te kunnen springen op veranderingen, het besef dat alle gewenste informatie niet *Up Front* beschikbaar is en de constante technologische verbeteringen, werd in 2002 het Agile Manifesto gepubliceerd. De filosofie is dat het vooraf gedetailleerd definiëren van de requirements leidt tot een verspilling van tijd en resources (waste) (Cockburn, 2002). In plaats daarvan worden de requirements volgens het '*Just Enough' en 'Just In Time' (JIT)* principe gespecificeerd. Dit houdt in dat er voorafgaand aan het project net genoeg requirements worden gespecificeerd om het project te kunnen starten en de requirements pas (gedetailleerd) worden uitgewerkt als dit in het proces vereist is (Leffingwell, 2010). De volgende requirements artefacten zijn leidend binnen agile processen:

PRODUCT BACKLOG / WORK ITEM LIST:

In agile ontwikkelprocessen wordt er geen gebruik gemaakt van een vooraf gedefinieerde lijst van eisen waaraan het systeem moet voldoen, maar van een op prioriteit geordende lijst, genaamd een *Product Backlog*. De *Product Backlog* introduceert flexibiliteit in het proces, dit doordat de lijst met requirements niet meer 'bevroren' wordt zoals in traditionele processen, maar de samenstelling en ordening van de requirements ten alle tijden kan worden aangepast. Hiermee kunnen de werkzaamheden van het team tussen de iteraties worden bijgesteld en kan de productontwikkeling gedurende het proces worden bijgestuurd.

Kenmerkend aan de requirements op de backlog, is dat de requirements geleidelijk tot stand kunnen komen en niet *Up Front* in detail hoeven te worden uitgewerkt. De Product Backlog bevat dus verschillende niveaus van detaillering, de functionaliteiten met de hoogste prioriteit zijn gedetailleerder gespecificeerd omdat deze functionaliteiten als eerste worden gerealiseerd in een iteratie. Toekomstige functionaliteiten, die wellicht nooit gerealiseerd worden, zijn globaal beschreven. Functionaliteiten die op korte termijn worden ontwikkeld zijn gedetailleerd beschreven (Ambler S. , 2014). De product backlog is een verzameling van type wensen of wijzigingen in het te ontwikkelen product. De product backlog kan verschillende type items bevatten zoals een *user story*, taak of een bug. In de volgende sectie wordt het artefact de user story nader toegelicht.

USER STORY

Een user story is een korte beschrijving van wat een gebruiker wilt dat het systeem kan. Het kenmerk van user stories is dat de requirements op een informele en pragmatische manier worden vastgelegd. User stories worden vaak als volgt geformuleerd: *"Als [rol],zou ik graag willen dat [functie],zodat [doel] wordt bereikt".* Deze samenvattende zin omschrijft zowel de wens, doel, en de betrokkenen stakholder. Een user story bestaat uit de volgende onderdelen (Duka, 2012):

> ***Card:*** Stories zijn traditioneel geschreven op een kaartje, vaak in het formaat van een *post-it*. Op dit kaartje kunnen, indien nodig, details worden toevoegt.
>
> ***Conversatie:*** De conversatie beschrijft alle relevante communicatie met betrekking tot de wens (Kleine Staarman, 2013). Deze details over de te realiseren functionaliteit komt voort uit conversaties met de Product Owner en het team. De benodigde specificatie kan bestaan uit: notities, screenshots, diagrammen of schetsen, specificaties, delen uit e-mails et cetera (Kleine Staarman, 2013).
>
> ***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 (Kleine Staarman, 2013).

User stories zijn ontworpen als startpunt voor de conversatie met het team om te bepalen wat de beste manier is om de gewenste functionaliteit te realiseren. Doordat user stories middels het *Just Enough* principe zijn gedefinieerd, bevatten deze niet genoeg detail om direct te kunnen worden gerealiseerd door een ontwikkelaar. Een belangrijk onderdeel in de detaillering van de user story zijn de acceptatiecriteria (confirmatie), deze definiëren de scope van de user story en bepalen wanneer een user story is afgerond en geclassificeerd kan worden als *done*. De acceptatiecriteria van elke user story worden gedefinieerd in periodieke meetings zoals de *backloog grooming* en de *iteration/sprint planning*. Dit zijn meetings waar het team en de klant bij elkaar komen om onduidelijkheden in wensen weg te nemen en de acceptatie criteria te verduidelijken.

3.4 Conclusie

In dit hoofdstuk worden de ontwikkelprocessen en de rol van requirements, zoals besproken in de voorgaande hoofdstuk, met elkaar vergeleken.

Conclusie t.a.v. het ontwikkelproces

Om het verschil tussen de verschillende type software-ontwikkelprocessen in kaart te brengen worden in deze sectie de verschillende type software-ontwikkelprocessen, aan de hand van een viertal criteria, met elkaar vergeleken. De criteria zijn: '*Omgaan met requirements'*, *Kosten & planning'*, '*Het team'* en '*Testen van de software'* (Saarnak & Gustafsson, 2003).

Omgaan met requirements

In de softwareontwikkeling staat niks vast, de wensen van de stakeholders veranderen vaak gedurende het project en door constante innovatie in de technologieën is de oplossing van vandaag, morgen al achterhaald. Onderzoek heeft uitgewezen dat requirements bij een eenjarig project met gemiddeld 27 procent toenemen, na anderhalf jaar met 43 procent en naar 2 jaar zelfs met 63 procent (Jones, 1995). De redenen hiervoor zijn: '*onjuist gespecificeerde requirements'* en '*gewijzigde behoeften van de business/stakeholders'*.

Valideren van de requirements

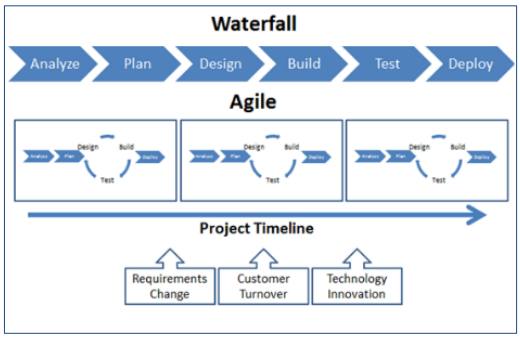
Het is dus essentieel dat de requirements voor het te ontwikkelen product correct gespecificeerd worden, maar hoe kan dit worden afgedwongen? In veel projecten worden de requirements gespecificeerd door een functioneel-analist. Doordat natuurlijke taal ambigu is en de opdrachtgever niet altijd helder voor ogen heeft welke functionaliteit gewenst is, kunnen er foutieve requirements worden opgesteld en/of kunnen requirements gedurende het proces onjuist geïnterpreteerd worden. Het is dus belangrijk dat de requirements worden gevalideerd, het liefst zo vroeg mogelijk in het proces, door de betrokkenen. Hoe eerder een (interpretatie) fout ontdekt wordt, hoe lager de kosten om tijdens het project de fout te verhelpen (Ambler, Scott; Lines, Mark, 2011).

De verschillende type software-ontwikkelprocessen hebben elk hun eigen manier hoe om te gaan met het valideren van de requirements. In het lineaire software-ontwikkelproces worden de requirements *Up Front* gespecificeerd, dit houdt in dat in dat alle requirements in de beginfase van het project worden gespecificeerd. Het volledige requirements-document wordt vervolgens ondertekend door de opdrachtgever, dit om te kunnen garanderen dat het opgeleverde systeem voldoet aan de wensen van de opdrachtgever. In iteratieve en incrementele processen worden de requirements niet meer *Up Front* gespecificeerd, maar gedurende het project. Na elke iteratie wordt er dus gevalideerd of de gerealiseerde functionaliteit voldoet aan de wensen van de stakeholders. In agile processen is het verifiëren van de requirements één van de belangrijkste sub-processen binnen het ontwikkelproces. Middels een speciale rol in het team, bij SCRUM (*Product Owner*), en intensief klantcontact in XP, worden de stakeholders betrokken in het proces en kunnen zodoende direct verifiëren of de gerealiseerde functionaliteit aan de wensen voldoet. Tevens worden in agile processen de functionaliteiten die het meeste waarde hebben, in de ogen van de stakeholders, als eerste gerealiseerd en vervolgens opgeleverd. Dit resulteert, al in de beginfasen van het project, in een (functioneel) werkend systeem.

Omgaan met wijzigingen (flexibiliteit)

Wijziging in de requirements, door voortschrijdend inzicht of wijzigende behoefte van de business, gedurende het project, is in lineaire processen een erg hekel punt (Leffingwell, 2010). Door de sequentiële en strikte fasering is het niet mogelijk om in de realisatie/testfase nog bepaalde functionaliteit aan te passen of functionaliteit toe te voegen. Dit omdat het ontwerp, de planning en het budget is gebaseerd op de oorspronkelijk opgestelde requirements. In iteratieve en incrementele processen wordt er middels het iteratieve proces, steeds opnieuw de requirements fase doorlopen, zodoende kunnen requirements tussentijd worden aangepast of worden toegevoegd. In agile processen worden wijzigingen in requirements juist omarmt. Dit is mogelijk doordat een agile proces geen fasering kent en, elke 2 tot 4 weken in overleg met de stakeholders, besloten wordt welke functionaliteiten er in de desbetreffende iteratie gerealiseerd gaan

worden. Tevens worden de requirements niet op voorhand volledig gespecifieerd maar middels het *'just in time'* principe. Dit voorkomt dat functionaliteiten volledig worden gespecifieerd, en vervolgens in de toekomst niet meer gerealiseerd worden (*waste*). Figuur 4 toont het verschil in 'mogelijke bijstuurmomenten' tussen het watervalproces en het agile-proces.

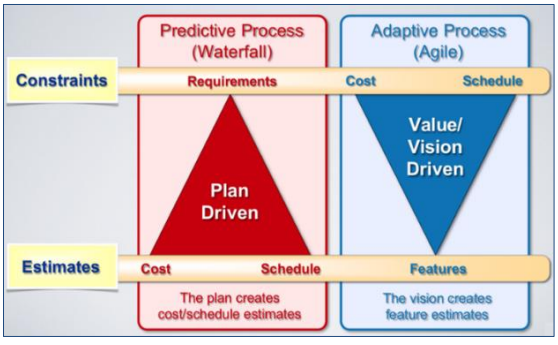


FIGUUR 4: TIJDLIJN AGILE EN WATERVAL MODEL (AGILE ENTERPRISE, IT CONSULTING SOLUTIONS, 2013)

KOSTEN & PLANNING

Uit het alom bekende 'Chaos report' (Standish Group, 1994)[3] blijkt dat 53 procent van de projecten meer kost dan 189 procent van de geschatte kosten. In traditionele ontwikkelprocessen wordt er op basis van de gespecificeerde requirements bepaald hoeveel tijd het team nodig heeft, en uiteindelijk ook hoeveel geld het gaat kosten, om de gewenste functionaliteit te realiseren. Vervolgens wordt er een plan gemaakt en wordt dit plan gedurende het project (exact) gevolgd. Tijdens de realisatie blijkt echter dat het realiseren van de gewenste functionaliteit moeilijker is dan gedacht, of blijkt dat de requirements niet duidelijk gespecificeerd zijn. Om de gewenste functionaliteiten alsnog te realiseren moet dus de planning en het budget worden aangepast, wat resulteert in budgetoverschrijdingen en niet behaalde deadlines.

In agile processen zijn niet de requirements bepalend maar de kosten en tijdspanne van het project. Gegeven de tijd en het beschikbare budget wordt er bepaald welke functionaliteiten binnen de gestelde voorwaarden kunnen worden opgeleverd. Tevens worden in agile projecten de functionaliteiten met de hoogste waarde, ook wel *business value* genoemd, als eerste gerealiseerd. Dus wanneer het project achterloopt op de planning, worden de functionaliteiten met de laagste prioriteit niet gerealiseerd, dit om toch binnen de kostenschatting en de tijdspanne te blijven. De verschuiving van de *plan driven* naar de *value driven* aanpak is geïllustreerd in figuur 5.



FIGUUR 5: VERSCHIL IN 'AANPAK' TUSSEN DE PLAN DRIVEN EN VALUE DRIVEN AANPAK (TOM SYLVESTER, 2013)

Door de *value-driven* aanpak in Agile processen, is het *return on investment* (ROI), in agile processen hoger dan in traditionele ontwikkelprocessen. De redenen hiervan zijn dat in agile projecten de *waste* tot het minimum wordt beperkt en, door de iteratieve oplevering van werkende software, het risico van 'oplevering van onjuiste software' aan het eind van het project wordt verkleind.

[3] Tekortkomingen '*Chaos report*' (Eveleens & Verhoef, 2010)

HET TEAM

In de traditionele softwareontwikkeling zijn 'mensen' een onderdeel van het proces, en worden de 'mensen' niet als constante factor binnen het proces beschouwd. Elk fase in het proces heeft zijn eigen specialisten, zo worden er in de realisatiefase ontwikkelaars ingezet en worden er in de testfase testers ingezet op het project. Dit heeft als voordeel dat de teams afhankelijk van elkaar kunnen opereren maar creëert geen betrokkenheid en samenhorigheid bij de verschillende teams. Documentatie is de primaire bron van communicatie tussen de verschillende disciplines. De kwaliteit van de documentatie speelt daarom een belangrijke rol in de uiteindelijke kwaliteit van het product (Kleine Staarman, 2013). In iteratieve processen heeft nog steeds elk team zijn eigen verantwoordelijkheid maar is er wel meer overlap tussen de verschillende fasen, en dus ook tussen de verschillende disciplines. In agile ontwikkelprocessen wordt er niet meer gesproken over verschillende teams met specialisten. Teamleden zijn multidisciplinair en de nadruk ligt op directe communicatie, in plaats van 'geschreven' documentatie. Om de directe communicatie mogelijk te maken zijn veel agile teams gepositioneerd op één locatie. Het voordeel hiervan is dat de kans dat er miscommunicatie optreedt kleiner is. Tevens bevordert het, doordat er geen hiërarchie bestaat tussen de teamleden en/of de teams, de betrokkenheid van de teamleden (Cao & Balasubramaniam, 2008).

Tevens wordt er in lineaire ontwikkelprocessen pas in de realisatiefase bekend of het team daadwerkelijk in staat is om de gewenste functionaliteit te bouwen. Indien dit niet het geval blijkt te zijn, is de tijd voorafgaand aan de realisatiefase dus verspilde tijd geweest. In iteratieve processen en in agile processen wordt er sneller geverifieerd of het team de gewenste functionaliteiten kan realiseren, dit doordat de software iteratief wordt opgeleverd. Toch kennen iteratieve processen nog een relatieve lange aanloopfase, van analyse en ontwerp, voordat de functionaliteit daadwerkelijk wordt ontwikkeld. In agile processen wordt daarentegen, na een korte start-up fase, direct begonnen met het ontwikkelen van de functionaliteit. Indien het team incapabel is, wordt dit dus in één van de eerste sprints al duidelijk en kan er snel worden bijgestuurd. Door de intensieve communicatie en de inzet van multidisciplinaire teams, kan dit bijsturen zelfs al plaatsvinden tijdens het dagelijkse overleg met het team en/of de iteratie-review dat plaatsvindt na elke iteratie.

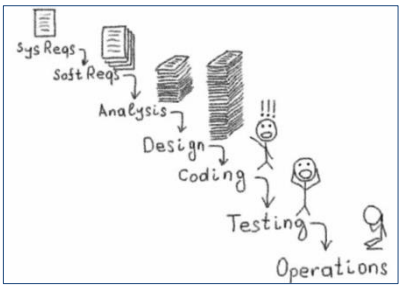
TESTEN VAN DE SOFTWARE

Waar in traditionele softwareontwikkeling het gerealiseerde systeem wordt getest in een aparte fase door een apart team, vinden in iteratieve en agile methodieken de testwerkzaamheden eerder en vaker in het proces plaats. In lineaire ontwikkelprocessen wordt er voornamelijk op een statische manier getest, dit houdt in dat eerst het hele systeem wordt ontwikkeld en vervolgens een apart testteam de gehele applicatie test en de fouten vastlegt in een document. In iteratieve processen wordt na elke iteratie de gerealiseerde functionaliteit getest, dit resulteert in een hogere kwaliteit van de software doordat fouten snel uit het systeem kunnen worden gehaald.

In agile processen wordt de kwaliteit van het systeem gewaarborgd middels de volgende drie principes: '*Constante validatie*', '*dynamische testtechnieken*' en '*gezamenlijke verantwoordelijkheid*'. Doordat er in agile processen veel wordt gecommuniceerd met de betrokkenen, worden de requirements constant geverifieerd middels prototyping en snelle opleveringen. Zodoende kunnen slecht gespecificeerde of foutieve requirements al in een vroeg stadium worden ontdekt. Ook wordt er in agile processen veel gebruikt gemaakt van dynamische testtechnieken, dit zijn technieken zoals *Test-driven development* en acceptatietesten die beide als doel hebben om functionaliteit op te leveren met zo min mogelijk fouten. Een ander groot verschil is dat in agile processen de verantwoordelijkheid van de kwaliteit van de software niet meer ligt bij een apart team, maar de ontwikkelaars zelf verantwoordelijk zijn voor de kwaliteit van de software. Dit wordt in de praktijk gebracht middels *code reviews*, *pair programming* en het constant *refactoren* van de code (Hasmi, 2007).

CONCLUSIE T.A.V. HET GEBRUIK VAN DE REQUIREMENTS (ARTEFACTEN)

In lineaire en sequentiële processen wordt er gebruik gemaakt van uitgebreide specificatie en documentatie om de risico's, voor de ontwikkelende partij, gedurende het project te verkleinen. Door de gewenste functionaliteit gedetailleerd vast te leggen en dit vervolgens te laten ondertekenen door de klant, weten de betrokkenen exact wat er ontwikkeld dient te worden en kan de klant zich achteraf niet beklagen over de gerealiseerde functionaliteit. Requirements in lineaire processen worden vastgelegd als mogelijkheden en beperkingen van het systeem. Door de verschillende disciplines, en dus verschillende teams, die betrokken zijn bij het ontwikkelen van het systeem, dient de uitgebreide documentatie ook ter overdracht en ter kenniswaarborging. De voordelen van de gedetailleerde specificatie in lineaire en sequentiële processen zijn dat de ontwikkelende partij minder risico loopt, dit doordat de gespecificeerde functionaliteiten zijn ondertekend door de opdrachtgever, exact staat beschreven hoe een functionaliteit gerealiseerd dient te worden en de requirements gedurende het proces traceerbaar zijn. Uiteraard kleven aan er ook nadelen aan deze aanpak, zo kan bijvoorbeeld door de vele requirements-documenten belangrijke functionaliteit over het hoofd gezien worden en kan het overzicht verloren gaan. Tevens kunnen de, vanuit het systeem geschreven, requirements de bruikbaarheid van het uiteindelijke product negatief beïnvloeden (Cockburn, 2002).



FIGUUR 6: DOCUMENTATIE IN HET WATERVAL PROCES GEÏLLUSTREERD (VISSER, 2007)

In iteratieve en incrementele processen zijn de hoofddoelen van de requirements artefacten hetzelfde als in lineaire en sequentiële processen. Het grote verschil is echter dat in iteratieve en incrementele processen, bijvoorbeeld in RUP, de requirements niet meer worden gespecificeerd in de vorm van wat het systeem moet kunnen, maar vanuit het gebruikersperspectief middels *use cases*. Het doel van een use case is het beschrijven, middels van een flow, van een functie die een gebruiker kan uitvoeren op het systeem. De voordelen van het gebruik van use cases zijn dat het begrijpelijk is voor alle betrokkenen, het door de gedefinieerde flow meer houvast biedt voor de ontwikkelaars en de use case ook gebruikt kan worden als testcase (BCoE). Het grootste voordeel is echter, ten opzichte van de traditionele requirements artefacten, dat middels een use case alle requirements met betrekking tot een stuk functionaliteit, worden gebundeld. Zo kunnen er bij een use case ook de gerelateerde business rules worden beschreven, dit verhoogd de traceerbaarheid van de requirements (Korsaa, Olesen, & Vinter, 2002).

In agile processen is de rol van requirements artefacten fundamenteel anders dan bij eerder besproken ontwikkelprocessen. Agile methodieken erkennen het feit dat requirements gedurende het project kunnen veranderen. Requirements artefacten hebben niet meer als doel om het risico gedurende het project te verminderen (iteratieve en incrementele processen), of als leidraad (lineaire processen) te fungeren, maar worden alleen gebruikt indien dit ter ondersteuning is van een doel in het proces. In agile processen wordt er nauwelijks meer gebruik gemaakt van deze traditionele requirements artefacten, dit omdat de doelen van deze artefacten totaal niet overeenkomen met de principes uit het Agile Manifesto. Dit omdat de vele requirements artefacten voornamelijk werden gebruikt ter communicatie en ter kennisoverdracht met als doel het verminderen van risico's gedurende het proces. Tevens kunnen de requirements artefacten niet omgaan met de adaptieve en incrementele manier van agile softwareontwikkeling (Hastie & Wick, 2014).

De requirements artefacten die gebruikt worden in agile processen zijn de *product-backlog* en de *user story* (Ambler S. , 2014). De product backlog, ook wel iteratie backlog genoemd, is uitermate geschikt om te gebruiken in agile processen, dit om de volgende redenen: De combinatie van een eenvoudige en een op prioriteit gerangschikte lijst van eisen, gecombineerd met het open karakter van het artefact, bevordert de communicatie met de betrokkenen. Tevens is de product backlog uitermate geschikt om in te kunnen inspelen op marktveranderingen, ondersteunt het iteratieve en incrementele ontwikkeling en geeft het de business de controle over welke functionaliteit als eerst ontwikkeld moet worden (Amrit, et al., 2012).

Alhoewel het hoofddoel van de iteratie backlog hetzelfde lijkt als van een requirements-document, het vastleggen van requirements voor het systeem te bouwen systeem, is het proces van de totstandkoming totaal verschillend. In plaats van dat er maanden wordt geïnvesteerd in het achterhalen en specificeren van requirements, focussen agile teams zich dus op het snel opleveren van waardevolle functionaliteit voor de opdrachtgever (*value driven*) (Ambler S. , 2014). Er wordt dus ook geen gedetailleerde planning en kostenschatting gemaakt aan het begin van het project. De product backlog bevat daarom in eerste instantie alleen maar een lijst met globaal gedefinieerde functionaliteiten en wordt gedurende het proces verder uitgewerkt. Toch brengt het gebruik van een product backlog, ten opzichte van traditionele requirements artefacten, niet alleen maar positieve veranderingen met zich mee, de nadelen van agile requirements engineering worden beschreven in *bijlage 2.*

Gewenste functionaliteiten op de product backlog worden in agile processen beschreven middels een *user story*. Een user story legt de focus op de waarde voor de klant en bevordert, middels de korte beschrijving van de gewenste functionaliteit, de communicatie met de betrokkenen. User stories zijn uitermate geschikt om te gebruiken in agile processen omdat er geen *waste* optreedt bij het specificeren van de requirements en, door de kleinere omvang, gepland kunnen worden in korte iteraties (Suscheck, 2012) (Cohn, 2004). Middels acceptatiecriteria op een user story wordt er gedefinieerd wanneer een functionaliteit als '*done'* kan worden bestempeld. Deze criteria worden gebruikt voor het (automatisch) testen van de gerealiseerde functionaliteit, dit maakt het traditionele artefact, het testplan, overbodig. Toch heeft het gebruik van user stories in agile processen niet alleen voordelen, door de beperkingen van user stories en de constante drive om *user stories* met een zo een beperkte scope te definiëren, kan men het overzicht kwijt raken en kunnen er traceerbaarheidsproblemen optreden. In *bijlage 2* zijn de problemen met requirements engineering in agile ontwikkelprocessen uitgebreid beschreven.

Om de problemen, zoals beschreven in bijlage 2, te verhelpen wordt er in sommige agile projecten gebruik gemaakt van *use cases*, dit in plaats van user stories of in combinatie met user stories (Gröber, 2013) (Jansen, 2014) (Nieboer, 2014). De eigenschappen van use cases komen grotendeels overeen met de agile principes zoals gedefinieerd in het Agile Manifesto, waar mensen en communicatie centraal staan. Use cases ondersteunen namelijk het iteratieve karakter van agile processen doordat ze verschillende mate van volwassenheid kunnen aannemen: '*Façade', 'filled' en 'focussed'* (Kulak & Guiney, 2003). Toch blijkt uit het gedane literatuuronderzoek en de expertinterviews dat use cases *by default* niet gebruikt kunnen worden in agile processen, dit omdat use cases niet geschikt zijn voor het incrementeel opleveren van software. De reden hiervan is dat een use case in de meeste gevallen zoveel functionaliteit bevat dat het niet binnen een iteratie kan worden gerealiseerd (Jacobson, Ivar; Spence, Ian; Bittner, Kurt, 2011) (Jansen, 2014). Ook hebben use cases hebben de neiging om monolithisch te zijn en ondersteunen ze de *'alles of niets'* gedachte, dit is tegenstrijdig met de adaptieve agile manier van denken (Hasmi, 2007). Tevens mist de use case de 'waarom' verantwoording en is het daardoor onmogelijk om de *business-value* van een (gedeeltelijke) use case te bepalen. Dit is problematisch omdat de *business-value* van een te ontwikkelen functionaliteit essentieel is binnen een agile ontwikkelproces (Hastie & Wick, 2014).

4. HET KIEZEN VAN DE 'JUISTE' AANPAK

Zoals eerder beschreven is er de laatste jaren een duidelijke verschuiving waarneembaar van het Up Front definiëren van de requirements naar het Just-In-Time definiëren van de requirements. Toch kunnen er bij de hedendaagse agile werkwijze, zie bijlage 2: *Problemen met requirements Engineering in Agile ontwikkelprocessen*, nog steeds problemen optreden. Het is dus essentieel dat de juiste werkwijze gekozen wordt, de werkwijze heeft immers (indirect) invloed op de uiteindelijke kwaliteit van het product. In dit hoofdstuk wordt er onderzocht wat de relatie is tussen de gekozen aanpak en het uiteindelijke systeem dat gerealiseerd dient te worden. Tevens wordt er onderzocht welke factoren een rol spelen bij de keuze van een juiste (requirements) aanpak en welke invloed deze keuze heeft op de kwaliteit van het product.

De onderstaande definities zijn gebaseerd op technieken uit de vakgebieden en *method engineering* (Amrit, et al., 2012) (Henderson-Sellers & Ralyté, 2010) en het vakgebied *decision support systems building* (Nepomuceno & Fontana, 2013).

4.1 MIJLPALEN EN DOELEN IN EEN SOFTWAREONTWIKKELINGSPROCES

Het hebben van duidelijke mijlpalen in een software ontwikkelproces is essentieel in het wel of niet slagen van het project (Boehm, 1996). Deze mijlpalen in een software ontwikkelproces werden voor het eerst beschreven door *Barry Boehm* in 1996 (Boehm, 1996). Het artikel beschrijft dat het succes van een project afhangt van de mate waarop er met de mijlpalen gedurende het proces wordt omgegaan. Deze mijlpalen, de *Life Cycle Objective* (LCO), de *Life Cycle Architecture (LCA)* en de *Initial Operational Capacity (IOC),* dienen te worden behaald in sequentiële volgorde op vastgestelde tijdstippen in het proces. Doordat de mijlpalen in lineaire volgorde behaald moeten worden, en door de striktheid van te behalen doelen, worden deze mijlpalen vaak geassocieerd met de traditionele manier van software ontwikkelen.

Om meer aan te sluiten op de hedendaagse softwareontwikkelingsprocessen, introduceerde *Alistair Cockburn* in 2013 een nieuwe 'kijk op softwareontwikkeling' genaamd *Disciplined Learning* (Cockurn, 2001). In deze aanpak wordt er gesproken over een viertal vraagstukken die essentieel zijn in een software-ontwikkelproces. Kern van deze aanpak is dat het 'leeraspect' van het team centraal staat gedurende de ontwikkeling. Het (tijdig) beantwoorden van de vraagstukken zal uiteindelijk resulteren in een succesvolle afronding van het project. De essentiële vraagstukken van *Alistair Cockburn* zijn als volgt gedefinieerd:

I. *Hoe kunnen we leren wat er daadwerkelijk gebouwd moet worden?*
II. *Hoe kunnen we leren hoeveel het gaat kosten? (tijd, geld, mensen)*
III. *Hoe kunnen we het team leren om samen te werken?*
IV. *Hoe kunnen we verkeerde aannames in het proces zo snel mogelijk corrigeren?*

Alhoewel deze vraagstukken op het eerste gezicht niet overeen lijken te komen met de mijlpalen zoals gedefinieerd door *Barry Boehm*, kennen deze opvallend veel overeenkomsten. Beide structureringen hebben namelijk als uitgangspunt risico's in het softwareontwikkelingsproces in een zo vroeg mogelijk stadium te elimineren. Het grote verschil is echter dat in agile, en ook in iteratieve processen, de mijlpalen niet op een vast tijdstip moeten worden behaald, maar meerdere keren op variabele tijdstippen (iteraties) in het proces moeten worden behaald. Tevens zijn de criteria niet van toepassing op het gehele systeem, maar op deelfunctionaliteiten (incrementen) van het systeem.

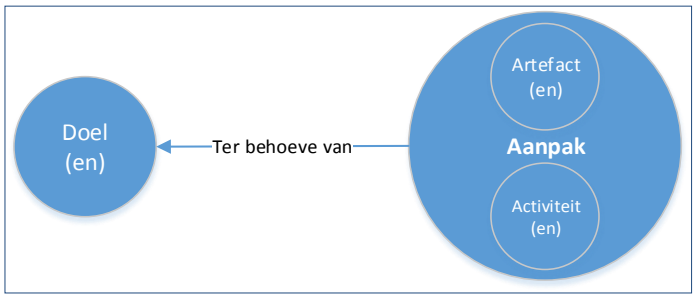
Er kan dus geconcludeerd worden dat men in hedendaagse software-ontwikkelprocessen niet meer spreekt over mijlpalen die op vaste tijdstippen gedurende het proces gehaald bereikt dienen te worden, maar over doelen die gedurende een proces (zo snel mogelijk) behaald dienen te worden. Een doel is een vooraf gedefinieerde doelstelling die behaald dient te worden gedurende een proces.

4.2 De relatie tussen een doel en een requirements specificatie-aanpak

Zoals geconcludeerd in het vorige hoofdstuk, dienen gedurende het software-ontwikkelproces doelen behaald te worden. Om de doelen te behalen, en zodoende de slagingskans van het project te verhogen, dient men de juiste werkwijze te kiezen. Eén van de keuzes die gemaakt moet worden, is de keuze voor een requirements specificatie-aanpak .

Een requirements specificatie-aanpak is de manier waarop men met de requirements om gaat gedurende het software-ontwikkelproces. Wordt er bijvoorbeeld gekozen om alle requirements *Up Front* te definiëren, *Just-in-Time* te definiëren of is er geen specificatie noodzakelijk in het desbetreffende project? Een requirements specificatie-aanpak is een verzameling van activiteiten, inclusief bijbehorende requirements-artefacten, ten behoeve van het behalen van een specifiek doel in een ontwikkelproces.

Een requirements-artefact, zie hoofdstuk 2, kan ter ondersteuning dienen of kan een vereiste zijn als input/output bij een specifieke aanpak. De relatie tussen de doelen, aanpakken, artefacten en activiteiten is in figuur 7 geïllustreerd.



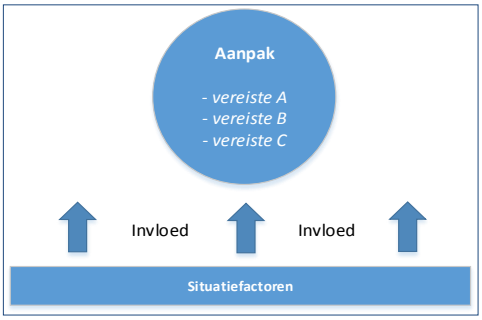
FIGUUR 7: DE RELATIE TUSSEN HET DOEL EN DE AANPAK

Het kiezen, en het gebruik maken van requirements specificatie-aanpakken, is geen doel op zich, maar is een hulpmiddel zijn om een doel te bereiken. De keuze van de juiste requirements specificatie-aanpak is afhankelijk van verschillende factoren, deze worden toegelicht in het volgende hoofdstuk.

4.3 Situatiefactoren en aanpak-vereisten

Zoals eerder beschreven is een aanpak een verzameling van activiteiten ten behoeve van het behalen van een doel. Doordat elk project unieke eigenschappen bevat, moet er worden vastgesteld welke aanpakken er bruikbaar zijn in welke omstandigheden. Om dit te kunnen vaststellen wordt het begrip *situatiefactor* geïntroduceerd. Een situatiefactor is een eigenschap van een fenomeen welke invloed heeft op het software-ontwikkelproces en dus ook op de te kiezen requirements specificatie-aanpak (Amrit, et al., 2012).

Maar ook aanpakken kunnen eisen stellen aan een software-ontwikkelingsproject, niet elke aanpak kan immers (juist) worden toegepast in elk project. Een aanpak kan daarom ook situatiefactoren afdwingen in de vorm van aanpak-vereisten. De relatie tussen situatiefactoren en aanpakvereiste is geïllustreerd in figuur 8.



FIGUUR 8: DE RELATUE TUSSEN DE SITUATIEFACTOREN EN DE AANPAK

De keuze van de juiste van requirements specificatie-aanpak kan worden vergeleken met de keuze voor de juiste projectmethodiek, een projectmethodiek bevat immers ook requirements engineering werkwijzen. Strikt gezien zou men dus kunnen concluderen dat de keuze voor een requirements specificatie-aanpak afhankelijk is van de keuze van een softwareontwikkelmethode. Echter laten veel hedendaagse (agile) projectmethodieken de keuze voor een de juiste requirements specificatie-aanpak over aan het team, dit omdat in de praktijk blijkt dat de keuze voor een requirements specificatie-aanpak sterk afhankelijk is van situatiefactoren (Snijder, 2014) (Lukassen, 2014).

Doordat de keuze van een requirements specificatie-aanpak overeenkomsten vertoond met de keuze van softwareontwikkelmethode, kan er gebruik worden gemaakte van bestaande literatuur over welke factoren invloed hebben op de keuze voor een softwareontwikkelmethode (Geambasu, Jianu, & Gavrila, 2011) (Trendowicz & Münch, 2009) (Office of Information Services, 2005). Op basis van de literatuur is er een nieuwe lijst samengesteld van relevante situatiefactoren. Deze situatiefactoren dienen later in het onderzoek als startpunt voor de situatiefactoren in het te ontwerpen model. De relevante situatiefactoren voor het te ontwerpen model staan opgesomd in tabel 1.

Categorie	Factoren
Stakeholder(s)	- Aanwezigheid. - Betrokkenheid. - Beslissingsbevoegdheid.
Team	- Zelfstandig. - Analytisch vermogen. - Discipline. - Ervaring. - Flexibel. - Grootte.
Product	- Grootte. - Levensduur. - Techniek. - Complexiteit. - Mate van 'kritieke systeem'. - Type {nieuwbouw, vervanging, experimenteel}.
Omgeving	- Duidelijkheid initiële requirements. - Release-frequentie. - Positionering betrokkenen. - (Externe) regelgeving. - Markveranderingen. - Beschikbare resources. - Kosten. - Tijd (projectduur).

TABEL 1: DE RELAVANTE SITUATIEFACTOREN

Nu bekend is welke factoren een rol kunnen spelen bij het kiezen van de juiste requirements specificatie-aanpak, kan er op basis van projecteigenschappen een juiste aanpak gekozen worden, Maar wat is nu eigenlijk de invloed van deze keuze op de uiteindelijke kwaliteit van het product? In het volgende hoofdstuk wordt er antwoord gegeven op deze vraag.

4.4 INVLOED VAN EEN AANPAK OP DE KWALITEIT VAN HET PRODUCT

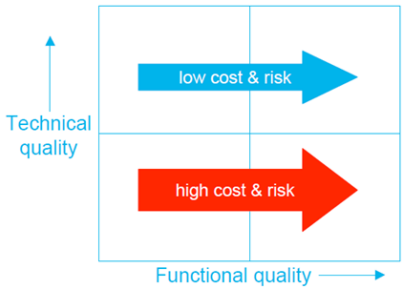
Welke rol spelen requirements in het software-ontwikkelproces en heeft de manier van requirements-management wel invloed op de kwaliteit van de software? Uit onderzoek in het verleden (Standish Group, 1994), maar ook uit recent onderzoek (Brame & Barlow, 2010) (Bloch, Blumberg, & Laartz, 2012), blijkt dat veel IT projecten falen door: '*niet afgerond binnen de vastgestelde tijd en budget*', '*de gewenste functionaliteit werd niet opgeleverd*' of '*de gewenste kwaliteit werd niet opgeleverd*'. Requirements-specificatie-activiteiten hebben dus een grote invloed op de kwaliteit van de software, en dus ook op de uiteindelijke slagingskans van het project. Maar wat wordt er verstaan onder het begrip softwarekwaliteit? Is bijvoorbeeld software waar 99% van de gebruikers 'zeer tevreden' is over het product, van hogere kwaliteit dan software waar 40% van de gebruikers 'zeer tevreden' is over het product? In de onderstaande sectie wordt het begrip nader toegelicht.

WAT IS SOFTWAREKWALITEIT?

Het begrip 'kwaliteit' is binnen de softwarewereld een begrip waar veel over gediscussieerd wordt, welke aspecten bepalen nu eigenlijk de kwaliteit van de software? Kwaliteit, zoals beschreven in het vakgebied kwaliteitsmanagement (Crosby, 1980), is gedefinieerd als: '*conform de eisen*'. Vertaalt naar de softwareontwikkeling zou dit conform de (opgestelde) requirements betekenen. Maar wat als de requirements onvolledig zijn of onjuist zijn gespecificeerd?

Doordat requirements in de softwareontwikkeling niet altijd betrouwbaar zijn, wordt in deze scriptie de definitie van *J.M. Juran's* gebruikt, '*softwarekwaliteit is de mate waarin het product geschikt is voor het gebruik*' (Black, 2002). De kwaliteit van een softwaresysteem houdt dus in dat het product geschikt is voor het doel waarvoor de gebruiker het gebruikt. Een software systeem is geschikt voor het gebruik wanneer: 1) Het overheersend klant-bevredigend gedrag bevat. 2) Het in relatief mindere mate klant-onbevredigend gedrag bevat (Black, 2002).

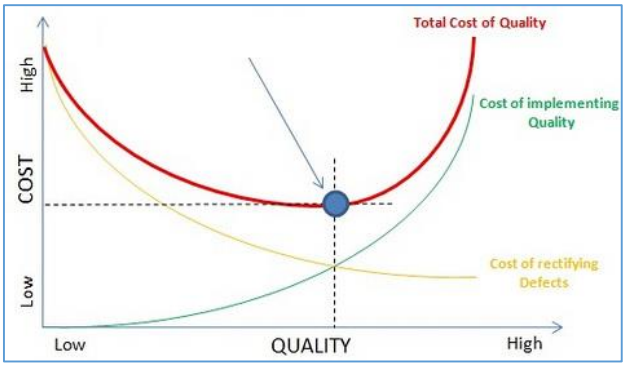
De kwaliteit van de software komt voort uit de combinatie van functionele en technische kwaliteit (Visser, 2007). De functionele kwaliteit van de software heeft betrekking op de functionele kant van het product, bevat het product wel de juiste functionaliteiten? De technische kwaliteit heeft betrekking op het technische aspect van het product, werken de gerealiseerde functionaliteiten wel naar wens? Software van een hoge technische kwaliteit evolueert, om gedurende lange tijd aan de functionele en niet-functionele requirements te voldoen, met een laag risico en lage kosten (Visser, 2007). Software van een lage kwaliteit, evolueert met een hoog risico en daardoor ook hoge kosten. Figuur 9 toont de verhouding tussen de kwaliteit, de kosten en het risico van een product.



FIGUUR 9: VERHOUDING TECHNISCHE KWALITEIT VS FUNCTIONELE KWALITEIT (VISSER, 2007)

Zoals al kort vermeldt, heeft de kwaliteit van software een sterke relatie met de kosten van de software. Ten eerste omdat het waarborgen of het verhogen van de kwaliteit vaak gepaard gaat met stijging in resources en dus uiteindelijk kosten. Alhoewel er geen causaal verband bestaat tussen de kwaliteit van het product en de kosten, wordt bij een lage kwaliteit software vaak de aspecten (te kort aan) tijd, geld of resources gebruikt als excuus. De uitdaging van het ontwikkelen van software ligt dus in het vinden van de juiste balans tussen kwaliteit en investeringskosten gedurende de levenscyclus van een product. Er kan immers heel veel geld worden geïnvesteerd in de ontwikkeling van een product, middels technieken zoals *Test Driven Development, pair programming of* constante gebruikersfeedback, wat (niet vanzelfsprekend) resulteert in een product met

hoge kwaliteit. Echter draait in de hedendaagse maatschappij alles om de *Return-On-Investment* (ROI) van een product. Het doel is dus om de juiste balans te vinden, waarin de investering voldoende is voor het vereiste kwaliteitsniveau voor het gewenste doel (Black, 2002). Figuur 10 illustreert het vinden van de juiste balans tussen de kosten en de kwaliteit, inclusief de kosten van mogelijke defecten, van een product.



FIGUUR 10 : VERHOUDING KOSTEN VS KWALITEIT (RILEY, GEOFF, 2012)

Het doel, van het te construeren model in deze scriptie, is om de requirements specificatie-aanpakken in het software-ontwikkelproces te verbeteren zodat de kwaliteit van uiteindelijke systeem wordt verhoogd.

DE INVLOED VAN EEN REQUIREMENTS SPECIFICATIE-AANPAK

Om de verschillende requirements specificatie-aanpakken later in het onderzoek met elkaar te kunnen vergelijken, wordt het begrip risico geïntroduceerd. De definitie van risico in softwareontwikkeling is: *'factoren die mogelijk kunnen leiden tot fouten en schade in de softwareontwikkeling'*. Gebaseerd op de literatuur zijn de volgende vergelijkingsfactoren vastgesteld (Black, 2002):

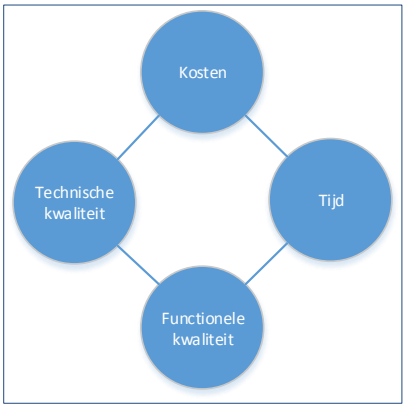
Financieel-risico: Hoe groot is de kans dat het project de begroting overschrijdt? Onder het financiële-risico wordt verstaan dat de gewenste functionaliteit binnen het vastgestelde budget, of een realistisch geschat budget, wordt opgeleverd.

Plannings-risico: Hoe groot is de kans dat het project de toegewezen tijdspanne overschrijdt? Onder het plannings-risico wordt verstaan dat de gewenste functionaliteit binnen een vastgesteld tijd, of een realistisch geschatte periode, wordt opgeleverd.

(Functionele) kwaliteits-risico: Hoe groot is de kans dat het er een verkeerd product wordt opgeleverd? Het bouwen van een verkeerd product is een product dat niet, of gedeeltelijk, de functionaliteiten bevat die gewenst zijn door de gebruiker.

(Technische) kwaliteits-risico: Hoe groot is de kans dat het product klant-onbevredigd gedrag bevat? Klant-onbevredigd gedrag kan optreden als het product niet voldoet aan de niet-functionele eisen die (impliciet) worden gesteld door de klant en de gebruikers. Denk hierbij aan *performance* of beveiligings-problemen van het product.

De verschillende aanpakken, gedefinieerd in het model, zullen door de experts beoordeeld worden op de bovenstaande factoren. Hoe hoger de positieve invloed van een aanpak is op een vergelijkingsfactor, des de kleiner is de kans dat het project faalt. Zoals al eerder beschreven zijn de factoren sterk van elkaar afhankelijk, in figuur 11 is de relatie tussen de bovengenoemde factoren geïllustreerd. Uiteraard is een laag of geen risico, bij alle factoren, geen garantie dat het project succesvol zal verlopen.



FIGUUR 11: RELATIE TUSSEN DE VERGELIJKINGSFACTOREN

5. METHODE

In dit hoofdstuk wordt de gekozen methode beschreven en verantwoord. Tevens wordt er toegelicht hoe het model tot stand is gekomen, hoe de data is verzameld en hoe uiteindelijk de data is geanalyseerd.

5.1 HET ONDERZOEKSMODEL

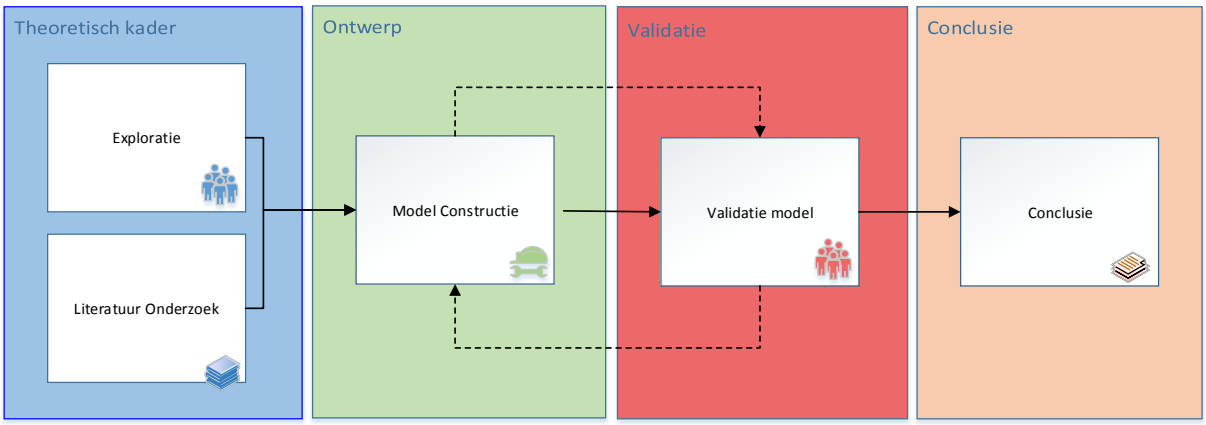
Het type onderzoek, uitgevoerd ten behoeve van deze scriptie is ontwerponderzoek, ook wel *Design Science Research* genoemd. Ontwerponderzoek, zoals beschreven in (Hevner & Chatterjee, 2010), is een onderzoeksparadigma dat als doel heeft om innovatieve (IT) artefacten te creëren om zodoende hedendaagse praktijkproblemen op te lossen. Kenmerkend aan het ontwerponderzoek is dat het de focus op een IT-artefact combineert met een hoge toepassings-relevantie binnen het domein.

In deze scriptie zijn er meerdere dataverzamelingsmethoden, namelijk kwalitatief en kwantitatief, gecombineerd (triangulatie) in één onderzoeksopzet. De reden hiervan is om de bruikbaarheid/toepasbaarheid van het model te vergroten en de geldigheid van het onderzoek te verhogen. Op de eerste plaats is er gekozen voor kwalitatief onderzoek omdat er zodoende dieper op de materie kan worden ingegaan en de mogelijke requirements specificatie-aanpakken, inclusief de beweegredenen, in software-ontwikkelprocessen in kaart kunnen worden gebracht. Op basis van het kwalitatieve onderzoek is het beslissingsondersteunende model tot stand gekomen. Om te kunnen valideren of het model daadwerkelijk bruikbaar en toepasbaar is in de praktijk, is het model kwantitatief getoetst middels een vragenlijst.

Gedurende het onderzoek zijn er verschillende fasen doorlopen, de fasen toegepast in het onderzoek zijn gebaseerd op de cyclussen zoals gedefinieerd in de *Design Science Research*, genaamd *Relevance Cycle*, *Design Cycle* en *Rigor Cycle* (Hevner & Chatterjee, 2010). In het onderstaande hoofdstuk wordt er per fase kort beschreven welke onderzoeksvormen er zijn toegepast.

5.2 FASEN

Het onderzoek kende de volgende fasen: het theoretisch kader, de ontwerpfase, de validatiefase en de conclusiefase. Figuur 12 illustreert de verschillende fasen, de relatie tussen de verschillende fasen en in welke volgorde deze zijn doorlopen.



FIGUUR 12: HET ONDERZOEKSMODEL

De verschillende fasen worden in de komende sectie kort toegelicht.

EXPLORATIE FASE

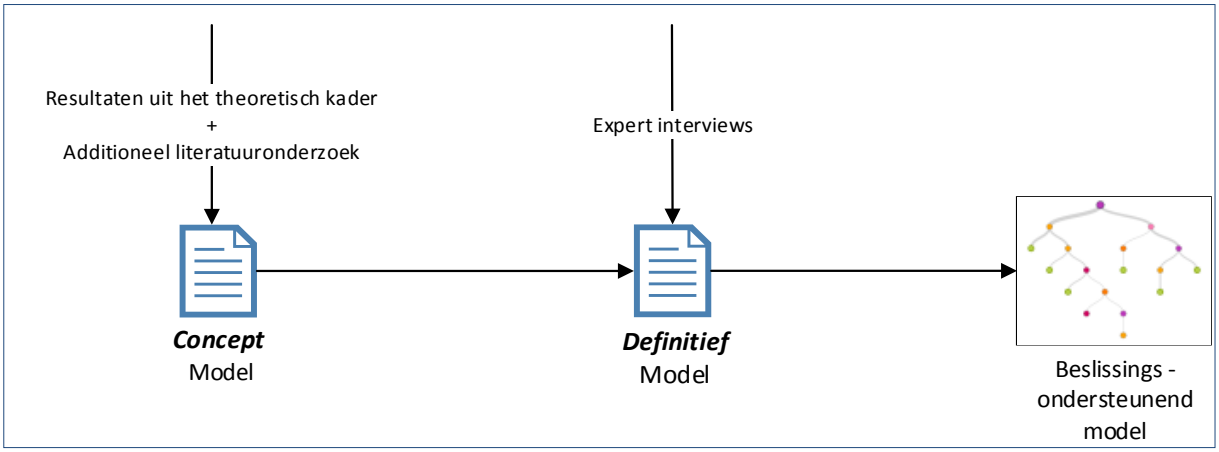
In de exploratiefase is er onderzocht wat het onderzoek exact inhield en wat de eisen en doelstellingen waren vanuit *Info Support*. De exacte doelstellingen zijn achterhaald middels diverse interviews met de opdrachtgever en overige betrokkenen. Tevens is er alvast gestart met het lezen van de relevante literatuur, dit om de materie eigen te maken. Gebaseerd op de expertinterviews en een gedeelte van het literatuuronderzoek, is uiteindelijk het exacte doel, de onderzoeksvraag (inclusief deelvragen) gedefinieerd.

LITERATUURONDERZOEK

Boeken, artikelen, blogs en gesprekken met medewerkers binnen *Info Support* zijn gebruikt om informatie te verzamelen over het onderwerp 'softwareontwikkeling' en 'requirements' binnen het vakgebied. Het literatuuronderzoek is in de basis heel breed opgezet, dit om de context van requirements binnen de softwareontwikkeling te kunnen begrijpen en is, naarmate het onderzoek vorderde, steeds specifieker geworden om zodoende de rol van requirements binnen de softwareontwikkeling gedetailleerd te kunnen onderzoeken. Er is getracht om zoveel mogelijk gebruik te maken van primaire bronnen, artikelen uit vaktijdschriften en monografieën, dit om de kwaliteit en de betrouwbaarheid van het gehele onderzoek te verhogen. Om toch de relatie met de praktijk niet uit het oog te verliezen, zijn de primaire bronnen gecombineerd met secundaire literatuur, tertiaire literatuur en ook met (actuele) blogs van gerenommeerde experts in het veld. De resultaten van het literatuuronderzoek zijn beschreven in het theoretisch kader en zijn samengevat in *hoofdstuk 5*.

MODEL CONSTRUCTIEFASE

In de model constructiefase is, op basis van de resultaten uit het theoretisch kader, het model geconstrueerd. Om het model echt van toegevoegde waarde te laten zijn in de praktijk, is er gekozen om het model in verschillende fasen te construeren en te laten evolueren (*Design Cycle*). Op basis van de resultaten uit het theoretisch kader is er een conceptmodel geconstrueerd. Dit conceptmodel is vervolgens aangevuld met ervaringen uit de praktijk middels interviews met een zestal experts binnen *Info Support*. De reden hiervoor is dat hierdoor het uiteindelijke model zodoende bruikbaarder wordt in de praktijk, dit doordat de experts bij *Info Support* veel praktijkervaring hebben en het kennisniveau binnen *Info Support* erg hoog is[4]. De combinatie van theoretische-kennis en praktijkkennis resulteerde uiteindelijk in het beslissingsondersteunende model. Het proces van de totstandkoming van het model is geïllustreerd in figuur 13.



FIGUUR 13: HET PROCES VAN DE TOTSTANDKOMING VAN HET MODEL

Het expertinterview, ter input van het model, is afgenomen middels een semigestructureerd-interview en had als doel meningen en achtergronden te achterhalen die nog niet bekend zijn in de literatuur. Er is gekozen om gebruik te maken van een semigestructureerd interviewtechniek om aan de ene kant het interview te kunnen sturen zodat alle relevante onderwerpen worden behandeld, maar toch aan de andere kant de expert niet te beperken in de vrijheid om datgene in te brengen wat hij/zij relevant vindt.

[4] Vermoeden, niet gestaafd.

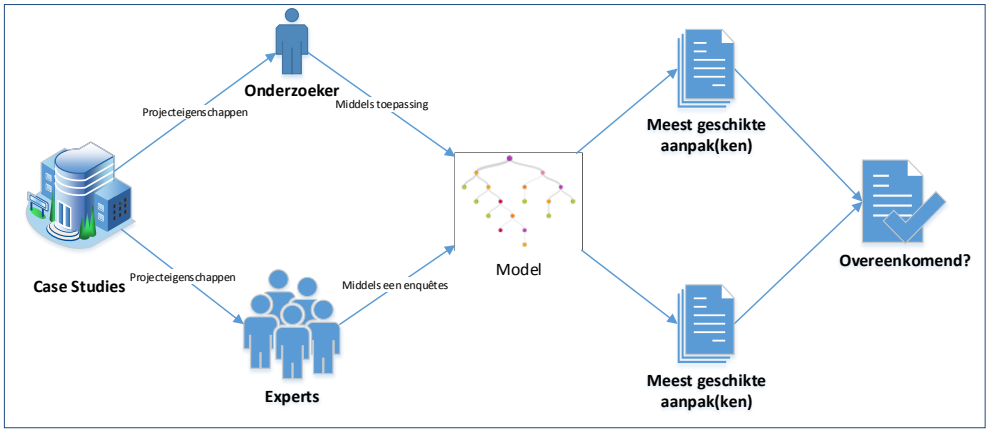
De validatiefase

In de validatiefase is het model gevalideerd. Om te kunnen toetsen of het model toepasbaar is in hedendaagse softwareontwikkelings-projecten, is het model middels de volgende stappen gevalideerd:

Stap 1) Om te valideren of het model kan worden toegepast op projecten uit de praktijk (generaliseerbaarheid) zijn er een drietal casestudies opgesteld. Deze casestudies zijn gebaseerd op lopende/afgeronde projecten van *Info Support* en vertegenwoordigen een zeer groot deel van de projecten die binnen Info Support, en IT bedrijven in het algemeen, worden uitgevoerd. De casestudies bevatten uiteenlopende projecteigenschappen en zijn daarom uitermate geschikt om het model te valideren. Vervolgens is het model door de onderzoeker toegepast op de drie casestudies, dit resulteerde per casestudie in een aanpak die het meest geschikte is voor dat type project.

Stap 2) Om te toetsen of het model correct functioneert, werden de resultaten van *stap 1,* gevalideerd middels een (niet-anonieme) vragenlijst. Er is hiervoor gekozen om zodoende meer mensen te kunnen bereiken, dan bijvoorbeeld met een interview of een groepsdiscussie, en hiermee de validiteit en betrouwbaarheid van het onderzoek te verhogen. De vragenlijst is kwantitatief van aard omdat de vragenlijst toetst of het model toepasbaar en bruikbaar is in de praktijk. De toepasbaarheid van het model is getoetst door de resultaten, van de toepassing van het model op de casestudies (stap 1) voor te leggen aan de respondenten en hier hun mening over te vragen. De bruikbaarheid van het model is getoetst middels een zestal vragen over de bruikbaarheid van het model als naslagwerk. De vragenlijst bevatte tevens open vragen omdat de onderzoeker ook geïnteresseerd is in de achterliggende redenatie van de respondent.

Het validatieproces is geïllustreerd in figuur 14.



FIGUUR 14: HET VALIDATIEPROCES

Conclusie

In de conclusie worden de resultaten, voortkomend uit de validatiefase, besproken en wordt er antwoord gegeven op de onderzoeksvragen.

5.3 Experts & Respondenten

De experts en respondenten in dit onderzoek zijn zorgvuldig geselecteerd. Om het model te construeren en uiteindelijk te valideren is er gebruik gemaakt van de kennis en ervaring van experts binnen *Info Support*. De experts in de constructie en validatiefase zijn geselecteerd op basis van hun specialisme. Info Support kent een achttal *Competence Centers* (CCs), dit zijn groepen waar medewerkers zich kunnen aansluiten om ervaring en kennis uit te wisselen over een specialisme binnen de IT. Omdat het onderwerp van deze scriptie gerelateerd is aan requirements, en projectmethodieken vaak worden gekozen door projectmanagers, zijn de relevante CCs binnen Info Support het *Competence Center Requirements Analysis* (CCRA) en het *Competence Center Projectmanagement* (CCPM).

Constructiefase: Expertinterviews

Het model is mede geconstrueerd middels een zestal interviews met experts uit het CCRA *binnen Info Support.* De experts zijn geselecteerd op basis van de ervaring, kennis van nieuwe technologieën en in welke type projecten de experts werkzaam zijn. De volgende experts zijn geïnterviewd:

Harry Nieboer	*Lid Competence Center Requirements & Analyse*	(Nieboer, 2014)
Joop Snijder	*Product Owner KnowNow*	(Snijder, 2014)
Rene Hietkamp	*Lead Competence Center Requirements & Analyse*	(Hietkamp, 2014)
Stefan Jansen	*Lid Competence Center Requirements & Analyse*	(Jansen, 2014)
Vincent Lukassen	*Lid* Competence Center Projectmanagement	(Lukassen, 2014)
Guido van Loon	*Expert in Test-Driven-Development*	(van Loon, 2014)

Validatiefase: De vragenlijst

Om het model te valideren zijn de resultaten uit de ‘stap 1’ van de validatie, middels een vragenlijst, voorgelegd aan requirements-specialisten en projectmanagers binnen het vakgebied. Binnen *Info Support* is dit gedaan door uit de leden van het CCRA en het CCPM, aselect (random) 40 mensen te trekken. Deze steekproefpopulatie is aangevuld met een drietal projectmanagers en een tweetal requirements-specialisten van *MivarGroup* B.V.[5]. Er is gekozen om de vragenlijst alleen af te nemen onder specialisten op het gebied van requirements en/of projectmanagement, dit omdat de materie niet voor elke IT-professional toegankelijk/begrijpelijk is. Tevens is er gekozen om ook IT-professionals buiten *Info Support* te laten participeren in het onderzoek, dit om de validiteit en de betrouwbaarheid van het onderzoek te verhogen. De reden hiervan is dat het kennisniveau van de IT-professionals binnen *Info Support* hoger ligt dan van de gemiddelde IT-professional[6]. Indien alleen specialisten van Info Support uitgenodigd worden, kan het voorkomen dat de uitkomsten van de vragenlijst niet representatief zijn voor de gehele populatie. De uitnodiging om de vragenlijst in te vullen is verstuurd aan 59 mensen, de vragenlijst was in de periode van 15 augustus 2014 tot en met 26 augustus 14 beschikbaar. Omdat niet verlangd kan worden dat de respondenten de voorgeschreven requirements specificatie-aanpakken van de drie verschillende casestudies gaan valideren, is er in de vragenlijst per respondent (random) 1 casestudie voorgelegd.

[5] http://www.mivargroup.nl/overons/
[6] Subjectief oordeel, gebaseerd op ‘meningen’ uit het vakgebied.

5.4 METHODE VAN DATA-ANALYSE

De volgende methoden van data-analyses zijn gebruikt:

CONSTRUCTIEFASE: EXPERTINTERVIEWS

De interviews met de experts zijn opgenomen met een audio-recorder en achteraf uitgeschreven in een document (transcript), deze kunt u terugvinden in *bijlage 3*. Vervolgens is de relevante informatie, uit de transcripten, gefilterd en is de informatie per doel gecategoriseerd op bruikbaarheid. Als laatste stap is de (nieuwe) informatie geïntegreerd in het model en is het modelontwerp voltooid.

VALIDATIEFASE: VRAGENLIJST

Nadat de invultermijn van de vragenlijst was verlopen, zijn de resultaten geanalyseerd met behulp van *Qualtrics* [7]. Per doel van het model zijn de resultaten geanalyseerd en is de data, waar nodig, in Microsoft Excel[8] gevisualiseerd. De resultaten zijn opgenomen in *hoofdstuk 7*.

[7] http://www.qualtrics.com/about/
[8] http://office.microsoft.com/nl-nl/excel/

6. HET MODELONTWERP

In dit hoofdstuk wordt het beslissingsondersteunend model geconstrueerd en worden de concepten achter het model nader toegelicht.

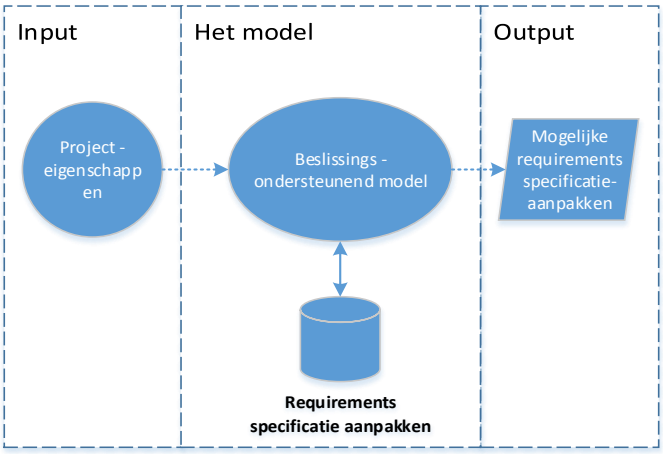
6.1 ACHTERGROND INFORMATIE

Zoals eerder beschreven in de in het hoofdstuk *Scope en structuur,* dient het model een tweetal doelen:

1. *Het model dient ter ondersteuning voor IT-professionals, in het bijzonder projectmanagers en requirements-engineers, voor de te maken keuzes op het gebied van requirements specificatie-aanpakken in software-ontwikkelprocessen.*
2. *Het model dient als naslagwerk om meer te weten te komen over de hedendaagse beschikbare requirements specificatie-aanpakken, dit om bijvoorbeeld gemaakte keuzes te toetsen of risico's in een gekozen aanpak te identificeren.*

De gebruikte concepten in het model zijn gebaseerd op technieken uit de vakgebieden en *method engineering* (Amrit, et al., 2012) (Henderson-Sellers & Ralyté, 2010) en *decision support systems building* (Nepomuceno & Fontana, 2013). De basis van het model is de '*aanpak template*', in deze tabelstructuur staan alle mogelijk te kiezen requirements specificatie-aanpakken met bijbehorende kenmerken beschreven. De verschillende aanpakken vormen gezamenlijk de basis voor het beslissingsondersteunende model. Het model zelf combineert deze aanpakken met de te behalen doelen, die gedurende het project behaald dienen te worden, en geeft per doel de mogelijke requirements specificatie-aanpakken weer.

Indien de gebruiker het model gebruikt, kan de gebruiker aan de hand van de projecteigenschappen, per doel de juiste requirement-specificatie-aanpak kiezen. Wordt het model gebruikt als naslagwerk, dan kan de gebruiker het model gebruiken meer te weten te komen over de hedendaagse beschikbare requirements specificatie-aanpakken. Bij elke aanpak wordt de toepasbaarheid, toepassingseisen en de kwaliteit van de desbetreffende aanpak beschreven. De werking van het beslissingsondersteunende model wordt, indien het wordt gebruikt ter ondersteuning van een te maken keuze, weergegeven in figuur 15.:



FIGUUR 15: DE WERKING VAN HET MODEL

Merk op dat tussen de input, het model en de output een stippellijn wordt weergegeven, dit omdat het matchingsproces, tussen de projecteigenschappen en de mogelijke requirements specificatie-aanpakken, handmatig plaatsvindt. Er is voor een handmatig matchingsproces gekozen, en dus geen beslissingssysteem te ontwerpen, omdat de keuze van een geschikte requirements-specificatie aanpak een keuze is die gemaakt moet worden op basis van (domein) kennis en ervaring. Tevens is de keuze in veel situaties ook afhankelijk van complexe, vaak niet relevante, (externen) factoren en deze buiten de scope van het onderzoek vallen.

6.2 De concepten

In de onderstaande sectie worden de concepten achter het model nader toegelicht.

Doel, activiteit, artefact en aanpak

Uit het literatuuronderzoek is gebleken dat requirements specificatie-aanpakken altijd ten behoeve van een doel worden toegepast. Zoals besproken in het voorgaande hoofdstuk, introduceerde *Alistair Cockurn* een nieuwe 'kijk op softwareontwikkeling' genaamd *Disciplined Learning.* In deze aanpak wordt gesproken over een viertal vraagstukken die essentieel zijn in een software-ontwikkelproces. Om deze vraagstukken te kunnen gebruiken als basis voor het te ontwerpen model, worden deze vertaald naar doelen die gedurende een project gehaald dienen te worden om een project succesvol af te ronden (Cockurn, 2001):

1. *Het bouwen van de gewenste functionaliteit.*
2. *Het juist schatten van de kosten en tijd van het project.*
3. *Achterhalen of dit team de gewenste functionaliteit kan bouwen.*
4. *Het opleveren van een systeem dat functioneert in de praktijk.*

In deze scriptie ligt de focus op het kiezen van de juiste requirements specificatie-aanpakken in software-ontwikkelprocessen, om deze reden wordt de scope daarom beperkt tot het behalen van het eerste doel: '*Het bouwen van de gewenste functionaliteit'*. Omdat dit doel zeer generiek is, wordt dit hoofddoel opgesplitst in meerdere subdoelen. Deze subdoelen dragen direct of indirect bij aan het bouwen van de juiste functionaliteit en zijn afgeleid uit de literatuur en interviews met experts uit de praktijk.

De subdoelen zijn als volgt geformuleerd:

1. *Definiëren van een visie.*
2. *Proces van het definiëren van de requirements vaststellen.*
3. *Beheren en specifiëren van de requirements*
4. *Requirements overzichtelijk houden.*

Om een doel gedurende een ontwikkelproces te behalen, dient men de juiste requirements specificatie-aanpak te kiezen die bijdraagt aan het behalen van het doel.

DE AANPAK TEMPLATE

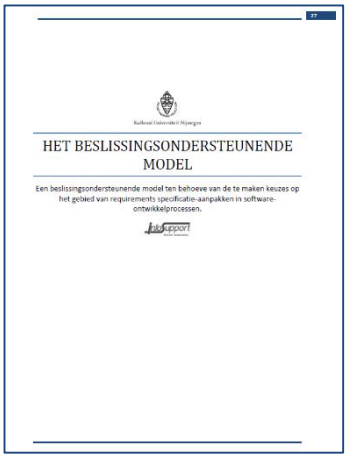
Voordat daadwerkelijk het beslissingsondersteunend model kan worden geconstrueerd , dienen de verschillen soorten requirements specificatie-aanpakken in kaart te worden gebracht. Voor het in kaart brengen van de aanpakken, is de template gebruikt zoals weergegeven in tabel 2. Deze template is gebaseerd op concepten uit het vakgebied *Method Engineering* (Amrit, et al., 2012) en is voor de doeleinden van dit onderzoek aangepast.

Aanpak template	
Eigenschap	***Beschrijving***
Aanpaknaam	*Volledige naam van de aanpak.*
Aanpak Code:	*Code van de aanpak.*
Aanpak Type:	*In welke categorie kan deze aanpak worden ingedeeld? {Lineair (sequentieel), iteratief en incrementeel, Agile, n.v.t.}*
Beschrijving:	*Beschrijving van de aanpak, wat kenmerkt deze aanpak en welke invloed heeft deze aanpak op het ontwikkelproces en dus op het uiteindelijke product?*
Toepasbaar in:	*In welke situaties is deze aanpak 'by default' toepasbaar?*
Toepassingseisen	*Indien deze aanpak wordt toegepast in een project, welke interne en externe factoren zijn dan vereist (situatiefactoren)?*
Artefacten:	*Welke artefacten worden door deze aanpak geproduceerd of gebruikt?*
Kwaliteitsanalyse:	***Financieel:*** *Wat is het resultaat van het toepassen/gebruik van deze aanpak op het gebied van het financiële-risico in het project? Neemt het risico van budgetoverschrijding af door het toepassen van deze aanpak?* ***Planning:*** *Wat is het resultaat van het toepassen van deze aanpak op het gebied van het plannings-risico in het project? Wordt bij het toepassen van deze aanpak de kans, dat het project niet binnen de vastgestelde kaders wordt opgeleverd, kleiner?* ***Functioneel:*** *Wat is het resultaat, indien deze aanpak wordt toegepast, op het gebied van het functionele risico? Zorgt deze aanpak ervoor dat de juiste functionaliteit wordt gebouwd, of resulteert de toepassing van deze aanpak in minder functionele zekerheid?* ***Kwaliteit:*** *Wat is het resultaat van het toepassen van deze aanpak op de uiteindelijk kwaliteit van het product? Zorgt deze aanpak ervoor dat het uiteindelijke product een hogere mate van klant-bevredigend gedrag bevat?*

TABEL 2: DE AANPAK TEMPLATE

6.3 HET MODEL

Zoals al eerder beschreven is het hoofddoel om de gewenste functionaliteit te bouwen voor de klant. Om dit te bereiken zijn er een viertal subdoelen opgesteld. Voor elk van de doelen is er gekeken welke aanpakken kunnen worden toegepast ten behoeve van welk doel. In bijlage 4 is het gehele beslissingsondersteunende model bijgevoegd. Het model bestaat uit een korte introductie van het doel van het model, de beschrijving van de te behalen doelen in een software ontwikkelingsproject en de uitgebreide beschrijving van de mogelijk te kiezen requirements specificatie-aanpakken. Figuur 16 toont het titelblad van bijlage 4, het beslissingsondersteunende model.

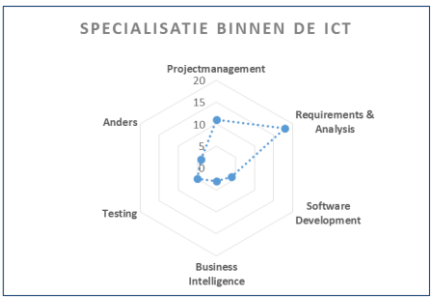

HET BESLISSINGSONDERSTEUNENDE MODEL

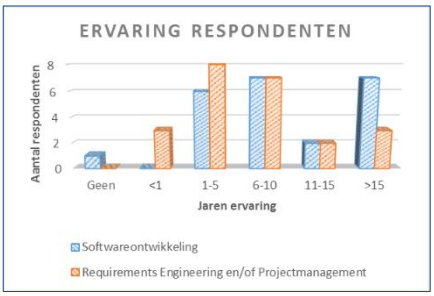
Een beslissingsondersteunende model ten behoeve van de te maken keuzes op het gebied van requirements specificatie-aanpakken in software-ontwikkelprocessen.

FIGUUR 16: BIJLAGE 4, HET BESLISSINGSONDERSTEUNENDE MODEL

7. RESULTATEN

In dit hoofdstuk worden de resultaten van de validatie van het model besproken. De vragenlijst is verstuurd aan 59 specialisten die werkzaam zijn binnen de IT sector, daarvan hebben 23 respondenten (39%) de vragenlijst ingevuld. Van deze 23 respondenten zijn er 21 bij Info Support werkzaam en 2 respondenten bij MivarGroup. De volledige (steekproef)populatie is mannelijk en de gemiddelde leeftijd van de respondenten is 35 jaar.





FIGUUR 17: SPECIALISATIE EN ERVARING RESPONDENTEN GEÏLLUSTREERD

Wanneer er gekeken wordt naar de ervaring op het gebied van Softwareontwikkeling (SO) en op het gebied van Requirements & Analysis (RA) en/of Projectmanagement (PM) blijkt uit de resultaten dat het grootste gedeelte van de respondenten specialist is op het gebied van RA, op de tweede plaats staat PM [9]. Dit is niet verwonderlijk omdat dit de doelgroep was van de vragenlijst en de vragenlijst expliciet naar deze specialisten is verstuurd. Tevens is er gevraagd naar het aantal jaar ervaring van de respondenten, in de softwareontwikkeling in het algemeen en op het gebied van RA en PM. Figuur 17 illustreert dat, op 1 respondent na, alle respondenten meer dan 1 jaar ervaring op het gebied van SO en RA en PM. Bijna 40 procent van de respondenten heeft meer dan 10 jaar ervaring op het gebied SO en meer dan 20 procent heeft meer dan 10 jaar ervaring op het gebied van RA en/of PM. De resultaten zijn gevisualiseerd in figuur 17.

7.1 HET MODEL

Zoals beschreven dient het model een tweetal doelen, in deze sectie worden de resultaten van de twee doelen afzonderlijk beschreven en wordt het algemene oordeel over het model besproken.

HET MODEL ALS HULPMIDDEL VOOR DE MOGELIJK TE MAKEN KEUZES

Het eerste doel van het model, het als hulpmiddel dienen voor het kiezen van de juiste requirements specificatie-aanpak op basis van de projecteigenschappen, is getoetst middels de toepassingsvragen op de casestudies. In het onderzoek waren een drietal casestudies opgenomen, elke respondent kreeg 1 casestudie voorgelegd met daarbij, per doel, de vraag of hij/zij het eens was met de voorgestelde requirements specificatie-aanpak keuze van het model. Om de resultaten te kunnen interpreteren zijn de antwoorden van de drie casestudies samengevoegd en gegroepeerd op de doelen uit het model.

In figuur 23 is de verdeling van de resultaten geïllustreerd. Zoals de resultaten weergeven zijn de respondenten het over het algemeen eens, 68%[10], met de gemaakte keuze voor een requirements specificatie-aanpak door het model. In de onderstaande sectie worden de resultaten van het onderzoek, per doel van het model, kort besproken:

DOEL 1: HET DEFINIËREN VAN EEN VISIE

Uit de resultaten blijkt dat 65% van de experts het eens was met de keuze van het model (*doel 1*) toegepast op de drie casestudies. Indien de experts het niet eens waren met de keuze van het model, werd vaak als reden gegeven dat de gekozen aanpak niet het 'enige' hulpmiddel hoefde te zijn en dat het combineren van de aanpakken beter zou aansluiten op de situatie. Een andere veel genoemd antwoord was dat er, in welke situatie dan ook, gekozen moet worden voor aanpak *2: "Het valideren en bijstellen van een visie*".

[9] Meerdere specialisaties mogelijk.
[10] Gemiddeld gezien op basis van de 4 doelen in het model

DOEL 2: PROCES VAN HET DEFINIËREN VAN DE REQUIREMENTS VASTSTELLEN

Uit de resultaten blijkt dat 74% van de experts het eens was met de, door het model, gemaakte keuze van het model (*doel 2*). Een veel geplaatste opmerking, van experts die het oneens waren, is dat een '*Just In Time en het Just Enough definiëren van de requirements'* ook een erg risicovolle aanpak kan zijn, dit door mogelijke *'hick-ups'* in het requirements engineering proces.

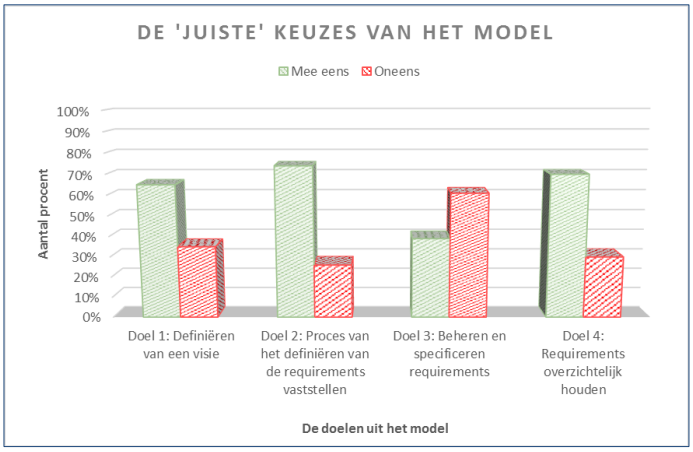
DOEL 3: BEHEREN EN SPECIFICEREN REQUIREMENTS

Uit de resultaten blijkt dat 39% van de ondervraagde het eens was met de gekozen aanpak door het model (*doel 3*). De belangrijkste reden waarom de experts het niet eens waren met de gekozen aanpak was dat, in de ogen van de experts, in elke situatie (nieuwe) agile technieken moeten worden toegepast. Aanpak 5, *"Het beheren en specificeren van de requirements middels epics/features, user stories en een product backlog in combinatie met Specification By Example"* werd door bijna alle experts in alle situaties geprefereerd als (additionele) aanpak.

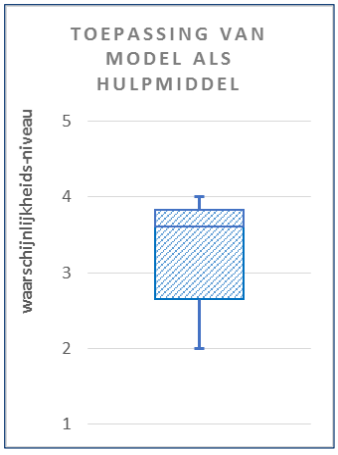
DOEL 4: REQUIREMENTS OVERZICHTELIJK HOUDEN

Uit de resultaten blijkt dat 70% van de experts het eens was met de gekozen additionele aanpakken door het model (*doel 4*). Indien de experts het niet eens waren met de (niet) gekozen additionele aanpakken dan werd er als reden geven dat in alle situaties *story mapping* zou moeten worden toegepast.

De toepasbaarheid van het model is niet enkel en alleen getoetst middels de casestudies maar is ook gevalideerd aan de hand beoordelingsvragen over de 'toepasbaarheid van het model als hulpmiddel'. De vraag luidde: "*Zou u het model als hulpmiddel gebruiken wanneer u in de toekomst een requirements specificatie-aanpak moet kiezen voor een project?".* De resultaten van deze vraag zijn geïllustreerd in figuur 24[11]. Uit de resultaten blijkt dat het merendeel van de experts van mening is dat het model, als hulpmiddel voor het kiezen van de juiste requirements specificatie-aanpak, toepasbaar is in de praktijk (M = 3.35, SD = 0.83). De resultaten zijn gevisualiseerd in figuur 18 en figuur 19.



FIGUUR 18: DE 'JUISTE' KEUZES VAN HET MODEL GEÏLLUSTREERD



FIGUUR 19: DE TOEPASSING VAN HET MODEL GEÏLLUSTREERD

[11] Op basis van de Likertschaal: 1 = Zeer onwaarschijnlijk, 2 = Onwaarschijnlijk, 3 = Neutraal, 4 = Waarschijnlijk, 5 = Zeer waarschijnlijk

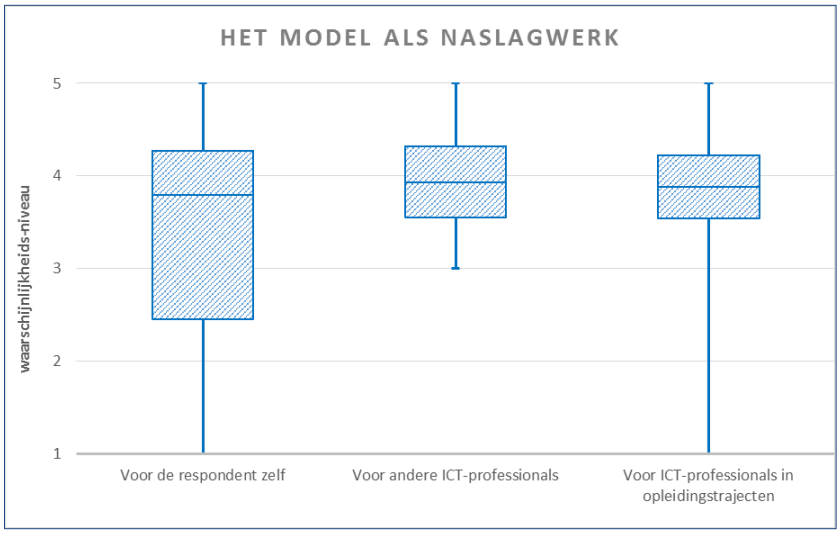
HET MODEL ALS NASLAGWERK OM MEER TE WETEN TE KOMEN OVER DE MOGELIJK TE MAKEN KEUZES

Om te toetsen of het model als naslagwerk kan fungeren, om meer te weten te komen over de hedendaagse beschikbare requirements specificatie-aanpakken, zijn er een drietal beoordelingsvragen gesteld aan de respondent. De resultaten van deze vragen zijn geïllustreerd in figuur 20.

De eerste vraag luidde: "*Zou het model, voor u, als naslagwerk kunnen dienen om meer te weten te komen over de hedendaagse beschikbare requirements specificatie-aanpakken?".* Uit de resultaten blijkt dat het merendeel van de experts van mening is dat ze het model 'waarschijnlijk' gaan gebruiken als naslagwerk in de toekomst(*M* = 3.48*, SD* = 1.12). Uit de resultaten blijkt wel dat de experts hierover zeer verdeeld zijn, zo acht bijna 26% van de experts het 'zeer onwaarschijnlijk' of 'onwaarschijnlijk' dat het model als hulpmiddel door hen gebruikt gaat worden in de toekomst. Als dieper wordt ingegaan op de resultaten blijkt dat het aantal jaren werkervaring in de softwareontwikkeling, en expliciet ervaring op het gebied van Requirements Engineering en/of Projectmanagement, een duidelijk invloed heeft op het feit of de experts het model wel of niet zouden gebruiken als naslagwerk. Van de 11 respondenten met minder dan 11 jaar ervaring, geeft 73%aan het model waarschijnlijk tot zeer waarschijnlijk te gaan gebruiken als naslagwerk.

De tweede vraag: "*Zou het model, voor andere IT professionals, als naslagwerk kunnen dienen om meer te weten te komen over de hedendaagse beschikbare requirements specificatie-aanpakken?",* heeft betrekking op de bruikbaarheid van het model als naslagwerk voor de gemiddelde IT-professional. Uit de resultaten blijkt dat 78% van experts van mening is dat het model, voor de IT-professional in het algemeen, als naslagwerk kan dienen (*M* = 3.91*, SD* = 0.60). Figuur 25 illustreert de minimale spreiding in de verdeling.

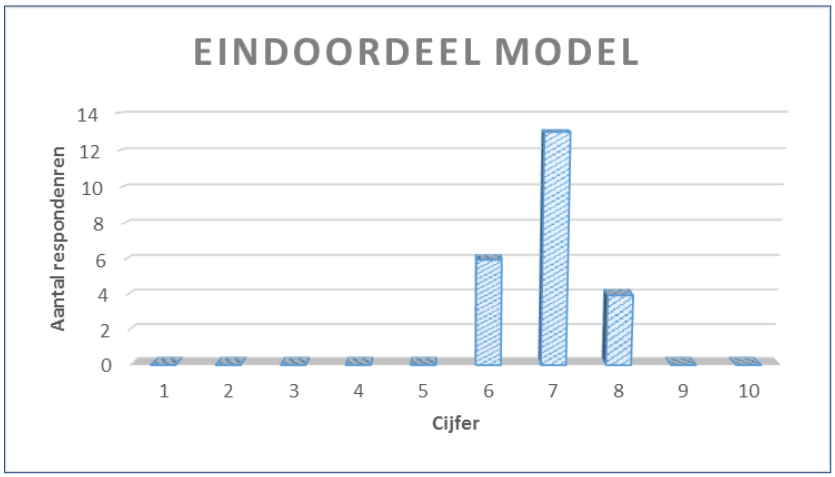
De derde en tevens laatste vraag met betrekking tot de bruikbaarheid van het model als naslagwerk luidde: *"Zou het model in opleidingstrajecten als naslagwerk kunnen dienen om meer te weten te komen over de hedendaagse beschikbare requirements specificatie-aanpakken?".* Uit het onderzoek blijkt dat meer dan 75 % van de ondervraagde het 'waarschijnlijk' tot 'zeer waarschijnlijk' acht dat het model in opleidingstrajecten als naslagwerk kan dienen (*M* = 3.74*, SD* = 0.57).



FIGUUR 20: RESULTATEN VAN HET MODEL ALS NASLAGWERK GEÏLLUSTREERD

ALGEHELE BRUIKBAARHEID

Als laatste werd aan de experts gevraagd om een eindoordeel te geven over de algehele bruikbaarheid van het model in de praktijk. Met een gemiddelde van een 6.9 (M = 6.86, SD = 0.69) blijkt dat het model als bruikbaar wordt ervaren door de experts. Waar bij eerdere vragen nog een duidelijk verschil waarneembaar was tussen de waardering van onervaren en ervaren experts, is dit bij het eindoordeel over de bruikbaarheid van het model niet het geval. In de figuur 21 zijn de resultaten grafisch weergegeven:



FIGUUR 21: RESULTATEN EINDOORDEEL MODEL GEÏLLUS0REERD

7.2 OPMERKINGEN

Acht respondenten hebben van de mogelijkheid gebruik gemaakt om op- en/of aanmerkingen te geven op het model. De algemene tendens in de opmerking was dat het keuzemodel vooral bruikbaar is voor 'nieuwelingen' en het voor ervaren IT-professional wellicht (alleen) als naslagwerk kan dienen.

Een van de experts merkt op: "*Het model kan met name nuttig zijn om keuzes te verantwoorden aan opdrachtgevers en andere stakeholders*". Dit is een zeer interessante opmerking, oorspronkelijk was het model bedoeld voor de ondersteuning van projectmanagers en/of requirements engineers, maar als er vanuit dit perspectief naar wordt gekeken, kan het model inderdaad ook ter verantwoording gebruikt worden.

Een andere respondent geeft aan dat de keuze van een bepaalde aanpak complexer is dan beschreven staat in het model: *"Voor een complete waterval organisatie is het niet handig om requirements agile aan te pakken, terwijl de rest waterval blijft werken. Je zult dus over de hele linie van het software ontwikkelproces moet kijken en niet alleen naar een specifieke discipline*". Dit is een zeer terechte opmerking en hier moet zeker rekening mee gehouden worden, echter is er geprobeerd om het model zo simplistisch mogelijk te houden zodat het model generiek toepasbaar blijft.

Als laatste opmerking werd genoemd dat het model niet aangeeft wat de kosten zijn van het toepassen van een bepaalde aanpak op korte en op lange termijn. De expert zegt hierover het volgende: "*Het model geeft niet aan wat de kosten van het gebruik van een aanpak zijn (als iedereen op training moet is dat goedkoper dan wanneer de mindset van de organisatie anders moet). Wat ontbreekt is kans op succes bij eerste gebruik en lange termijn effect. Als kans groot is zou je het sowieso wel willen proberen. Als lange termijn effect laag is loont het misschien niet de moeite.*" Hier heeft de expert helemaal gelijk in en dit zou een zeer nuttige uitbreiding kunnen zijn van het bestaande model.

8. CONCLUSIE

Aan het begin van het onderzoek is de volgende (hoofd) onderzoeksvraag gedefinieerd:

> *"Welke requirements specificatie-aanpakken, en bijbehorende requirements artefacten, kunnen ten behoeve van welke doel en in welke situatie worden toegepast in software-ontwikkelingsprocessen?"*

Om deze vraag te kunnen beantwoorden is er eerst ingezoomd op de verschillende type software-ontwikkelprocessen en welke rol requirements spelen gedurende de ontwikkeling van een product. Requirements worden in de softwareontwikkeling gebruikt om eisen vast te leggen waaraan een systeem moet voldoen. Uit het onderzoek blijkt dat het requirements engineering proces, de totstandkoming, het documenteren en onderhouden van de requirements, bestaat uit drie kernactiviteiten: het eliciteren, het specificeren en het valideren van de requirements.

In de afgelopen decennia hebben verschillende experts en organisaties geprobeerd om het softwareontwikkelingsproces op verschillende manieren te structuren. In de traditionele softwareontwikkeling ging men er van uit dat alle requirements, voorafgaand aan het project, bekend waren en deze gedurende het project niet kunnen wijzigen. In de hedendaagse agile softwareontwikkeling wordt er vanuit gegaan dat requirements, gedurende het project, evolueren en worden (late) wijzigingen in de requirements omarmd. Op basis van deze verschillende aanpakken om software te ontwikkelen, is er onderzocht welke factoren een rol spelen bij de keuze van een geschikte requirements specificatie-aanpak. Deze factoren, situatiefactoren genoemd, zijn geïdentificeerd en gecategoriseerd in de volgende categorieën: stakeholder(s), team, product en omgeving. Per categorie zijn de relevante situatiefactoren voor de toepassing van het model beschreven. Om te onderzoeken welke invloed een requirements specificatie-aanpak heeft op de uiteindelijke (kwaliteit) van de software, zijn er een viertal vergelijkingsfactoren opgesteld. Dit zijn: functionele kwaliteit, technische kwaliteit, kosten en tijd.

Op basis van de bevindingen, gecombineerd met informatie verkregen middels expertinterviews, is er een model geconstrueerd waarin de mogelijk te kiezen requirements specificatie-aanpakken per doel zijn weergegeven. Zoals gedefinieerd, dient het model een tweetal doelen, 1) het ondersteunen van IT-professionals bij de te maken keuzes op het gebied van requirements specificatie-aanpakken in software-ontwikkelprocessen en 2) het dienen als naslagwerk om meer te weten te komen over de hedendaagse beschikbare requirements specificatie-aanpakken. Om te toetsen of het model daadwerkelijk beiden doelen dient, is de toepasbaarheid en de bruikbaarheid van het model getoetst middels een vragenlijst onder 59 experts in het vakgebied. 23 experts hebben de vragenlijst ingevuld. Het merendeel van de respondenten, 63%, is het eens met de keuze van het model voor een requirements specificatie-aanpak[12]. Tevens zijn de respondenten van mening dat het model door henzelf, andere IT-professionals of in opleidingstrajecten, waarschijnlijk van toegevoegde waarde is in de vorm van naslagwerk.

Alhoewel het merendeel (63%) het eens was met de gemaakte keuzes door het model, kan er niet hard worden geconcludeerd dat het model daadwerkelijk toepasbaar is in de praktijk als beslissingsondersteunend model. Zoals een deel van de respondenten ook aangeeft, is de keuze voor de juiste requirements specificatie-aanpak complexer dan beschreven in het model.

[12] Getoetst op drie casestudies

9. DISCUSSIE EN VERDER ONDERZOEK

Softwareontwikkeling en requirements engineering zijn complexe onderzoeksgebieden binnen de IT. In de snelle wereld van de IT volgen ontwikkelingen elkaar in een rap tempo op en daardoor is de beschikbare literatuur vaak op het moment van publiceren al weer achterhaald. Om dit tegen te gaan is er geprobeerd om zoveel mogelijke actuele kennis, van bijvoorbeeld blogs of artikelen op het internet, te combineren met wetenschappelijke literatuur. Het probleem is echter dat online bronnen variëren in de mate van kwaliteit en hierdoor de validiteit van het onderzoek in gevaar kan komen. Om deze reden is er gekozen om enkel online bronnen te raadplegen van gerenommeerde experts. Tevens is er geprobeerd om zoveel mogelijk gebruik te maken van de ervaring en kennis van de experts binnen Info Support.

Op basis van de literatuur en expertinterviews is het uiteindelijke beslissingsondersteunende-model tot stand gekomen. Doordat het model gedeeltelijk is gebaseerd op de ervaring en kennis van werknemers binnen Info Support, worden er in het model aannames gedaan die niet zijn geverifieerd door meerdere bronnen. Toch mag er worden aangenomen dat het model toepasbaar en bruikbaar is voor andere bedrijven, dit omdat het expertiseniveau binnen Info Support op dit vakgebied zeer hoog is.

Om het model te toetsen is er gebruik gemaakt van een kwantitatieve vragenlijst. Er is gekozen voor deze onderzoeksvorm om zodoende zoveel mogelijk experts binnen het vakgebied te bereiken, dit om de validiteit en de betrouwbaarheid van het onderzoek te verhogen. Toch is achteraf gebleken dat dit wellicht niet de meest verstandigste manier was om het model te valideren. Uit de motivaties, waarom een respondent het niet eens was met de door het model gemaakte keuze, blijkt dat sommige respondenten het achterliggende model te weinig hebben geraadpleegd en op basis van (eigen) interpretatie keuzes hebben gemaakt. Het houden van een focusgroep, waar het model eerst volledig wordt uitgelegd, zou in de toekomst een betrouwbaardere manier zijn om het model te valideren. Helaas was dit binnen de huidige scope en de tijdsplanning van de experts niet mogelijk.

Het model, voortgekomen uit deze scriptie, blijkt niettemin bruikbaar te zijn in de praktijk. Het model als hulpmiddel, dus als beslissingsondersteunend model, voor de te kiezen requirements specificatie-aanpakken op basis van projecteigenschappen blijkt minder bruikbaar te zijn als van te voren gedacht. Een reden hiervoor is dat experts verschillende meningen blijken te hebben over een ‘juiste’ aanpak in een specifieke situatie. Ook zijn sommige begrippen verkeerd geïnterpreteerd door de respondenten waardoor een objectief oordeel over het model niet mogelijk was. Tevens wordt het model, door sommige respondenten, afgedaan als basiskennis voor een IT-professional. Dit is misschien het geval voor de experts binnen Info Support ,maar dit is zeker niet het geval voor de gemiddelde IT-professional in de branche.

Uit analyse van de resultaten blijkt ook dat veel respondenten altijd voor de meest idealistische requirements specificatie-aanpak kiezen. De meeste idealistische aanpak, en in veel gevallen ook de ‘nieuwste’ (agile) aanpak, hoeft niet in alle gevallen de juiste aanpak te zijn. Dit zou te wijten kunnen zijn aan missende details in de voorgelegde casestudies, aspecten zoals bedrijfscultuur, opleidingsniveau en beschikbaarheid van de medewerkers zijn niet benoemd in de casestudies.

9.1 VERDER ONDERZOEK

Zoals één van de respondenten ook aangaf, voegt de visualisatie van het model weinig toe. De visualisatie is statisch en is niet van toegevoegde waarde naast de tekstuele beschrijvingen van het model. In de toekomst zou het model uitgebreid kunnen worden met een interactief keuzemodel, zodat op basis van de ingegeven projecteigenschappen de mogelijk te kiezen aanpakken worden weergegeven. In een nog later stadium zou het beslissingsondersteunende model getransformeerd kunnen worden tot een beslissingssysteem. Een beslissingssysteem is in staat om op basis van projecteigenschappen automatisch de juiste aanpak te kiezen.

Een ander welkome uitbreiding op het model, wat uit de resultaten naar voren is gekomen, is dat het model rekening gaat houden met het type bedrijf waarin het model wordt toegepast. Is het bijvoorbeeld een volledige waterval organisatie, dan is het niet handig om een agile aanpak voor te schrijven. De verschillende

bedrijfsculturen, in combinatie met de requirements-specificatie-aanpakken, zullen dus in kaart gebracht moeten worden.

Als laatste suggestie zou de kwaliteitsanalyse in het model uitgebreid kunnen worden met het kostenplaatje van de invoering van de desbetreffende aanpak. Tevens zijn aspecten zoals de succeskans bij het kiezen van een aanpak op lange termijn, zeer interessant.

10. BIBLIOGRAFIE

Agile Enterprise, IT Consulting Solutions. (2013). *Advantages of Agile Development*. Opgehaald van Agile Enterprise, IT Consulting Solutions: http://www.agileenterprises.com/agile-development/advantages-of-agile-development

Ambler, S. (2002). *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process.* New York: John Wiley & Sons, Inc.

Ambler, S. (2014, Juli 5). *Effective Practices for Modeling and Documentation*. (Scott Ambler + Associates) Opgeroepen op 05 02, 2014, van Agile Modeling: http://www.agilemodeling.com/

Ambler, Scott; Lines, Mark. (2011). *Disciplined Agile Delivery: An introduction.* Somers: IBM Corperation.

Amrit, C., Debije, L., Engelsman, W., Gaaloul, K., Heerink, L., Hoppenbrouwers, S., . . . ….. (2012). *Agile Service Development: Combining Adaptive Methods and Flexible Solutions.* New York: Springer Heidelberg New York Dordrecht London.

Beck, K., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., . . . Thomas, J. (2001). *Manifesto for Agile Software Development*. Opgehaald van Agile Manifesto: http://agilemanifesto.org/iso/nl/

Benington, B. (1956). Production of Large Computer Programs. *ONR Symposium on Advanced Programming Methods for Digital Computers*, 15-27.

Black, R. (2002). Investing in Software Testing: The Risks to System Quality. Rex Black Consulting.

Bloch, M., Blumberg, S., & Laartz, J. (2012). *Delivering large-scale IT projects on time, on budget, and on value.* Oxford: McKinsey & Company in conjunction with the University of Oxford.

Boehm, B. (1996). Anchoring the Software Process. *IEEE Software*, 73-82.

Brame, A., & Barlow, G. (2010). *KPMG New Zealand Project Management Survey 2010.* KPMG.

Cao, L., & Balasubramaniam, R. (2008). Agile Requirements Engineering Practices: An Empirical Study. *IEEE Computer Society, 08*(0740 - 7459), 60-67.

Chan, Jeremy . (2010, 06 02). *Agile Development and the Mona Lisa*. (Johan Group) Opgeroepen op 05 01, 2014, van http://aseriesoftubes.com/articles/agile-development-and-the-mona-lisa/

Cockburn, A. (2002). *Agile software development.* Boston: Addison-Wesley.

Cockurn, A. (2001, Februari). *How "Learn Early, Learn Often" Takes Us Beyond Risk Reduction*. (Alistair Cockburn) Opgeroepen op 06 2014, 06, van http://alistair.cockburn.us/Disciplined+Learning

Cohn, M. (2004). Advantages of User Stories for Requirements. *InformIT Network*.

Crosby, P. (1980). *Quality is Free - The Art of Making Quality Certain.* Mentor . Opgehaald van http://www.philipcrosby.com/

Duka, D. (2012). Agile Experiences in Software Development. *Proceedings of the 35th International Convention*, 692-687.

Eberlein, A., & Sampaio do Prodo Leite, J. (2002). *Agile requirements definition: A view from requirements engineering.*

Eveleens, J., & Verhoef, C. (2010). The Rise and Fall of the Chaos Report Figures. *IEEE Software* , 30-36.

Geambasu, C. V., Jianu, I., & Gavrila, A. (2011). Influence factors for the choice of a software development methodology. *Accounting and Management Information Systems, 10*(4), 479-494.

Gröber, M. (2013). *Investigation of the Usage of Artifacts in Agile.* MÜNCHEN: DER TECHNISCHEN UNIVERSITÄT MÜNCHEN.

Hasmi, S. (2007). Software Quality Assurance in XP and Spiral - A Comparative Study. *Computational Science and its Applications*, 367-374.

Hastie, S., & Wick, A. (2014, 03 04). *User Stories and Use Cases - Don't Use Both!* (BusinessAnalystTimes) Opgeroepen op 95 22, 2014, van http://www.batimes.com/articles/user-stories-and-use-cases-dont-use-both.html

Henderson-Sellers, B., & Ralyté, J. (2010). Situational Method Engineering: State-of-the-Art Review. *Journal of Universal Computer Science*(vol. 16, no. 3), 424-478.

Hevner, A., & Chatterjee, S. (2010). *Design Research in Information Systems.* Springer.

Hietkamp, R. (2014, Augustus 11). Requirements Engineering Practices.

Jacobson, Ivar; Spence, Ian; Bittner, Kurt. (2011). *USE-CASE 2.0: The Guide to Succeeding with Use Cases.* Ivar Jacobson International.

Jansen, S. (2014, 07 23). Requirements Engineering Practices.

Jones, C. (1995). Patterns of large software systems: failure and success. *Computer*, 86-87.

Kleine Staarman, C. (2013). *Agile requirements engineering & SCRUM.* Nijmegen: Radboud Universiteit.

Korsaa, M., Olesen, R., & Vinter, O. (2002). *Iterative Software Development - A Practical View.* Delta: Datateknisk.

Kulak, D., & Guiney, E. (2003). *Use Cases: Requirements in Context.* Addison-Wesley Professional.

Leffingwell, D. (2010). *Agile Software Requirements.*

Lukassen, V. (2014, 22 07). Requirements Engineering Practices.

Nehru, J., Jagityal, K., & Munassar, N. (2010). A Comparison Between Five Models Of Software Engineering. *IJCSI International Journal of Computer Science Issues*, 94-101.

Nepomuceno, V., & Fontana, M. (2013). Decision support system to project software management. *IEEE International Conference on Systems, 2013*, 964-969.

Nieboer, H. (2014, 07 02). Interview Requirements Engineering Practices.

Office of Information Services. (2005). *Selecting a development approach .* Centers for Medicare & Medicaid services.

Riley, Geoff. (2012, September 23). *Monopoly & economic efficiency*. Opgehaald van tutor2u: http://tutor2u.net/economics/revision-notes/a2-micro-monopoly-economic-efficiency.html

Royce, W. (1970). Managing the Development of Large. *IEEE CS Press*, 328-339.

Saarnak, S., & Gustafsson, B. (2003). *A comparison of lifecycles - Agile software processes vs. projects in non-Agile.* Ronneby: Software Engineering and Computer Science, Blekinge Institute of Technology.

Snijder, J. (2014, 08 11). Requirements Engineer Practices.

Sommerville, I. (2011). *Software Engineering.* Addison-Wesley.

Standish Group. (1994). *Charting the Seas of Information Technology—Chaos.* West Yarmouth: The Standish Group International.

Suscheck, C. (2012, 01 18). *Defining Requirement Types: Traditional vs. Use Cases vs. User Stories*. (CM Crossroads) Opgeroepen op 25 05, 2014, van http://www.cmcrossroads.com/article/defining-requirement-types-traditional-vs-use-cases-vs-user-stories?page=0%2C0

The Waterfall Development Methodology. (sd). (The Smart Method Limited) Opgeroepen op 05 01, 2014, van http://learnaccessvba.com/application_development/waterfall_method.htm

Thomas, S. (2003, December). *Revisiting the Iterative Incremental Mona Lisa*. (It's a Delivery Thing) Opgeroepen op 05 01, 2014, van http://itsadeliverything.com/revisiting-the-iterative-incremental-mona-lisa

Tom Sylvester. (2013, Juni 11). *Waterfall, Agile & the "Triple Constraint"*. Opgehaald van Tom Sylvester: http://tom-sylvester.com/LEAN-AGILE/WATERFALL-AGILE-THE-TRIPLE-CONSTRAINT/

Trendowicz, A., & Münch, J. (2009). Factors Influencing Software Development Productivity - State of the Art and Industrial Experiences. *Computers Elsevier, 77*, 185-241.

University of Houston. (2012). *Rational Unified Process: Artifacts*. (Rational Software Corporation) Opgeroepen op 05 21, 2014, van Rational Unified Process: http://sce.uhcl.edu/helm/rationalunifiedprocess/process/artifact/ovu_arts.htm

van Loon, G. (2014, Augustus 18). Requirements Engineer Practices.

VersionOne. (2014). *8th Annual State of Agile Survey.* VersionOne.com.

Visser, J. (2007). Software Quality and Risk Analysis - Invited lecture Master Sw Technology. Utrecht: Software Improvement Group.

11. BIJLAGENBOEK

Radboud Universiteit Nijmegen

BIJLAGENBOEK

Requirements specificatie in software-ontwikkelprocessen

MASTERSCRIPTIE INFORMATIEKUNDE
RADBOUD UNIVERSITEIT NIJMEGEN

S. VAN OOSTENBRUGGE

INHOUDSOPGAVE

1. LINEAIRE, ITERATIEVE EN INCREMENTELE EN AGILE PROCESSEN GEÏLLUSTREERD

Het lineaire proces

Om te illustreren hoe software wordt ontwikkeld in een lineair ontwikkelproces, wordt de vergelijking gemaakt met het schilderen van de *Mona Lisa*. Als *Da Vinci* het schilderij middels de lineaire structurering had geschilderd, was de *Mona Lisa* laag voor laag geschilderd. Er werd dus gestart met het schilderen van de achtergrond en er vervolgens werd er steeds meer detail in de voorgrond aangebracht. Het laatste, en tevens het belangrijkste, detail van het schilderij: het gezicht van *Mona Lisa*, werd in het laatste stadium van het proces geschilderd. Het lineaire schilderproces word als volgt geïllustreerd (Chan, Jeremy , 2010):

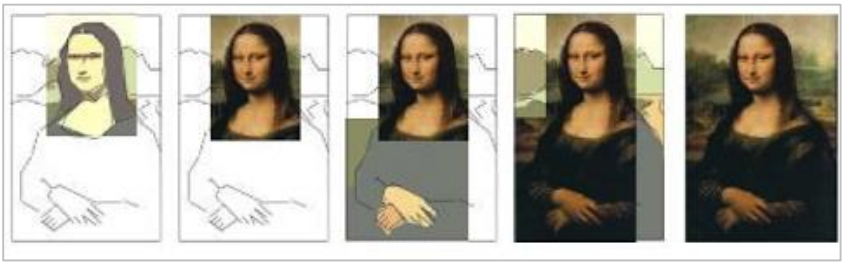
FIGUUR 18: HET WATERVAL PROCES GEÏLLUSTREERD

Een aantal belangrijke punten kunnen hier worden opgemerkt. Ten eerste is het duidelijk dat, als het project was geannuleerd na fase 3, het schilderij nu niet in het *Louvre* had gehangen. Het schilderij bevatte na fase 3 namelijk nog te weinig waarde in de ogen van de klant, dit doordat het belangrijkste detail: het gezicht van *Mona Lisa*, pas in het laatste stadium van het proces zou worden getekend. Ook de vraag van de klant om de kleur van de achtergrond te veranderen, in een laat stadium van het proces, zou grote gevolgen hebben voor de planning en de kosten van het project. Door het wijzigen van de achtergrond kleur, moet immers het kleurpalet in het gehele schilderij worden aangepast (Chan, Jeremy , 2010).

Dezelfde aspecten, zoals hierboven benoemd, zijn van toepassing op lineaire software-ontwikkelprocessen. Kritieke beslissingen, zoals het kiezen van een kleurpalet, worden aan het begin van het proces genomen en functionaliteit met de meeste waarde voor de klant, zoals het gezicht van de *Mona Lisa*, wordt pas aan het eind van het project gerealiseerd (Chan, Jeremy , 2010).

Iteratieve en Incrementele processen:

Om te illustreren hoe I & I processen verlopen, wordt ook hier het schilderen van de Mona *Lisa* als metafoor gebruikt. Als *Da Vinci* de *Mona Lisa* op een iteratieve en incrementele manier had geschilderd, was *Da Vinci* eerst met het belangrijkste deel van het schilderij begonnen: Het hoofd. Voordat *Da Vinci* aan een ander deel van het schilderij gaat werken, wordt eerst het hoofd verfijnd tot het voldoet aan de wensen van de klant. Vervolgens begint *Da Vinci* aan het tweede belangrijkste onderdeel van de tekening: Het lichaam van *Mona Lisa*. Dit proces herhaalt zich totdat het schilderij is voltooid. Het iteratieve en incrementele schilderproces word als volgt geïllustreerd (Thomas, 2003):



FIGUUR 19: HET ITERATIEVE EN INCREMENTELE PROCES GEÏLLUSTREERD

Uit deze metafoor kan het volgende worden opgemaakt: Indien er wordt besloten om het project te annuleren na de tweede fase, is er wellicht een hoop geld verloren gegaan maar is dit slechts een fractie van het verlies dat zou optreden als dit besluit zou worden genomen aan het eind van het proces. Doordat eerst de functionaliteit (het hoofd) in grote lijnen wordt gerealiseerd, kan de opdrachtgever tussentijds gemakkelijker feedback geven.

Agile processen

Om te illustreren hoe softwareontwikkeling er in een agile processen uitziet, wordt er ook hier gebruik gemaakt van de metafoor van het schilderen van de *Mona Lisa*. In tegenstelling tot de iteratieve en incrementele processen, worden niet eerst de contouren van het schilderij geschilderd, maar begon *Da Vinci* direct met het belangrijkste onderdeel voor de klant: Het hoofd van de *Mona Lisa*. Doordat *Da Vinci* niet eerst de contouren van het schilderij tekende, weet de klant niet hoeveel tijd en geld het schilderij in totaal gaat kosten, totdat alle wensen van de klant zijn vervuld. Nadat het hoofd van de *Mona Lisa* was voltooid, werd het schilderij opgeleverd aan de klant. Vervolgens werd in overleg met de klant besloten wat er als volgende geschilderd moest worden. Dit proces herhaalde zich totdat het schilderij voldeed aan de wensen van de opdrachtgever. Het schilderproces, gebaseerd op de Agile principes, word als volgt geïllustreerd (Thomas, 2003):

FIGUUR 20: HET AGILE PROCES GEÏLLUSTREERD

In dit proces vallen een paar punten op: Doordat er niet wordt gestart met het definiëren van het ontwerp van het gehele systeem (contouren van het schilderij), kan er direct worden gestart met het realiseren van het systeem. Er kan zowaar geen exacte planning worden gegeven, maar de klant ziet direct resultaat door de snelle start van het project. Het tweede punt is de volledige oplevering van de functionaliteit, als de opdrachtgever aan het eind van fase 2 besluit om het project te annuleren, bevat het systeem de belangrijkste functionaliteit en is het vervolgens ook bruikbaar in de praktijk. Het laatste punt heeft betrekking op de flexibiliteit van het proces, doordat er niet gestart is met een volledig ontwerp van het systeem, heeft de klant de mogelijkheid om het systeem tussentijds het proces bij te sturen en dus de requirements te wijzigen.

2. Problemen met requirements Engineering in Agile ontwikkelprocessen

Het gebruik van agile requirements technieken en artefacten in agile ontwikkelprocessen brengt een aantal problemen of risico's met zich mee. In deze paragraaf worden de problemen beschreven die het meest naar voren komen in de literatuur:

1. Gebrek aan planning en kostenschatting

Doordat er geen aparte requirements en analyse fase meer plaatsvindt in een agile project, wordt de initiële schatting van het project gebaseerd op het aantal bekende *user stories* op DAT moment in het proces. Veel *user stories* worden uiteindelijk niet gerealiseerd, of nieuwe *user stories* worden toegevoegd gedurende het project, hierdoor moet de oorspronkelijk planning en kostenschatting worden bijgesteld tijdens de ontwikkeling. Doordat de scope constant wijzigt, is het geven van een accurate kostenschatting en een planning een probleem in agile projecten. Het verkrijgen van managementondersteuning voor dergelijke agile projecten kan hierdoor een uitdaging zijn (Cao & Balasubramaniam, 2008).

2. Minimale specificatie en documentatie

In agile projecten wordt het systeem gedefinieerd middels het *Just Enough* principe en wordt er enkel een requirements artefact gebruikt indien er geen andere middelen zijn om het desbetreffende doel te bereiken (Ambler S. , 2002). Door de minimale specificatie van requirements en er pas aan het eind van het proces, als dit noodzakelijk is, wordt gedocumenteerd, kunnen er een aantal problemen optreden als gevolg van de minimale specificatie en documentatie gedurende het project:

Het eerste probleem kan worden veroorzaakt door het, gepland of ongepland, wegvallen van personeel. Als er gedurende het project 'ongepland' mensen het project verlaten, kan kennis over het systeem verloren gaan (Cao & Balasubramaniam, 2008). Dit zal moeten worden opgevangen door de overige medewerkers, wat resulteert in een verlaagde productiviteit. Een ander probleem wat kan optreden, doordat het systeem in sommige agile projecten pas wordt gedocumenteerd in de laatste fasen van het proces, is dat de juiste medewerkers niet meer beschikbaar zijn. Dit omdat het project zich al in het laatste stadium bevindt en dus de medewerkers al zijn overgeplaatst op een ander project (Ambler S. , 2002). Ook heeft het documenteren in een laat stadium als risico dat de financiering kan worden stopgezet of de motivatie, om de documentatie te schrijven, er niet meer is (Ambler S. , 2014).

3. Inadequate architectuur

In agile softwareontwikkeling staat het snel opleveren van *business value* centraal, de aanloopperiode en de start-up fase van het project wordt zo kort mogelijk gehouden zodat al in een vroeg stadium het team werkende software kan opleveren (*time-to-market*) (Duka, 2012). Het snel willen opleveren van werkende software kan leiden tot een inadequate architectuur van de applicatie, dit doordat de architectuur gekozen is in de eerdere cycli van het ontwikkelproces en de requirements gedurende het kunnen proces veranderen (Cao & Balasubramaniam, 2008). Het *refactoren* van de architectuur van het systeem kan aanzienlijk veel geld kosten en wordt vaak als onnodig bestempeld door de betrokkenen. Dit komt doordat het *refactoren* van het systeem geen concrete *business value* oplevert (Cao & Balasubramaniam, 2008). Inadequate architectuur kan leiden tot (toekomstige) problemen met de software, bijvoorbeeld een niet schaalbare architectuur of een onveilig systeem (Cao & Balasubramaniam, 2008).

4. Gebrek aan stakeholder participatie en betrokkenheid

Actieve stakeholder participatie en betrokkenheid is essentieel binnen een agile project, dit om informatie over het te bouwen systeem te verkrijgen en beslissingen te nemen op het gebied van requirements (Cao & Balasubramaniam, 2008). Mensen van naturen niet goed zijn in het definiëren van details, uit onderzoek blijkt echter dat mensen, wanneer er een oplossing wordt gepresenteerd, exact kunnen aangeven wat de positieve en negatieve punten van de desbetreffende oplossing zijn (Ambler S. , 2014). Actieve stakeholder participatie stelt het team in staat om een functionaliteit te realiseren en direct feedback te krijgen sen vervolgens aan de hand van de verkregen feedback de gerealiseerde functionaliteit te perfectioneren (Ambler S. , 2014). Uit

onderzoek (Cao & Balasubramaniam, 2008) blijkt dat veel bedrijven problemen hebben met het bewerkstelligen van een *on-site* klant representatie. Tevens blijkt er, indien er meer dan een klantgroep betrokken is, het bereiken van een consensus of een compromis in een korte ontwikkelcyclus uitdagend is.

De effectiviteit van de communicatie met de stakeholders en het team is afhankelijk van de volgende factoren: 1) klant beschikbaarheid, 2) consensus tussen de stakeholders en 3) het vertrouwen tussen de klant en het team (Cao & Balasubramaniam, 2008). Gebrek aan stakeholder participatie en betrokkenheid kan leiden tot verkeerd geïnterpreteerde requirements, verlaagde productiviteit van het team en in het ergste geval tot software die niet aan de wensen van de klant voldoet (Cao & Balasubramaniam, 2008).

5. Gebrek aan overzicht en traceerbaarheid van de requirements

Een best practice in agile processen, is het definiëren van requirements met een zo klein mogelijke scope (Ambler S. , 2014). De reden hiervan is dat requirements met een beperkte scope gemakkelijker te schatten en te realiseren zijn. Door het (kunstmatig) opsplitsen van functionaliteit naar kleinere brokken functionaliteit (*user stories*), ontstaat er fragmentatie in de requirements, dit kan leiden tot een gebrek aan overzicht bij de betrokkenen en vermindering in traceerbaarheid [13] van de requirements. Traceerbaarheid van requirements is noodzakelijk zodat bij wijzigingen in de requirements de gevolgen voor overige requirements kunnen worden geanalyseerd en voor elke requirement de bron of motivatie bekend is.

Door de fragmentatie in requirements focussen de betrokkenen zich op gedeeltes van het systeem en wordt het overzicht verloren (*big picture*). Grote brokken requirements worden in veel projecten in de vorm van *epics*, *use cases* of *features* gespecificeerd, dit om de samenhang tussen de functionaliteiten overzichtelijk te houden. Omdat deze grote brokken functionaliteit moeilijk te schatten zijn, en tevens niet gerealiseerd kunnen worden in één iteratie, worden deze brokken opgedeeld in meerdere kleinere brokken functionaliteit. Het streven is om het kleinste stuk functionaliteit te vinden dat nog van waarde is voor de klant (*business value*). Deze brokjes functionaliteit worden vervolgens in een iteratie gerealiseerd en getest. Doordat een iteratie kort is, wordt er maar een gedeelte van de *epic*, *use case* of *feature* opgeleverd. Dit kan resulteren in een oplevering van half werkende functionaliteit en doordat nergens, of met behulp van omslachtige technieken, staat geregistreerd wat er wel of niet van de *epic*, *use case* of *feature* in de desbetreffende sprint is gerealiseerd. Het raadplegen van de status van de gerealiseerde requirements, of de projectstatus in het geheel, is zodoende niet mogelijk.

6. Verwaarlozing van niet-functionele requirements

Niet-functionele requirements (NFR) worden vaak niet gedefinieerd en genegeerd gedurende de eerste cycli van het agile ontwikkelproces (Cao & Balasubramaniam, 2008). De reden hiervan is dat de stakeholder zich focussen zich op de *core* functionaliteit, wat de *business value* bevat, en aspecten zoals *scalability, maintainability, portability, safety, or performance* vergeten (Cao & Balasubramaniam, 2008).

Ook als het team en de desbetreffende stakeholders de belangrijkheid van NFRs inzien, hebben agile teams moeite met het definiëren van de NFRs in een Agile proces (Eberlein & Sampaio do Prodo Leite, 2002). Dit omdat een NFR niet als acceptatiecriteria beschouwd kan worden maar een *functionaliteit* vaak niet als *done* bestempeld kan worden indien de gerelateerde NFR niet is bewerkstelligd. Ook kunnen NFRs betrekking hebben op meerdere user stories, of zelfs op de gehele applicatie als bijvoorbeeld wordt gekeken naar de performance van de applicatie. Verwaarlozing van niet-functionele requirements kan leiden tot kritieke problemen bij in gebruik name van de software, bijvoorbeeld het uitvallen (*reliability*) van een bedrijfs-kritische applicatie

[13] Traceerbaarheid in requirements is de mogelijkheid om verschillende aspecten in het proces met elkaar te relateren.

3. DE EXPERTINTERVIEWS

DIT BETREFT EEN PUBLIEKE VERSIE VAN HET DOCUMENT. DE INTERVIEWS ZIJN ALS VERTROUWELIJK AANGEMERKT EN ZIJN DAAROM IN DEZE VERSIE WEGGELATEN.

4. HET MODEL

Radboud Universiteit Nijmegen

HET BESLISSINGSONDERSTEUNENDE MODEL

Een beslissingsondersteunend model ten behoeve van de te kiezen requirements-specificatie-aanpakken in software-ontwikkelprocessen

MASTERSCRIPTIE INFORMATIEKUNDE
RADBOUD UNIVERSITEIT NIJMEGEN

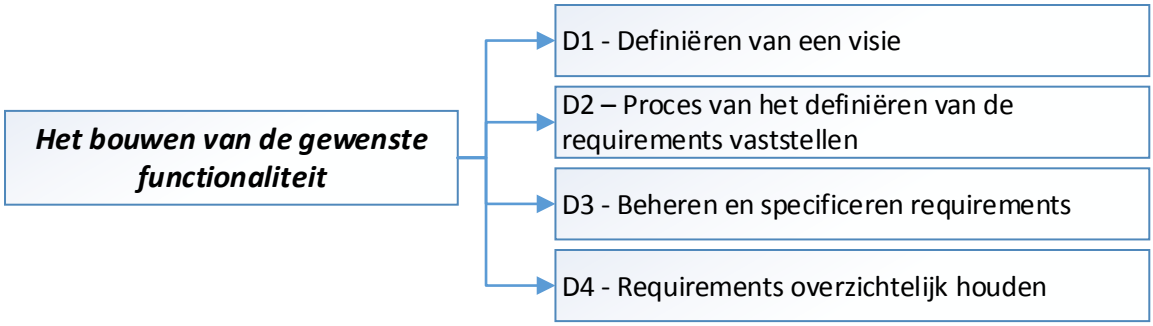
S. VAN OOSTENBRUGGE

INLEIDING

Het doel van het model is om ICT-professionals, in het bijzonder projectmanagers en requirements-engineers, te ondersteunen bij de te maken keuzes op het gebied van requirements-specificatie-aanpakken in softwareontwikkelingsprojecten. Tevens kan het model, inclusief toepassingseisen en de kwaliteitsanalyse, als informatiebron dienen voor de beschikbare requirements-specificatie-aanpakken in software-ontwikkelprocessen. Het model is tot stand gekomen middels het gedane literatuuronderzoek, interviews met experts uit de praktijk en blogs over 'nieuwe' aanpakken op het gebied van requirements engineering.

Achtergrondinformatie model

Het model is gebaseerd op de 'succesfactoren' in de softwareontwikkeling van *Dr. Alistair Cockurn*. Om deze succesfactoren, ook wel doelen genoemd, te behalen, dient men de juiste requirements-specificatie-aanpak te kiezen. In figuur 1 zijn de 4 subdoelen, ten behoeven van het hoofddoel: "*Het bouwen van de gewenste functionaliteit*" geïllustreerd:



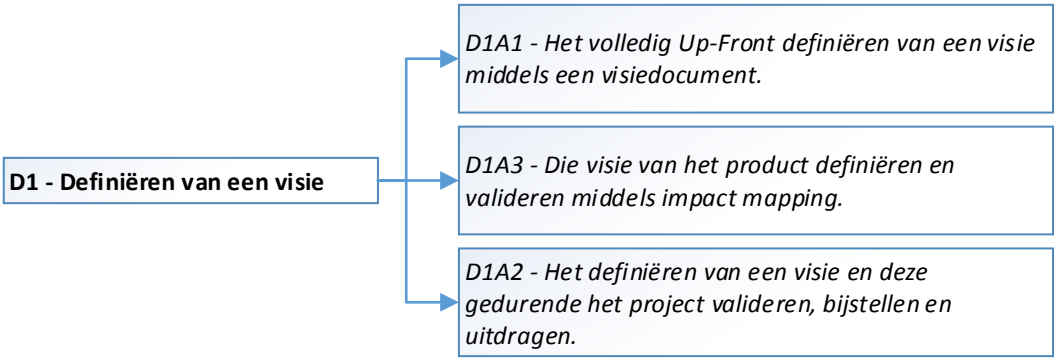
FIGUUR 1: HET HOOFDDOEL MET DE 4 SUBDOELEN

Voor elk van deze doelen zijn in het model de mogelijke requirements-specificatie-aanpakken beschreven. In het volgende hoofdstuk worden doelen, inclusief de mogelijke aanpakken, kort toegelicht.

DE DOELEN

DOEL 1: DEFINIËREN VAN EEN VISIE

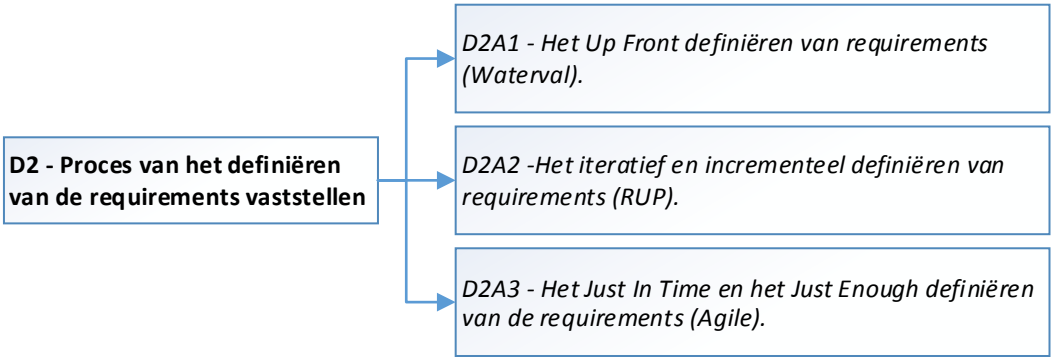
Het definiëren van een missie en een visie is de eerste stap die moet worden gezet wanneer er besloten wordt om een softwareontwikkelingsproject te starten. In de visie wordt vastgelegd waarom het project wordt geïnitieerd en aan welke eisen het systeem globaal moet voldoen om de doelstellingen te behalen. Het hebben van een visie is essentieel om de volgende redenen: 1) Achterhalen waarom het systeem gebouwd moet worden. 2) Projectdoelen kunnen worden afgestemd met de business doelen. 3) De scope van het project vast te leggen. 4) Door de visie te communiceren met de betrokkenen dient iedereen hetzelfde doel. De geanalyseerde aanpakken zijn weergegeven in figuur 2.



FIGUUR 2: HET DEFINIËREN VAN EEN VISIE

DOEL 2: PROCES VAN HET DEFINIËREN EN BEHEREN VAN DE REQUIREMENTS VASTSTELLEN

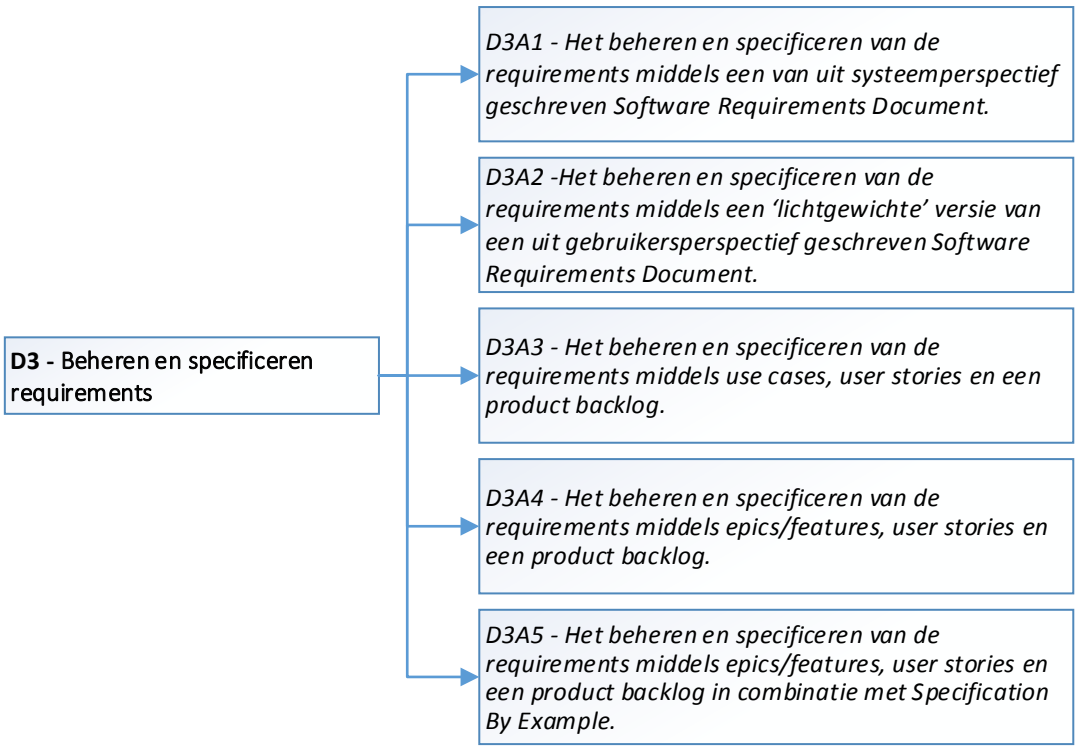
Nadat de productvisie en het systeem globaal zijn gedefinieerd in het visiedocument, dient men een algemene requirements-specificatie-procesaanpak te kiezen. Er wordt gesproken over een procesaanpak omdat deze keuze het gehele requirements-proces beïnvloedt en een aanpak in deze sectie niet direct resulteert in de productie van een artefact. De geanalyseerde procesaanpakken zijn weergegeven in figuur 3.



FIGUUR 3: PROCES VAN HET DEFINIËREN EN BEHEREN VAN DE REQUIREMENTS VASTSTELLEN

DOEL 3: BEHEREN EN SPECIFICEREN REQUIREMENTS

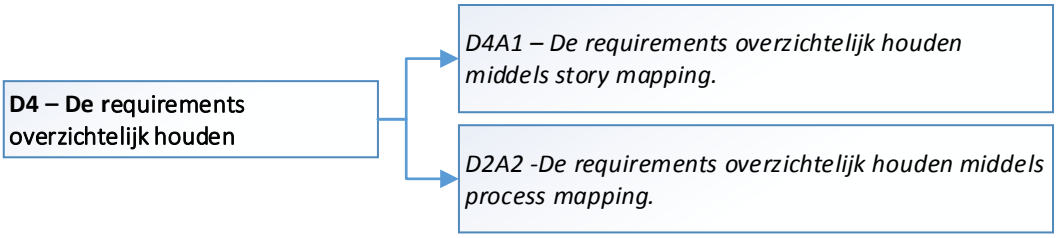
Nadat de visie en de requirements-specificatie-procesaanpak gekozen zijn, moet er worden vastgesteld op welke manier de requirements worden gespecificeerd en worden beheerd gedurende het project. Het beheren en specificeren van requirements is noodzakelijk omdat de gewenste functionaliteiten, voordat deze daadwerkelijk gerealiseerd kunnen worden, gedetailleerder moeten worden uitgewerkt. Het beheren van de requirements is vereist om de (gerealiseerde) requirements inzichtelijk en overzichtelijk te houden. De geanalyseerde aanpakken zijn weergegeven in figuur 4.



FIGUUR 4: BEHEREN EN SPECIFICEREN REQUIREMENTS

DOEL 4: DE REQUIREMENTS OVERZICHTELIJK HOUDEN

Aan de start van elk software-ontwikkelingsproject worden er grote hoeveelheden requirements gespecificeerd. Een geprioriteerde lijst van requirements, middels een *backlog* of een ander artefact, helpt het team te begrijpen welke functionaliteiten er in de toekomst gerealiseerd moeten worden, maar zegt weinig over het (huidige) systeem zelf. De ontwikkeltaken (*user stories*) op de *backlog* zijn versplinterd, en gedurende het project worden de requirements aangevuld en/of gewijzigd, hierdoor kan men op den duur het overzicht van het te bouwen systeem kwijt kan raken. Om dit te voorkomen zijn er een tweetal aanpakken geanalyseerd, deze zijn weergegeven in figuur 5.

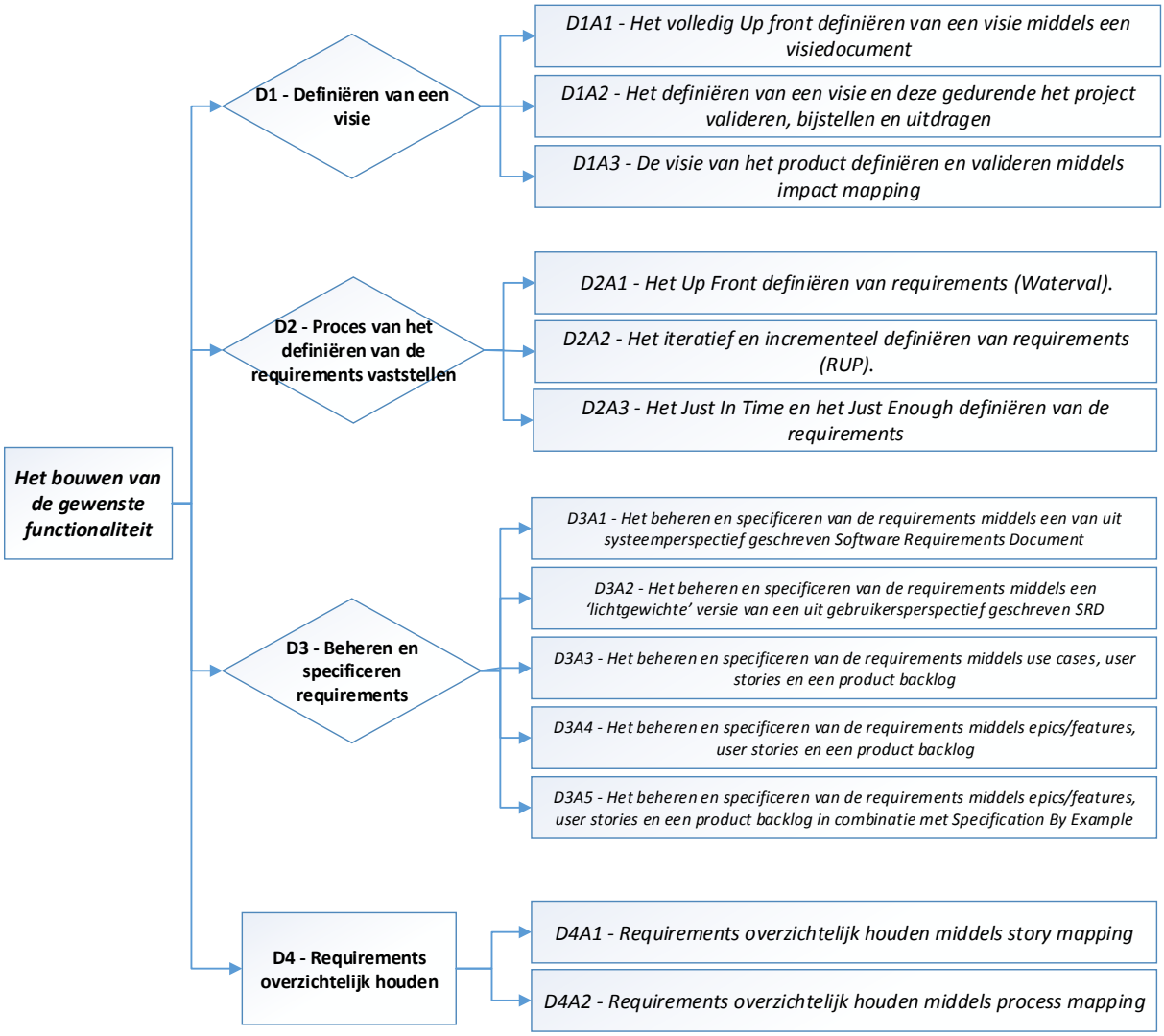


FIGUUR 5: DE REQUIREMENTS OVERZICHTELIJK HOUDEN

De aanpakken in deze sectie brengen geen risico's met zich mee, dit omdat het gaat om additionele aanpakken die, indien ze correct worden toegepast, enkel positieve invloed kunnen hebben op de kwaliteit van de software.

HET MODEL GEVISUALISEERD

Het gevisualiseerde model, de samenvoeging van alle doelen inclusief mogelijk aanpakken, zijn weergegeven in figuur 6.



FIGUUR 6: HET MODEL GEVISUALISEERD

DE MOGELIJK TE KIEZEN AANPAKKEN BESCHREVEN

DOEL 1: DEFINIËREN VAN EEN VISIE

Aanpak 1 - Het Up Front definiëren van een visie middels een visiedocument.	
Eigenschap	*Beschrijving*
Aanpaknaam	Het *Up Front* definiëren van een visie middels een visiedocument.
Aanpak Code:	D1A1
Aanpak Type:	Lineair (sequentieel), iteratief en incrementeel.
Beschrijving:	Het Up Front definiëren van een visie houdt in dat er, voorafgaand aan het project, door de stakeholders een visiedocument wordt opgesteld. Het visiedocument beschrijft de gezamenlijke visie van de opdrachtgever en opdrachtnemer met betrekking tot het project. In het visiedocument worden de belanghebbende toegelicht, de gewenste producteigenschappen beschreven en wordt er een planning en kostenschatting gegeven. Het visiedocument is het eerste document waarin beschreven wordt waaraan het systeem moet gaan voldoen en dient als input voor de (gedetailleerdere) requirements documenten zoals het software-architectuur document.
Toepasbaar in:	I. Projecten waar eenmalig overeenstemming noodzakelijk is, bijvoorbeeld bij projecten met veel stakeholders, en gedurende het project de visie minder belangrijk is. II. Projecten waarin de visie algemeen bekend is bij alle betrokkenen. III. Projecten waar voldoende resources beschikbaar zijn en het project geen tijdsdruk kent zodat alle gespecificeerde functionaliteiten in het visiedocument volledig gerealiseerd kunnen worden.
Toepassingseisen	*geen*
***Artefacten*:**	▪ Het visiedocument
***Kwaliteitsanalyse*:**	Het *Up Front* definiëren van visie, met bijbehorende planning, kostenschatting en gewenste functionaliteiten, is uitermate geschikt om te gebruiken als middel om met stakeholders te communiceren over het te bouwen systeem. Zodoende kan er door de stakeholders een (formeel) eenmalig akkoord gegeven worden en kan de volgende stap in het proces worden gezet (Nieboer, 2014). De impact van het *Up Front* definiëren van een visie is erg afhankelijk van welke mate het visiedocument gedurende het project bindend is. Wordt ervoor gekozen om een gedetailleerde planning en kostenschatting *Up Front* op te stellen, dan heeft dit bij langdurende projecten later in het proces nadelige gevolgen doordat er gewenste functionaliteit over het hoofd gezien is, de gewenste functionaliteit toch complexer blijkt te zijn of de wensen van de stakeholders gedurende het project veranderen (Lukassen, 2014). Het visiedocument verdwijnt, indien deze volledig *Up Front* is gedefinieerd, in veel gevallen in de figuurlijke 'la' en wordt gedurende het project niet meer ingezien. De visie wordt dus niet gebruikt als richtlijn voor de te nemen beslissingen gedurende het project, dit kan leiden tot software die niet in lijn is met de bedrijfsdoelen (Lukassen, 2014). Tevens raakt zodoende de visie in vergetelheid, of is de visie simpelweg niet bekend bij nieuwe teamleden, wat kan resulteren in gebrek aan focus bij de betrokkenen (Jansen, 2014) (Lukassen, 2014).

Aanpak 2 – Het definiëren van een visie en deze gedurende het project valideren, bijstellen en uitdragen middels een Product Vision Board	
Eigenschap	*Beschrijving*
Aanpaknaam	Het definiëren van een visie en deze gedurende het project valideren, bijstellen en uitdragen middels een *Product Vision Board.*
Aanpak Code:	D1A2
Aanpak Type:	Agile
Beschrijving:	Het definiëren van een visie en deze gedurende het gehele project uitdragen houdt in dat de visie niet eenmalig wordt gebruikt als projectdocument om bijvoorbeeld een volgende stap te zetten in het proces (D1A1), maar gedurende het project als hulpmiddel wordt gebruikt om de visie uit te dragen en de gerealiseerde functionaliteit te valideren. Er wordt niet meer gesproken over het *Up Front* definiëren van een 'vaststaande' visie maar over flexibele visie die gedurende het project, indien noodzakelijk, kan worden bijgesteld. De visie wordt gedurende het project constant gevalideerd door alle betrokkenen. Uiteraard wordt de visie nog wel *Up Front* gedefinieerd, dit om de voordelen van het proces van het *Up Front* definiëren van een visie te behouden.
Toepasbaar in:	I. Projecten waar de stakeholders geen behoeften hebben aan een formeel gedefinieerde visie, inclusief een planning en kostenschatting . II. Grote projecten waarin elk team aan een 'eigen 'stukje functionaliteit werkt, en/of er veel teamwisselingen zijn, waardoor de *Big Picture* verloren gaat. III. Projecten waar de wensen, en dus de visie, van de stakeholders gedurende het project kan veranderen.
Toepassingseisen:	*Geen*
Artefacten:	▪ *The Product Vision Board* (eventueel i.c.m. het *Product Canvas Board*) ▪ *Project/Iteration Kick-off meetings*
Kwaliteitsanalyse:	Door gebruik te maken van deze aanpak wordt de visie van een project getransformeerd van een statisch document naar een visuele weergave die ten alle tijden door de betrokkenen kan worden ingezien. Doordat de visie visueel zichtbaar is, kunnen ideeën en nieuwe inzichten met het team en de stakeholders besproken worden. "Pas als erover gediscussieerd wordt komen de exacte requirements boven tafel" (Nieboer, 2014) . Doordat op het *Product Vision Board* niet alleen de visie staat beschreven, maar ook nadrukkelijker wordt nagedacht over de *needs*, het te bouwen product en uiteindelijk de meerwaarde die het product zou moeten gaan opleveren voor de eindgebruiker, wordt door de discussies de te realiseren functionaliteit al in een vroeg stadium gevalideerd door de betrokkenen. Door het informele karakter en de minimale specificatie van deze aanpak stelt het de betrokkenen ook in staat om tussentijds in het project, indien er is gebleken dat een gerealiseerde oplossing niet resulteert in het gewenste resultaat, de visie tussentijds gemakkelijk aan te passen. Het gebruiken van deze aanpak resulteert daarom in een hogere functionele kwaliteit en in minder risico dat het product niet binnen de vastgestelde tijd en budget wordt afgerond. Tevens zorgt de visualisatie van de visie ervoor dat gedurende het project de betrokkenen gefocust blijven en er minder specificatie noodzakelijk is, dit doordat het doel (*Big Picture*) constant zichtbaar is voor alle betrokkenen (Jansen, 2014). Toch kan deze aanpak niet in alle projecten worden toegepast, dit door het deels informele karakter van de aanpak waardoor stakeholders het idee kunnen krijgen dat er niks formeels is vastgelegd waar in een later stadium op terug kan worden gevallen.

Aanpak 3 – De visie van het product definiëren en valideren middels impact mapping	
Eigenschap	*Beschrijving*
Aanpaknaam	*De visie van het product definiëren en valideren middels impact mapping*
Aanpak Code:	D1A3
Aanpak Type:	Agile
Beschrijving:	Het ontwikkelen van de visie middels *impact mapping* is een aanpak die de betrokkenen in staat stelt om de scope van het te bouwen systeem af te leiden van de bedrijfsdoelen en aannames over het gewenste resultaat snel te valideren. In *impact mapping* staan de volgende vragen centraal: - Waarom doen we dit? (bedrijfsdoelen) - Wie is er bij betrokken bij dit doel? (actoren) - Hoe kunnen de actoren ons helpen bij het behalen van de doelen? (oplossingsrichting) - Wat kunnen wij als organisatie doen ter ondersteuning van deze oplossingsrichting (Scope & features)
Toepasbaar in:	I. In innoverende projecten waarin een nieuwe *business case* wordt uitgeprobeerd en men niet weet hoe het door de klant en/of de markt ontvangen wordt. II. Projecten met een beperkt budget en het niet bekend is welke oplossingsmogelijkheid leidt tot het gewenste resultaat. III. Projecten waar meerdere oplossingsmogelijkheden zijn, maar een snelle *time-to-market* vereist is.
Toepassingseisen:	*Team*: ✓ Het team moet flexibel zijn, er moet immers vaak geschakeld worden tussen de verschillende mogelijkheden. ✓ Het team moet in staat zijn om met de stakeholders de mogelijkheden snel te kunnen valideren. *Stakeholders:* ✓ Stakeholders moeten beschikbaar zijn waar nodig, dit om het resultaat van de mogelijkheden en/of aannames snel te kunnen valideren. *Omgeving*: ✓ In het project moeten er korte iteraties mogelijk zijn, dit doordat het resultaat van mogelijkheden en/of aannames zo snel mogelijk gevalideerd moeten worden.
Artefacten:	▪ *Impact map*
Kwaliteitsanalyse	In veel projecten is er niet voldoende tijd en geld beschikbaar om alle gewenste functionaliteiten te realiseren en is het niet bekend op welke functionaliteit de gebruiker en/of de markt zit te wachten (Nieboer, 2014). Doordat de *impact m*ap de verschillende mogelijkheden visualiseert kan er, indien een bepaalde mogelijkheid niet resulteert in het gewenste resultaat, er snel overgestapt worden op één van de andere gespecificeerde mogelijkheden (Nieboer, 2014). Doordat er geen onnodige functionaliteit (*waste*) wordt gerealiseerd, dit is immers geconcludeerd bij de validatie, kan er meer tijd worden besteedt aan het bouwen van kwalitatief hoge functionele en technische kwaliteit software en blijft het project binnen de vastgestelde tijd en planning. Een ander voordeel van *impact mapping* is dat er vanuit de bedrijfsdoelen van het bedrijf wordt geredeneerd, de *impact map* begint immers met de vraag: '*Waarom doen we dit*'. Hierdoor blijven de betrokkenen, en dus ook de functionaliteiten, altijd in lijn met de bedrijfsdoelen en hierdoor is er een grotere kans dat de software voldoet aan de wensen van de eindgebruiker.

DOEL 2: PROCES VAN HET DEFINIËREN VAN DE REQUIREMENTS VASTSTELLEN

Procesaanpak 1 - Het Up Front definiëren van requirements (Waterval).	
Eigenschap	*Beschrijving*
Aanpaknaam	Het *Up Front* definiëren van requirements (Waterval).
Aanpak Code:	D2A1
Aanpak Type:	Lineaire (sequentieel)
Beschrijving:	Het *Up Front* definiëren van de requirements houdt in dat, alvorens er wordt ontwikkeld, alle wensen van de opdrachtgever worden achterhaald en vastgelegd. De requirements worden gedurende de daaropvolgende fasen ontworpen, gerealiseerd, getest en uiteindelijk uitgerold in productie. Het tussentijds wijzigen van de requirements is *by default* niet toegestaan en resulteert en het opnieuw doorlopen van fasen.
Toepasbaar in:	I. Projecten waar doormiddel van regelgeving wordt afgedwongen om de requirements Up Front te definiëren. Een voorbeeld van strenge regelgeving is de gezondheidszorg, waar kwaliteits-en risicomanagement aanpakken verplicht zijn, zoals de ISO 13485 en ISO 14971 normen (Amrit, et al., 2012). II. De tijdsduur van het project kort is en alle wensen van de opdrachtgever helder zijn en vaststaan.
Toepassingseisen:	<u>*Team*</u>*:* ✓ De technologie is bekend. ✓ Er voldoende *resources* beschikbaar zijn met een hoge mate van ervarenheid. <u>Product</u>: ✓ De wensen van de opdrachtgever zijn bekend, zijn duidelijk, staan vast en de productdefinitie is stabiel. <u>*Omgeving*</u>: ✓ Een lange aanloopperiode, waarin geen bedrijfswaarde wordt opgeleverd, geen probleem vormt voor de stakeholders.
Kwaliteitsanalyse:	Het *Up Front* definiëren van de requirements brengt in de praktijk veel risico's met zich mee, dit blijkt zowel uit de literatuur (Standish Group, 1994) alsmede uit de interviews met de experts (Nieboer, 2014) (Jansen, 2014). Bij deze aanpak wordt er vanuit gegaan dat alle specificaties *Up Front* gedefinieerd kunnen worden, maar in de praktijk blijkt dat dit vooral aannames zijn die uiteindelijk niet blijken te kloppen. De reden hiervan is, dat de opdrachtgever vaak niet exact weet wat hij/zij wilt en de wensen gedurende het project meermaals veranderen. Doordat de planning en kostenschatting zijn gebaseerd op de oorspronkelijke gedefinieerde functionaliteit, worden weinig tot geen projecten binnen de vastgestelde kaders opgeleverd. Op functioneel gebied resulteert dit in veel gevallen, door de harde deadlines, in kwalitatief slechte software die relatief veel fouten bevat. Toch is het *Up Front* definiëren van de requirements niet in alle situaties onverstandig, zo zijn er situaties waarin een hoge mate van vooraf gedefinieerde requirements verplicht is, of als een andere aanpak alleen maar voor overhead zorgt.

Procesaanpak 2 - Het iteratief en incrementeel definiëren van requirements (RUP).	
Eigenschap	*Beschrijving*
Aanpaknaam	Het iteratief en incrementeel definiëren van de requirements (RUP).
Aanpak Code:	D1A2
Aanpak Type:	Iteratief en Incrementeel
Beschrijving:	Het iteratief en incrementeel definiëren van de requirements houdt in dat de requirements middels iteraties steeds gedetailleerder worden gedefinieerd. Na elke iteratie vindt er overleg plaats tussen de stakeholders zodat de requirements kunnen worden gevalideerd. Tevens wordt het systeem opgesplitst in meerdere incrementen die gezamenlijk de complete functionaliteit bevatten. Bij iteratief en incrementeel definiëren van de requirements zijn de requirements al voor het grootste gedeelte (± 80 %) gedefinieerd, alvorens men begint aan de daadwerkelijke realisatie van de (grote brokken) functionaliteit.
Toepasbaar in:	I. Projecten waar het *Up Front* definiëren teveel risico's met zich meebrengt en een *Agile* aanpak door de betrokkenen 'wordt geclassificeerd' als een te weinig gestructureerde aanpak. II. Projecten waar het te bouwen systeem een grote mate van (externe) afhankelijkheid bevat en hierdoor de bouw vertraging op kan lopen. III. Als de te realiseren functionaliteiten dermate complex zijn dat deze een relatief lange voorbereidingstijd vergen.
Toepassingseisen:	*Product*: ✓ De wensen van de opdrachtgever relatief duidelijk en stabiel zijn. *Omgeving*: ✓ Een lange aanloop periode, waarin (bijna) geen bedrijfswaarde wordt opgeleverd, geen probleem vormt voor de stakeholders.
Kwaliteitsanalyse	Deze aanpak wordt vaak gezien als en tussenweg tussen de lineaire en sequentiële (waterval) en de *agile* manier van het definiëren van requirements. Doordat de requirements niet meer *Up Front* worden gedefinieerd, maar iteratief gedurende het project, kunnen de gespecificeerde requirements gedurende het project gewijzigd worden als tijdens de validatie blijkt dat de functionaliteit niet voldoet aan de verwachtingen. Incorrectheden in de requirements worden zodoende sneller gevalideerd. Het risico dat de software uiteindelijk niet voldoet aan de wensen van de eindgebruiker wordt hierdoor sterk verminderd. Toch wordt er, ten opzichte van de *agile* aanpak, nog relatief veel *Up Front* gedocumenteerd wat uiteindelijk kan resulteren in een *waste* van tijd en *resources*. Het kan dus voorkomen dat de positieve invloed van deze aanpak, op financieel-en planningsgebied, teniet wordt gedaan door de overmatige documentatie werkzaamheden. Het voordeel van deze aanpak is dat, als men enigszins zeker weet welke functionaliteiten er gerealiseerd dienen te worden, in de eerste fasen van het project alle benodigde informatie kan worden verzameld en/of afhankelijke systemen geconfigureerd kunnen worden. Dit zodat de ontwikkelaars in de realisatiefase direct aan de slag kunnen gaan en niet worden opgehouden door interne en/of externe factoren (Lukassen, 2014).

***Procesaanpak 3* - Het Just In Time en het Just Enough definiëren van de requirements (Agile).**	
Eigenschap	*Beschrijving*
Aanpaknaam:	Het *Just In Time* en het *Just Enough* definiëren van de requirements (Agile).
Aanpak Code:	D1A3
Aanpak Type:	Agile
Beschrijving:	Het *Just In Time* definiëren van de requirements houdt in dat requirements pas gespecificeerd worden wanneer daar, ten behoeve van het bereiken van een business doel, behoefte aan is. Het *Just Enough* principe betekent dat er precies genoeg gedetailleerde requirements worden gespecificeerd dan dat er nodig is op dat moment voor het bereiken van de bedrijfsdoelen. Het *Just In Time* en *het Just Enough* definiëren van de requirements kan in verschillende mate worden toegepast. Het uitwerken van de requirements, voorafgaand aan de daadwerkelijk realisatie-iteratie, kan bijvoorbeeld 2 iteraties van tevoren plaatsvinden, 1 iteratie van tevoren plaatsvinden of zelfs geïntegreerd zijn in de daadwerkelijk realisatie-iteratie.
Toepasbaar in:	I. In projecten waar nieuwe software wordt ontwikkeld en daardoor de stakeholders nog niet exact de wensen kunnen definiëren. II. In projecten waar men snel in wilt kunnen springen op markt veranderingen (dynamische markt). III. In projecten waar software wordt ontwikkeld die (onbekende) complexe bedrijfsprocessen bevat. IV. In projecten waar software wordt ontwikkeld middels een 'nieuwe' technologie waarover weinig tot geen (interne) kennis voorhanden is. V. In projecten waar een snelle *time-to-market* is vereist (deadlines).
Toepassingseisen:	*Stakeholders:* ✓ Stakeholder moet beschikbaar zijn wanneer nodig en moet actief participeren gedurende het project. ✓ *Product owner* (afgevaardigde) is beschikbaar waar nodig en is in staat en is bevoegd om beslissingen te nemen. *Team:* ✓ Het team moet goed op de hoogte zijn van de doelen die de organisatie nastreeft. ✓ Het team moet in staat zijn om met verandering om te gaan. Omgeving: ✓ Is het efficiëntst als alle betrokkenen gepositioneerd zijn in hetzelfde gebouw, liefst in dezelfde ruimte.
Kwaliteitsanalyse:	Het *Just In Time* en het *Just Enough* definiëren van de requirements is uitermate geschikt voor projecten waar alles in het teken staat van zoveel mogelijke waarde opleveren binnen een zo kort mogelijk tijd. Doordat de requirements *Just In Time en het Just Enough* worden gespecificeerd is het risico dat een volledige gespecificeerde functionaliteit, indien de prioriteiten veranderen, uiteindelijk niet gerealiseerd wordt zeer klein. Dit voorkomt *waste* en komt zowel de technische als de functionele kwaliteit van de software ten goede. De mate van *waste* die, na wijziging van de wensen en/of prioriteit optreedt, is afhankelijk van de tijd die tussen het gedetailleerd specificeren van de requirements zit en de daadwerkelijke realisatie-iteratie. Deze aanpak heeft gevolgen voor de planning , voorafgaand aan het project kan er namelijk geen gedetailleerde planning en kostenschatting voor de gewenste functionaliteit worden afgegeven. De tijd en kosten staan vast, de te realiseren functionaliteiten zijn variabel. Dit betekent niet dat er niet kan worden geschat hoeveel tijd en geld het realiseren van een systeem gaat kosten, hiervoor zijn andere technieken beschikbaar. Een risico van deze aanpak is, dat bij plotselinge afwezigheid van stakeholders of door onvoorziene omstandigheden, het kan voorkomen dat de werkzaamheden binnen een iteratie stilvallen, dit doordat de requirements nog niet voldoende zijn uitgewerkt. De overige risico's van *agile* specificeren zijn toegelicht in *Bijlage 2*.

	Indien deze aanpak wordt toegepast, en alle betrokkenen geloven in deze aanpak, heeft deze een zeer positieve invloed op de algehele kwaliteit van het systeem en wordt het risico dat het systeem niet binnen de afgesproken kaders wordt opgeleverd sterk verlaagd.

DOEL 3: BEHEREN EN SPECIFICEREN REQUIREMENTS

Aanpak 1 - Het beheren en specificeren van de requirements middels een van uit systeemperspectief geschreven Software Requirements Document	
Eigenschap	*Beschrijving*
Aanpaknaam	Het beheren en specificeren van de requirements middels een van uit systeemperspectief geschreven Software Requirements Document.
Aanpak Code:	D3A1
Aanpak Type:	Lineair (sequentieel)
Beschrijving:	Het beheren en specificeren van de requirements middels een *Software Requirements Document* (SRD) is afkomstig uit de traditionele ontwikkelmethodieken en houdt in dat alle eisen, waaraan de software moet voldoen, in de vorm van 'wat moet het systeem kunnen' statements worden beschreven. Het document bevat een beschrijving van de gewenste functionaliteiten, externe koppelingen en niet-functionele eisen als performance en security en ontwerp beperkingen zoals standaarden waar rekening mee gehouden moet worden. Alle requirements die staan beschreven in het *SRD*, worden ook in dit document beheerd en geprioriteerd.
Toepasbaar in:	I. Projecten waarin weinig tot geen overleg over de requirements noodzakelijk is. II. Projecten waar software wordt gerealiseerd waar de requirements duidelijk en stabiel zijn. III. Projecten waarin *Embedded software* wordt gerealiseerd.
Toepassingseisen:	<u>*Team:*</u> ✓ Extreem gedisciplineerd team, door vele requirements vastgelegd in natuurlijke taal vergt het discipline om de documenten *up-to-date* te houden met de programmacode, de testen en de documentatie.
Artefacten:	▪ Software Requirements Document
Kwaliteitsanalyse:	De requirements in het document zijn beschreven als mogelijkheden en beperkingen van het systeem, hierdoor wordt de nadruk op het systeem gelegd en niet op de gebruikers-interactie van het systeem of op de bedrijfsdoelen. Hierdoor worden er aannames gemaakt over wat de gebruikers willen. Vervolgens wordt er, op basis van de requirements, exact gemaakt waar de gebruiker om vroeg, maar wordt er software opgeleverd die niet de beoogde meerwaarde bevat voor de gebruiker. Tevens is het, door de statische requirements, moeilijk om te communiceren en vooral te discussiëren over de functionaliteiten met de betrokkenen. Bovengenoemde redenen resulteren in een systeem met een zeer lage functionele kwaliteit. Het hebben van een document waarin alle requirements zijn gespecificeerd, en ook in worden beheerd, resulteert in software die erg moeilijk te onderhouden en uit te breiden is, indirect heeft dit dus nadelige gevolgen voor de functionele en technische kwaliteit van de software. De reden hiervan is dat het document zo in omvang toeneemt dat er geen relatie is met elementen uit andere documenten en hierdoor het wijzigen van requirements erg tijdrovend is. Op den duur kan dit zelfs onmogelijk worden, elke wijziging in de code moet immers worden doorgevoerd in de specificaties, testen en de documentatie en visa versa.

Aanpak 2 - Het beheren en specificeren van de requirements middels een 'lichtgewichte' versie van een uit gebruikersperspectief geschreven Software Requirements Document	
Eigenschap	*Beschrijving*
Aanpaknaam	Het beheren en specificeren van de requirements middels een 'lichtgewichte' versie van een uit gebruikersperspectief geschreven *Software Requirements Document*
Aanpak Code:	D2A1
Aanpak Type:	Iteratief en Incrementeel
Beschrijving:	Het beheren en specificeren van de requirements middels een 'lichtgewichte' versie van het Software Requirements Document (SRD) is afkomstig uit de iteratieve en incrementele ontwikkelmethodieken (RUP) en wordt gekenmerkt door dat de requirements worden geschreven vanuit het gebruikersperspectief. Dit om het systeem beter aan te laten sluiten op de wensen van de gebruiker. In de *'lichtgewichte'* versie *van het* SRD wordt ook gebruik gemaakt van visuele modellen om de context, (gebruikers) flow en de functionaliteiten overzichtelijk weer te geven.
Toepasbaar in:	I. Projecten waar software wordt gerealiseerd waar de requirements grotendeels duidelijk en stabiel zijn. II. Projecten waar de omgeving (stakeholder) nog een bepaalde mate van 'zekerheid' en 'formaliteit' eist en dit middels uitgebreide documentatie wordt ingevuld.
Toepassingseisen	*Geen*
Artefacten:	▪ Software Requirements Document ▪ RUP-artefacten (*zie: http://www-01.ibm.com/software/rational/rup/)*
Kwaliteitsanalyse	In de meer 'lichtgewichte' versie van het SRD wordt er middels diverse artefacten geprobeerd om het gat tussen de requirements en de gebruikers van het systeem te dichten. Doordat de requirements uit gebruikersperspectief zijn beschreven kan er met de stakeholders over de requirements worden gediscussieerd, zodoende worden de exacte wensen van de stakeholders sneller achterhaald. Het gebruik van uit gebruikersperspectief geschreven requirements verhoogd dus de functionele kwaliteit van de software en heeft hierdoor ook, doordat er minder wijzigingen achteraf moeten plaatsvinden, een positieve invloed op de kosten en planning van het project. Het gevaar van deze aanpak is dat de requirements, met behulp van alle artefacten die worden voorgeschreven door bijvoorbeeld RUP, resulteren in vele pagina's tellende documenten. De oorzaak hiervan is dat de standaard templates willen dat de requirements dermate gedetailleerd worden vastgelegd, dat elke gewenste functionaliteit in de toekomst, inclusief uitzonderingen op de functionaliteit, wordt vastgelegd. De gevolgen hiervan zijn dat de documenten te groot, te complex en te proces-georiënteerd worden en hierdoor de aanpak gaat lijken op de traditionele requirements-specificatie-aanpak, zoals beschreven in *D2A1*.

Aanpak 3 - Het beheren en specificeren van de requirements middels use cases, user stories en een product backlog	
Eigenschap	*Beschrijving*
Aanpaknaam	Het beheren en specificeren van de requirements middels use cases, user stories en een product backlog.
Aanpak Code:	D3A3
Aanpak Type:	Iteratief en incrementeel / Agile
Beschrijving:	In deze aanpak worden de requirements voor het grootste gedeelte vastgelegd in *use cases* (*UC's*). Omdat in de meeste gevallen de *UC's* te groot zijn om in een iteratie te realiseren, worden deze opgesplitst in kleinere brokken functionaliteit middels *user stories*. Per *user story* kunnen er acceptatiecriteria gedefinieerd worden en kunnen waar nodig additionele (requirements) documenten gekoppeld worden. De *user stories* worden op de *product backlog* beheerd en worden uiteindelijk in een iteratie gerealiseerd.
Toepasbaar in:	I. Projecten waar er geen contract-verhouding noodzakelijk is tussen de opdrachtgever en de ontwikkelende partij. II. Projecten waar veel stakeholders bij betrokkenen zijn, wat resulteert in conflicterende requirements. III. Projecten waar software wordt ontwikkelt die dermate complex is, door bijvoorbeeld vele externe koppeling en/of bestaande requirements, dat het niet afdoende is om alleen epics/features en user stories te gebruiken om de requirements te specificeren. IV. Projecten die worden uitgevoerd in grote organisaties waar *face-to-face communicatie* met alle betrokkenen niet mogelijk is.
Toepassingseisen:	*Geen*
Artefacten:	▪ Product *backlog* ▪ Iteratie *backlog* ▪ *Use case (Use case 2.0)* ▪ *User story*
Kwaliteitsanalyse:	Deze aanpak wordt in veel hedendaagse projecten gebruikt. De globale functionaliteiten worden vastgelegd in use cases, dit om de gebruikers-flow door het gehele systeem te kunnen beschrijven en zodoende met de stakeholders over de gewenste functionaliteiten te discussiëren. De nadelen van *use cases*, zoals dat het veel tijd kost om een use cases met bijbehorende scenario's te schrijven, zijn hier in mindere mate van toepassing doordat alleen de essentiële informatie wordt vastgelegd in de use cases. De traditionele documenten, zoals het *SRD* worden niet meer (of gedeeltelijk) gebruikt omdat er niet meer wordt gesproken over een opdrachtgever - opdrachtnemer contract maar over een samenwerkingsverband. Doordat de functionaliteiten, beschreven in de *use cases*, middels *user stories* kunnen worden opgesplitst, kunnen de *user stories* op de *backlog* eenvoudig worden beheerd en geprioriteerd. De opsplitsing naar *user stories* betekent meer flexibiliteit, dit omdat de *user stories* per iteratie worden ingepland en het veranderingen in prioriteit mogelijk maakt. Toch brengt het gebruik van *user stories* ook risico's met zich mee. Het grootste risico van deze aanpak is dat de gewenste functionaliteiten overzichtelijk staan beschreven in de *use cases*, maar dit bij het opsplitsen van de *use case* naar *user stories* verloren gaat. Dit kan resulteren in de volgende problemen: - *User stories* zonder duidelijke waarde voor de gebruiker. - Een oplevering waar slechts delen van een *use cases* zijn gerealiseerd en niemand exact weet welke. - Problemen bij het testen van de functionaliteit doordat het testteam de *user stories* niet afzonderlijk kan testen omdat afhankelijke *user stories* nog niet gerealiseerd zijn. - Wijzigingen in *user stories*, of zelfs in de code, die niet worden doorgevoerd in de desbetreffende *use case* waardoor de documentatie inconsistent wordt. De invloed op de kwaliteit van de software, indien deze aanpak wordt gebruikt, is dus erg afhankelijk van hoe deze aanpak wordt toegepast, wordt gecoördineerd en of het team de discipline kan opbrengen om de requirements, de code en de documentatie consistent te houden.

Aanpak 4 - Het beheren en specificeren van de requirements middels epics/features, user stories en een product backlog.	
Eigenschap	*Beschrijving*
Aanpaknaam	Het beheren en specificeren van de requirements middels epics/features, user stories en een product backlog.
Aanpak Code:	D3A4
Aanpak Type:	Agile
Beschrijving:	De requirements worden vastgelegd middels *epics/features*, dit zijn beschrijvingen van grote brokken functionaliteit die het systeem moet gaan bevatten. Deze *epics/features* worden, naarmate het proces vordert, 'opgebroken' in kleinere stukken functionaliteit met behulp van *user stories*. Additionele documenten zoals een schermontwerp of een *procesflow*, worden aan de *user story* gekoppeld en middels acceptatiecriteria wordt er vastgelegd aan welke eisen de uiteindelijke oplossing moet gaan voldoen. De *user stories* worden beheerd op de *product backlog* en kunnen in een gewenste iteratie worden gerealiseerd. Kenmerkend aan deze aanpak is dat de requirements minimalistisch worden gespecifieerd, is er iets niet duidelijk dan wordt dit middels directe communicatie met de stakeholders verholpen.
Toepasbaar in:	I. Projecten waar software wordt ontwikkeld met relatief stabiele teams die gepositioneerd zijn op dezelfde locatie. II. Projecten waar software wordt ontwikkeld door een relatief 'klein' team waardoor *face-to-face communicatie* mogelijk is. III. Projecten waar relatief 'simpele' software wordt ontwikkeld met weinig tot geen niet-functionele requirements. IV. Projecten waar software wordt ontwikkeld waarvan de context van de te bouwen functionaliteit bekend is.
Toepassingseisen:	<u>*Team*</u>: ✓ Het team begrijpt de *big-picture* van het project. <u>*Stakeholders*</u>: ✓ Actieve stakeholder participatie gedurende het gehele project. ✓ *Product owner* (afgevaardigde) is beschikbaar waar nodig en is bevoegd om beslissingen te nemen. <u>*Omgeving:*</u> ✓ Naleving van (externe) regelgeving niet vereist is.
Artefacten:	▪ Product *backlog* ▪ Iteratie *backlog* ▪ *Epic / feature* ▪ *User story*
Kwaliteitsanalyse	Doordat de requirements minimalistisch worden gespecificeerd middels *epics/features* en uiteindelijk user stories, en er dus geen gedetailleerde beschrijving van de gebruikers-flow aanwezig is zoals in bijvoorbeeld use cases, voorkomt deze aanpak *waste* in het specificatieproces. Het schrijven van *use cases,* en bijbehorende scenario's, neemt veel tijd in beslag en vaak wordt er bij de daadwerkelijke realisatie van de functionaliteit er door de ontwikkelaar geen aandacht meer aan besteedt. Doordat de *epics/features middels user stories* worden opgesplitst, kunnen de *user stories* op de *backlog* eenvoudig worden beheerd en geprioriteerd. *Voor voordelen user stories zie aanpak D3A3.* Doordat de requirements minimalistisch worden gespecificeerd, wordt deze aanpak veelal gebruikt in kleine en overzichtelijke softwareontwikkelingsprojecten. De reden hiervoor is dat deze aanpak *by default* in complexe projecten te weinig houvast biedt voor de betrokkenen doordat er te weinig beschreven staat en directe communicatie niet altijd mogelijk is. Deze aanpak kan dus positieve invloed hebben op de kwaliteit van het systeem maar kan, indien het wordt toegepast in de verkeerde situaties, ook een negatieve invloed hebben op de kwaliteit van het product.

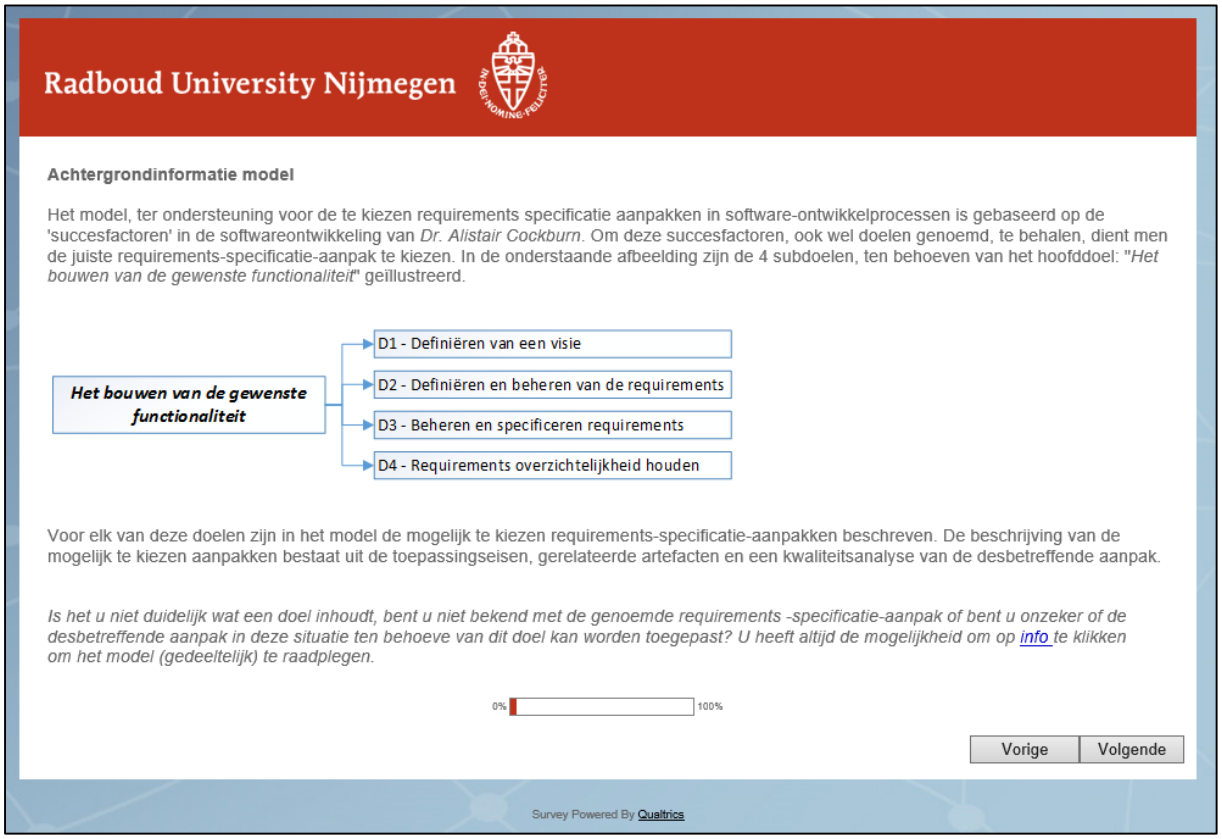
Aanpak 5 – Het beheren en specificeren van de requirements middels Specification By Example, user stories en een product backlog	
Eigenschap	*Beschrijving*
Aanpaknaam	Het beheren en specificeren van de requirements middels *Specification By Example*, *user stories* en een *product backlog*
Aanpak Code:	D4A6
Aanpak Type:	Agile
Beschrijving:	Waar in normale projecten de requirements worden vastgelegd in documenten, diagrammen of *use cases,* worden deze bij *Specification By Example (SbE)* vastgelegd in realistische voorbeeldscenario's uit de praktijk. De scenario's worden gezamenlijk gedefinieerd door de business en het team. Doordat de scenario's in *SbE* op een formele manier en volledig worden vastgelegd, dienen deze scenario's ook als startpunt voor de ontwikkeling, als automatisch testcases en door de algehele leesbaarheid ook als documentatie. *SbE* richt zich dus zowel op requirements, ontwikkeling, test en de documentatie van een systeem. De scenario's van *SbE* zijn leidend binnen het softwareontwikkelingstraject, er wordt gebruik gemaakt van *user stories* en een *backlog* om gedurende het (ontwikkel)proces het overzicht te behouden.
Toepasbaar in:	I. Grootschalige projecten met veel requirements, een complexe organisatie/domein waar functionele testen noodzakelijk zijn. II. Projecten waarin administratieve systemen worden ontwikkeld. III. Bedrijven waar men investeert in innovatie en men teams de ruimte geeft om innoverende aanpakken toe te passen.
Toepassingseisen	<u>*Team*</u>: ✓ Het team begrijpt de *big-picture* van het project. ✓ Het team is in staat om (formele) scenario's te definiëren. ✓ Het team bevat een hoge mate van analytisch vermogen. <u>*Stakeholders*</u>: ✓ De product owner (afgevaardigde) is in staat om, samen met het team, (formele) scenario's te definiëren
Artefacten:	▪ *.Feature file*
Kwaliteitsanalyse	Doordat natuurlijke taal ambigu is, en de opdrachtgever niet altijd helder voor ogen heeft welke functionaliteiten gewenst zijn, zit er vaak een (groot) verschil tussen datgene wat de klant voor ogen heeft, en hetgene wat er uiteindelijk wordt opgeleverd door het team. Met *SbE* wordt niet alleen de ambiguïteit van de requirements wegenomen, maar worden de requirements, doordat scenario's gezamenlijk met de business worden opgesteld, al in het beginstadium van het project gevalideerd. Tevens voelt, doordat het team wordt betrokkenen bij de requirements-specificatie, het team zich meer verantwoordelijk voor de te realiseren functionaliteit, wat resulteert in meer gefocuste en gemotiveerde teams. *SbE* heeft ook een positieve invloed op de technische kwaliteit van de software, dit omdat de specificatie, code en de testcases 1-op-1 overeenkomen. Zodoende kan er bij elke wijziging in de code, er direct gezien worden wat de impact is van deze wijziging. Het uitvoeren van een kleine wijziging, wat achteraf in de praktijk negatieve impact blijkt te hebben op bestaande functionaliteit, is onmogelijk indien er gebruik wordt gemaakt van *SbE*. Toch kan deze aanpak, voornamelijk op korte termijn, nadelige gevolgen hebben op de planning en kostenschatting van het project. Doordat *SbE* een andere manier van denken vergt van het team, kost het tijd voordat men deze aanpak beheerst. Tevens gaat er, in vergelijking met aanpak *D3A4*, meer tijd zitten in het specificeren van de requirements doordat elk mogelijk scenario uitgeschreven moet worden. In de praktijk blijkt echter dat deze tijd ruimschoots wordt gecompenseerd doordat men deze tijd anders zou besteden aan het uitvoeren van *change requests*. Een ander risico van deze aanpak is, doordat de requirements van het systeem worden opgesplitst in tientallen scenario's, men het overzicht (*big picture*) kwijtraakt. Dit vormt een groot risico omdat binnen deze aanpak de scenario's leidend zijn in het gehele project.Het gebruiken van één van de aanpakken zoals beschreven in D1 en/of D4 is dus noodzakelijk.

DOEL 4: DE REQUIREMENTS OVERZICHTELIJK HOUDEN

Additionele aanpak 1 – De requirements overzichtelijk houden middels story mapping	
Eigenschap	*Beschrijving*
Aanpaknaam	De requirements overzichtelijk houden middels *story mapping*
Aanpak Code:	D4A1
Aanpak Type:	Agile
***Beschrijving*:**	Een *story map* rangschikt en visualiseert de te realiseren functionaliteiten tot een model om een overzicht te generen van de totale (te realiseren) functionaliteit, identificeert gaten in de product *backlog* en stelt de betrokkenen in staat om releases op een holistisch manier te plannen.
Toepasbaar in:	I. Projecten waar, de overige gekozen requirements-specificatie-aanpakken, niet het gewenste overzicht bieden. II. Projecten waarin software vanaf scratch wordt gebouwd en waarbij de grote lijnen goed bewaakt moeten blijven. III. Projecten waar software wordt gebouwd waar een grote afhankelijkheid bestaat tussen de verschillende user stories. IV. Projecten met wisselende teamsamenstelling en/of wisselende teambezetting. V. Projecten waar zeer regelmatig releases plaatsvinden. VI. Projecten waar complexe gebruikersprocessen aanwezig zijn.
Toepassingseisen	*Geen*.
Artefacten:	▪ Story map
Kwaliteitsanalyse:	Door de opsplitsing van ontwikkeltaken (*user stories en/of scenario's*) komt het voor dat, door de incrementele oplevering, er op het eerste gezicht functionaliteit met veel *business value* wordt opgeleverd maar deze in de praktijk niet bruikbaar is doordat afhankelijkheden nog niet gerealiseerd zijn. Door deze aanpak te gebruiken wordt de functionele kwaliteit van de software verhoogd doordat er bewust wordt gekeken welke afhankelijkheden er binnen de software zijn en wat door de stakeholders wordt beschouwd als een 'werkende' functionaliteit. Deze informatie kan vervolgens worden gebruikt om releases op een zinvolle manier te plannen en uit te rollen naar de eindgebruikers. Een ander aspect wat invloed heeft op de kwaliteit van de software, wat geldt voor elke fysieke visualisatie, is dat de *story map* aan de muur voor iedereen zichtbaar is en hierdoor discussie over de functionaliteit bevordert en helpt de focus te behouden.

Additionele aanpak 2 – De requirements overzichtelijk houden middels process mapping	
Eigenschap	*Beschrijving*
Aanpaknaam	De requirements overzichtelijk houden middels *process mapping*
Aanpak Code:	D4A2
Aanpak Type:	Agile
Beschrijving:	De requirements overzichtelijk houden middels *process mapping* is een aanpak om de te realiseren functionaliteiten *(user stories)* op een overzichtelijke manier te rangschikken op bedrijfsproces. Met behulp van een *process map* wordt inzichtelijk aan welk deel van het (gehele) bedrijfsproces het team momenteel werkt, welke *user stories* uit het desbetreffende bedrijfsproces nog gerealiseerd moeten worden en welke afhankelijkheden er zijn met andere bedrijfsprocessen.
Toepasbaar in:	I. Projecten waar software wordt gebouwd waar een grote afhankelijkheid bestaat tussen de verschillende bedrijfsprocessen. II. Projecten waar zeer regelmatig releases plaatsvinden. III. Projecten waar software wordt ontwikkeld die complexe bedrijfsprocessen bevat. IV. Projecten met wisselende teamsamenstelling en/of wisselende teambezetting.
Toepassingseisen	*Geen.*
Artefacten:	▪ Process map
Kwaliteitsanalyse	Vaak, wanneer er agile-aanpakken worden gebruikt, hangt er in de kamer een *SCRUM-board* om te kunnen zien waar iedereen mee bezig is en welke *user stories* er nog gerealiseerd moeten worden. Toch, als je het *SCRUM-board* observeert, is het (voornamelijk voor de business) niet duidelijk wat exact de status is van het te ontwikkelen systeem. Vragen zoals: "*Welk onderdeel van het bedrijfsproces wordt in de huidige iteratie gerealiseerd"* en "*hoeveel iteraties zijn er nog nodig om het gehele bedrijfsproces op te leveren"*, kunnen niet worden beantwoordt. Een *process map* levert antwoord op deze vragen en is, zeker voor de business, van toegevoegde waarde in elk project. Doordat de bedrijfsprocessen duidelijk zichtbaar zijn, voor zowel de business als het team, verhoogd deze aanpak de betrokkenheid en de focus van alle betrokkenen en komt deze aanpak ten goede van de functionele kwaliteit van de software.

6. DE VRAGENLIJST


Radboud University Nijmegen
Achtergrondinformatie model
Het model, ter ondersteuning voor de te kiezen requirements specificatie aanpakken in software-ontwikkelprocessen is gebaseerd op de 'succesfactoren' in de softwareontwikkeling van Dr. Alistair Cockburn. Om deze succesfactoren, ook wel doelen genoemd, te behalen, dient men de juiste requirements-specificatie-aanpak te kiezen. In de onderstaande afbeelding zijn de 4 subdoelen, ten behoeven van het hoofddoel: "Het bouwen van de gewenste functionaliteit" geïllustreerd.
Het bouwen van de gewenste functionaliteit
D1 - Definiëren van een visie
D2 - Definiëren en beheren van de requirements
D3 - Beheren en specificeren requirements
D4 - Requirements overzichtelijkheid houden
Voor elk van deze doelen zijn in het model de mogelijk te kiezen requirements-specificatie-aanpakken beschreven. De beschrijving van de mogelijk te kiezen aanpakken bestaat uit de toepassingseisen, gerelateerde artefacten en een kwaliteitsanalyse van de desbetreffende aanpak.
Is het u niet duidelijk wat een doel inhoudt, bent u niet bekend met de genoemde requirements -specificatie-aanpak of bent u onzeker of de desbetreffende aanpak in deze situatie ten behoeve van dit doel kan worden toegepast? U heeft altijd de mogelijkheid om op info te klikken om het model (gedeeltelijk) te raadplegen.
0%
100%
Vorige
Volgende
Survey Powered By Qualtrics

Radboud University Nijmegen

***Case Study:* Adora - Het realiseren van een interactief en aanpasbaar sales-dashboard voor het management.**
Adora, opgericht in 1985, is al jaren marktleider op het gebied van schildersproducten. Door de groeiende markt (jaren 90) zijn er binnen korte tijd door heel Nederland filialen opgericht. Door deze exponentiële groei kon het destijds niet voorkomen worden dat elk bedrijf zijn eigen administratiesoftware aanschafte. De laatste jaren gaat het echter steeds slechter in de branche en is *Adora* haastig op zoek naar manieren om beter inzicht te krijgen in de prestaties (KPIs) van het bedrijf. De wens van *Adora* is dan ook om een interactief en aanpasbaar sales dashboard met bijbehorende KPIs van het bedrijf te laten bouwen. De exacte eisen van het dashboard zijn uitgewerkt in een twintig pagina's tellend document.

Omdat *Adora* zelf geen dataspecialisten in huis heeft, de data uit verschillende losstaande systemen gecombineerd moet worden en er bij eerdere (interne) pogingen data-kwaliteit issues waren, is er gekozen om het project uit te laten voeren door een klein extern team dat hier gespecialiseerd in is. *Adora* stelt de eis dat het project binnen 4 maanden is afgerond en het niet meer dan 40.000 euro gaat kosten. Tevens wilt *Adora* tussentijds op de hoogte gehouden worden van de voortgang van het project.

U bent de leidinggevende van het projectteam en uw team mag het bovenstaande project ten uitvoer brengen. Omdat dit voor project voor het management erg belangrijk is, is de hulp ingeroepen van een externe expert die geacht wordt om de meeste geschikte requirements-specificatie-aanpak te kiezen.

In de onderstaande vragen wordt u, per doel, gevraagd of u het eens bent met de gekozen aanpak van de expert. Is het u niet duidelijk wat een doel inhoudt, of bent u benieuwd naar de toepassingseisen of de kwaliteitsanalyse, dan kunt u op info klikken voor meer informatie.

Doel 1: Het definiëren van een visie (Info)

Om het bovenstaande doel te behalen, heeft de expert de onderstaande aanpak gekozen:

- [x] *Aanpak 1 - Het volledig 'Up-Front' definiëren van een visie middels een visiedocument.* (Info)
- [ ] *Aanpak 2 – Het definiëren van een visie en deze gedurende het project valideren, bijstellen en uitdragen middels een Product Vision Board.* (Info)
- [ ] *Aanpak 3 – De visie van het product definiëren en valideren middels impact mapping.* (Info)

Bent u het eens met de keuze van de expert, en denkt u dus dat deze aanpak het meest geschikt is in deze situatie?

- ○ Ja
- ○ Nee, omdat:

Doel 2: Definiëren en beheren van de requirements (Info)

Om het bovenstaande doel te behalen, heeft de expert de onderstaande procesaanpak gekozen:

- [ ] *Procesaanpak 1 - Het 'Up-Front' definiëren van requirements (Waterval).* (Info)
- [x] *Procesaanpak 2 - Het iteratief en incrementeel definiëren van requirements (RUP).* (Info)
- [ ] *Procesaanpak 3 - Het 'Just In Time' en het 'Just Enough' definiëren van de requirements (Agile).* (Info)

Bent u het eens met de keuze van de expert, en denkt u dus dat deze aanpak het meest geschikt is in deze situatie?

- ○ Ja
- ○ Nee, omdat:

Doel 3: Beheren en specificeren requirements (Info)

Om het bovenstaande doel te behalen, heeft de expert de onderstaande aanpakken gecombineerd:

- [ ] *Aanpak 1 - Het beheren en specificeren van de requirements middels een van uit systeemperspectief geschreven Software Requirements Document.* (Info)
- [x] *Aanpak 2 - Het beheren en specificeren van de requirements middels een 'lichtgewichte' versie van een uit gebruikersperspectief geschreven Software Requirements Document.* (Info)
- [ ] *Aanpak 3 - Het beheren en specificeren van de requirements middels use cases, user stories en een product backlog.* (Info)
- [x] *Aanpak 4 - Het beheren en specificeren van de requirements middels epics/features, user stories en een product backlog.* (Info)
- [ ] *Aanpak 5 – Het beheren en specificeren van de requirements middels epics/features, user stories en een product backlog in combinatie met Specification By Example.* (Info)

Bent u het eens met de keuze van de expert, en denkt u dus dat de combinatie van deze aanpakken het meest geschikt zijn in deze situatie?

- ○ Ja
- ○ Nee, omdat:

Doel 4: Requirements overzichtelijkheid houden (Info)

Om het bovenstaande doel te behalen, heeft de expert geen van de onderstaande additionele aanpakken gekozen:

- [ ] *Additionele aanpak 1 - Requirements overzichtelijk houden middels story mapping.* (Info)
- [ ] *Additionele aanpak 2 – Requirements overzichtelijk houden middels process mapping* (Info)

Bent u het eens met de keuze van de expert?

- ○ Ja
- ○ Nee, omdat:

Radboud University Nijmegen

***Case Study:* Clockster – Het vervangen en uitbreiden van het garagemanagementsysteem**
Clockster is een midden- en kleinbedrijf dat gespecialiseerd is in het leveren van totaaloplossingen aan autobedrijven. Eén van de producten, het garagemanagement systeem, staat al sinds 1995 in de markt en is dringend toe aan vernieuwing, dit omdat het ondersteunen van nieuwe standaarden steeds lastiger wordt en het onderhouden van de oude (VB 6) applicatie erg tijdrovend is. De doelstelling van *Clockster* is om gefaseerd binnen 2 jaar het oude systeem te vervangen en het nieuw systeem in de markt te zetten. Welke functionaliteiten, naast de 5 hoofd-bedrijfsprocessen, het product moet gaan bevatten is nog niet bekend.

Omdat *Clockster* niet over voldoende eigen middelen beschikt, is er gekozen om 4 externe teams in te huren. Omdat *Clockster* bang is dat, door de complexe bedrijfsprocessen en het complexe domein, het product uiteindelijk niet aan de verwachting gaat voldoen, is het gewenst dat er een bepaalde mate van 'formele' documenten wordt opgeleverd. Dit is tevens van belang omdat het project (visiedocument) eerst door de Raad van Bestuur moet worden goedgekeurd.

U bent de leidinggevende van het projectteam en uw team mag het bovenstaande project ten uitvoer brengen. Omdat dit voor Clockster een belangrijk project is, is de hulp ingeroepen van een externe expert die geacht wordt om de meeste geschikte requirements-specificatie-aanpak te kiezen.

In de onderstaande vragen wordt u, per doel, gevraagd of u het eens bent met de gekozen aanpak van de expert. Is het u niet duidelijk wat een doel inhoudt, of bent u benieuwd naar de toepassingseisen of de kwaliteitsanalyse, dan kunt u op info klikken voor meer informatie.

Doel 1: Het definiëren van een visie (Info)

Om het bovenstaande doel te behalen, heeft de expert de onderstaande aanpakken gekozen:

- [x] *Aanpak 1 - Het volledig 'Up'-Front' definiëren van een visie middels een visiedocument.* (Info)
- [x] *Aanpak 2 – Het definiëren van een visie en deze gedurende het project valideren, bijstellen en uitdragen middels een Product Vision Board.* (Info)
- [ ] *Aanpak 3 – De visie van het product definiëren en valideren middels impact mapping.* (Info)

Bent u het eens met de keuze van de expert, en denkt u dus dat de combinatie van deze aanpakken het meest geschikt zijn in deze situatie?

- ○ Ja
- ○ Nee, omdat:

Doel 2: Definiëren en beheren van de requirements (Info)

Om het bovenstaande doel te behalen, heeft de expert de onderstaande procesaanpak gekozen:

- [ ] *Procesaanpak 1 - Het 'Up-Front' definiëren van requirements (Waterval).* (Info)
- [ ] *Procesaanpak 2 - Het iteratief en incrementeel definiëren van requirements (RUP).* (Info)
- [x] *Procesaanpak 3 - Het 'Just In Time' en het 'Just Enough' definiëren van de requirements (Agile).* (Info)

Bent u het eens met de keuze van de expert, en denkt u dus dat deze aanpak het meest geschikt is in deze situatie?

- ○ Ja
- ○ Nee, omdat:

Doel 3: Beheren en specificeren requirements (Info)

Om het bovenstaande doel te behalen, heeft de expert de onderstaande aanpak gekozen:

- [ ] *Aanpak 1 - Het beheren en specificeren van de requirements middels een van uit systeemperspectief geschreven Software Requirements Document.* (Info)
- [ ] *Aanpak 2 - Het beheren en specificeren van de requirements middels een 'lichtgewichte' versie van een uit gebruikersperspectief geschreven Software Requirements Document.* (Info)
- [x] *Aanpak 3 - Het beheren en specificeren van de requirements middels use cases, user stories en een product backlog.* (Info)
- [ ] *Aanpak 4 - Het beheren en specificeren van de requirements middels epics/features, user stories en een product backlog.* (Info)
- [ ] *Aanpak 5 – Het beheren en specificeren van de requirements middels epics/features, user stories en een product backlog in combinatie met Specification By Example.* (Info)

Bent u het eens met de keuze van de expert, en denkt u dus dat deze aanpak het meest geschikt is in deze situatie?

- ○ Ja
- ○ Nee, omdat:

Doel 4: Requirements overzichtelijkheid houden (Info)

Om het bovenstaande doel te behalen, heeft de expert de onderstaande additionele aanpakken gekozen:

- [x] *Additionele aanpak 1 - Requirements overzichtelijk houden middels story mapping.* (Info)
- [x] *Additionele aanpak 2 – Requirements overzichtelijk houden middels process mapping* (Info)

Bent u het eens met de keuze van de expert, en denkt u dus dat deze additionele aanpakken het meest geschikt zijn in deze situatie?

- ○ Ja
- ○ Nee, omdat:

Radboud University Nijmegen

Case study: ING – Het vergoten van het betaalgemak
Eén van de hoofddoelen van de ING voor komende jaren is het vergroten van het betaalgemak van de consument. Op deze manier hoopt ING klanten nog meer aan zich te binden en indirect nieuwe klanten te werven. Eén van de manieren waarmee ING het betaalgemak wilt verhogen is met de introductie van 'contactloos' betalen. "Bij deze vorm van betalen moet u de betaalpas of de mobiele telefoon kortstondig dichtbij de betaalautomaat houden om te betalen". Omdat het nog niet bekend is of dit concept daadwerkelijk gaat aanslaan in de markt, dit omdat concurrenten zoals de Rabobank met vergelijkbare oplossingen bezig zijn (NFC), wordt door de ING het interne projectteam R&D ingezet. Het Research en Development team, bestaande uit een drietal softwareontwikkelaars en een tweetal business analisten, heeft als doelstelling om het product zo snel mogelijk in de markt te zetten middels een pilot, dit om de concurrentie voor te zijn.

U bent de leidinggevende van het R&D team van de ING en uw team mag het bovenstaande project ten uitvoer brengen. Omdat dit voor de ING een prestigieus project is, is de hulp ingeroepen van een externe expert die geacht wordt om de meeste geschikte requirements-specificatie-aanpak te kiezen.

In de onderstaande vragen wordt u, per doel, gevraagd of u het eens bent met de gekozen aanpak van de expert. Is het u niet duidelijk wat een doel inhoudt, of bent u benieuwd naar de toepassingseisen of de kwaliteitsanalyse, dan kunt u op info klikken voor meer informatie.

Doel 1: Het definiëren van een visie (Info)

Om het bovenstaande doel te behalen, heeft de expert de onderstaande aanpak gekozen:

- ☐ *Aanpak 1 - Het volledig 'Up-Front' definiëren van een visie middels een visiedocument.* (Info)
- ☐ *Aanpak 2 – Het definiëren van een visie en deze gedurende het project valideren, bijstellen en uitdragen middels een Product Vision Board.* (Info)
- ☑ *Aanpak 3 – De visie van het product definiëren en valideren middels impact mapping.* (Info)

Bent u het eens met de keuze van de expert, en denkt u dus dat deze aanpak het meest geschikt is in deze situatie?

- ○ Ja
- ○ Nee, omdat:

Doel 2: Definiëren en beheren van de requirements (Info)

Om het bovenstaande doel te behalen, heeft de expert de onderstaande procesaanpak gekozen:

- ☐ *Procesaanpak 1 - Het 'Up-Front' definiëren van requirements (Waterval).* (Info)
- ☐ *Procesaanpak 2 - Het iteratief en incrementeel definiëren van requirements (RUP).* (Info)
- ☑ *Procesaanpak 3 - Het 'Just In Time' en het 'Just Enough' definiëren van de requirements (Agile).* (Info)

Bent u het eens met de keuze van de expert, en denkt u dus dat deze aanpak het meest geschikt is in deze situatie?

- ○ Ja
- ○ Nee, omdat:

Doel 3: Beheren en specificeren requirements (Info)

Om het bovenstaande doel te behalen, heeft de expert de onderstaande aanpak gekozen:

- ☐ *Aanpak 1 - Het beheren en specificeren van de requirements middels een van uit systeemperspectief geschreven Software Requirements Document.* (Info)
- ☐ *Aanpak 2 - Het beheren en specificeren van de requirements middels een 'lichtgewichte' versie van een uit gebruikersperspectief geschreven Software Requirements Document.* (Info)
- ☐ *Aanpak 3 - Het beheren en specificeren van de requirements middels use cases, user stories en een product backlog.* (Info)
- ☑ *Aanpak 4 - Het beheren en specificeren van de requirements middels epics/features, user stories en een product backlog.* (Info)
- ☐ *Aanpak 5 – Het beheren en specificeren van de requirements middels epics/features, user stories en een product backlog in combinatie met Specification By Example.* (Info)

Bent u het eens met de keuze van de expert, en denkt u dus dat deze aanpak het meest geschikt is in deze situatie?

- ○ Ja
- ○ Nee, omdat:

Doel 4: Requirements overzichtelijkheid houden (Info)

Om het bovenstaande doel te behalen, heeft de expert **geen** van de onderstaande additionele aanpakken gekozen:

- ☐ *Additionele aanpak 1 - Requirements overzichtelijk houden middels story mapping.* (Info)
- ☐ *Additionele aanpak 2 – Requirements overzichtelijk houden middels process mapping* (Info)

Bent u het eens met de keuze van de expert?

- ○ Ja
- ○ Nee, omdat:

Radboud University Nijmegen

De onderstaande vragen gaan over uw mening over het gehele model en de toepassing van het model in de praktijk.

Zoals eerder genoemd is het doel van het model tweedelig: Het eerste doel is om ICT-professionals te ondersteunen bij de te maken keuzes op het gebied van requirements-specificatie-aanpakken. Het tweede doel van het model is om als naslagwerk te dienen om meer te weten te komen over de hedendaagse beschikbare requirements-specificatie-aanpakken. Het doel van het model is niet om de werking van de voorgestelde aanpakken exact te beschrijven, dit staat immers beschreven in de literatuur.

Het model bevat de volgende informatie:

- De te behalen doelen in een software-ontwikkelingsproject.
- Per doel de mogelijk te kiezen requirements-specificatie-aanpakken.
- Per aanpak:
  - Beschrijving: Wat is kenmerkend aan deze aanpak?
  - Toepassing eisen: Welke eisen stelt de aanpak aan het team, de stakeholders, het product en de omgeving?
  - Artefacten: In welke artefacten resulteert het gebruik van deze aanpak?
  - Welke artefacten worden er gebruikt door deze aanpak?
  - Kwaliteitsanalyse: Welke invloed heeft de aanpak op de functionele kwaliteit, technische kwaliteit, planning en kostenschatting van het systeem/project?

Het gehele model kunt u hier vinden.

Zou *u* het model als hulpmiddel gebruiken wanneer u in de toekomst een requirements-specificatie-aanpak moet kiezen voor een project?

Zeer onwaarschijnlijk | Onwaarschijnlijk | Geen mening | Waarschijnlijk | Zeer waarschijnlijk

Zou het model, *voor u*, als naslagwerk kunnen dienen om meer te weten te komen over de hedendaagse beschikbare requirements-specificatie-aanpakken?

Zeer onwaarschijnlijk | Onwaarschijnlijk | Geen mening | Waarschijnlijk | Zeer waarschijnlijk

Zou het model, *voor andere ICT professionals*, als naslagwerk kunnen dienen om meer te weten te komen over de hedendaagse beschikbare requirements-specificatie-aanpakken?

Zeer onwaarschijnlijk | Onwaarschijnlijk | Geen mening | Waarschijnlijk | Zeer waarschijnlijk

Zou het model *in opleidingstrajecten* als naslagwerk kunnen dienen om meer te weten te komen over de hedendaagse beschikbare requirements-specificatie-aanpakken?

Zeer onwaarschijnlijk | Onwaarschijnlijk | Geen mening | Waarschijnlijk | Zeer waarscijnlijk

Welk cijfer geeft u de bruikbaarheid van het model in het geheel?

1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10

Heeft u nog op- en/of aanmerkingen over het model?

0% 100%

Radboud University Nijmegen

De onderstaande vragen gaan over u als respondent.

Wat is uw geslacht?

- ○ Man
- ○ Vrouw

Wat is uw leeftijd?

Bij welk bedrijf bent u momenteel werkzaam?

Hoeveel jaren ervaring heeft u op het gebied van Software Ontwikkeling?

- ○ Geen
- ○ <1
- ○ 1-5
- ○ 6-10
- ○ 11-15
- ○ >15

Hoeveel jaren ervaring heeft u op het gebied van Requirements Engineering en/of Projectmanagement?

- ○ Geen
- ○ <1
- ○ 1-5
- ○ 6-10
- ○ 11-15
- ○ >15

Hoeveel jaren ervaring heeft u op het gebied van Requirements Engineering en/of Projectmanagement?

- ○ Geen
- ○ <1
- ○ 1-5
- ○ 6-10
- ○ 11-15
- ○ >15

Wat is uw specialisme binnen het vakgebied van ICT?

- ☐ Projectmanagement
- ☐ Requirements & Analysis
- ☐ Software Development
- ☐ Business Intelligence
- ☐ Testing
- ☐ Anders, namelijk:

Wilt u nog iets kwijt over de enquête of het onderzoek in het geheel?

0% 100%