AFSTUDEERSCRIPTIE INFORMATION SCIENCE

Gecontroleerde natuurlijke taal voor requirements

Een onderzoek naar de vereisten vanuit taal- en gebruikersperspectief

Auteur:	Martijn Speksnijder (s0828653)
Universiteit:	Radboud Universiteit Nijmegen
Studie:	Master Information Science
Afstudeernummer:	179IK
Begeleider:	prof. dr. E. (Erik) Barendsen
Referent:	dr. S.J.B.A. (Stijn) Hoppenbrouwers

Datum:	16 mei 2013
Versie:	1.0

Samenvatting

In veel softwareprojecten worden de requirementsdocumenten opgesteld in een natuurlijke taal. Door het gebruik van natuurlijke taal zijn de requirements voor domeinexperts gemakkelijk te lezen. Helaas zijn de requirements vaak dubbelzinnig doordat er in natuurlijke taal veel ambiguïteit voorkomt.

Het probleem van ambiguïteit kan worden opgelost door formele talen te gebruiken. Die hebben echter weer het nadeel dat slechts een beperkt aantal personen binnen een softwareproject in staat is om de requirements te begrijpen. Zo is het voor de requirements engineer moeilijk om met alle betrokkenen over de requirements te communiceren.

Om de voordelen van natuurlijke en formele talen zo goed mogelijk met elkaar te combineren zijn er zogenaamde *gecontroleerde natuurlijke talen* ontwikkeld. Zo'n CNL (controlled natural language) beperkt de woordenschat, grammatica en semantiek van een natuurlijke taal. Met een gecontroleerde taal kunnen requirements worden opgesteld die niet dubbelzinnig zijn maar wel begrepen worden door al de betrokkenen bij de softwareontwikkeling.

In deze studie is onderzocht welke aspecten belangrijk zijn voor het kiezen van een geschikte CNL in requirements engineering. Ten eerste is nagegaan welke typen ambiguïteit een rol spelen in requirements engineering. Ten tweede is onderzocht welke factoren voor de betrokkenen belangrijk zijn om requirements op te stellen en de requirements te begrijpen. Hieruit zijn selectiecriteria afgeleid, die zijn toegepast op vier bekende CNLs. Hieruit is een taal geselecteerd (PENG) die volgens de onderzoeksresultaten het meest geschikt is.

Voorwoord

Voor u ligt mijn afstudeerscriptie over *'Gecontroleerde natuurlijke taal voor requirements'*. Deze scriptie is het resultaat van een onderzoek naar aspecten die belangrijk zijn voor het kiezen van een geschikte CNL.

Ik heb dit onderzoek uit mogen voeren bij het softwarebedrijf VitalHealth software BV. Dit softwarebedrijf heeft mij de input geleverd die ik nodig had voor mijn onderzoeksactiviteiten. Hierbij valt te denken aan het beschikbaar stellen van software requirements voor het onderzoek. Ook zijn er veel medewerkers geweest die tijd beschikbaar hebben gesteld voor requirements analyse, interviews en enquêtes. Ik wil iedereen hartelijk bedanken die binnen dit bedrijf tijd heeft willen investeren door mij te helpen met input voor het onderzoek.

Ik wil ook Erik Barendsen bedanken voor zijn begeleiding en ondersteuning tijdens het verloop van dit onderzoek.

Martijn Speksnijder
Ede, april 2013

Inhoud

1. Inleiding

1.1 Aanleiding

In de Nederlandse gezondheidszorgsector wordt er minder gebruik gemaakt van ICT mogelijkheden dan in het bedrijfsleven. Uit een onderzoek van Ernst & Young (2011) blijkt dat de meeste Nederlanders willen dat ICT in de gezondheidszorg beter gebruikt wordt. Twee op de drie Nederlanders vinden dat de ICT mogelijkheden in gezondheidszorg nog te weinig worden benut. Volgens 60% van de ondervraagden is een versnelling van ICT implementatie in de zorg noodzakelijk. Een mogelijke oorzaak van het minder benutten is dat veel ICT-project in de gezondheidszorg vertraging oplopen of mislukken. Budgetoverschrijding is hierin ook een groot probleem. Ruim 39% van de ICT-projecten in de gezondheidszorg hebben een budgetoverschrijding. VitalHealth software is vijf jaar geleden opgestart met als doel om ICT innovatie in de zorg te versnellen. Hiermee wil het bedrijf een bijdrage leveren aan de gezondheid van de wereldwijde bevolking. Om dit doel te realiseren is er een softwareplatform ontwikkeld waarop snel softwareapplicaties gebouwd kunnen worden, zodat deze binnen het budget van een zorginstelling liggen. Doordat de softwareapplicaties aan steeds meer eisen moeten voldoen is de snelheid waarmee de applicaties gemaakt worden teruggelopen.

Buiten de gezondheidszorg zijn er ook nog steeds veel softwareprojecten die falen, ondanks de technologische ontwikkelingen van de laatste jaren. Uit het Chaos-rapport (2009) van de Standish Group blijkt dat 21% van de softwareprojecten (er zijn 10.000 projecten onderzocht) volledig zijn mislukt. De mislukte projecten zijn voortijdig stopgezet of geweigerd door de opdrachtgever. Ongeveer 37% van de onderzochte projecten was wel succesvol afgerond: op tijd, binnen het budget en tot tevredenheid van de klant. De overige 42% had problemen: te laat opgeleverd, kosten stijging van het project of ze kwamen niet tegemoet aan de wensen van de eindgebruiker. Een onderzoek van PM Solutions (2011), een Amerikaans project managementadviesbureau, beschrijft de top vijf van oorzaken waardoor softwareprojecten mislukken. De eerste oorzaak die genoemd wordt is: *"Requirements; Onduidelijk, geen overeenstemming, geen prioriteit, tegenstrijdig, dubbelzinnig, onnauwkeurig"*. Uit het onderzoek van PM Solutions blijkt hoe belangrijk het voor het slagen van een softwareproject is om de requirements analyse goed uit te voeren.

Bijna driekwart (72%) van de requirementsdocumenten worden opgesteld in een natuurlijke taal (Luisa, Mariangela, & Pierluigi, 2004). Zinnen die gemaakt worden met een natuurlijke taal kunnen meestal op meer dan één manier worden geïnterpreteerd. Requirements waarin ambiguïteit voorkomt zijn dubbelzinnig en onnauwkeurig. Om ambiguïteit in de requirements te voorkomen kan de requirements engineer gebruik maken een formele taal. Om een requirement in een formele taal te begrijpen heeft de opdrachtgever of applicatieontwikkelaar kennis van de formele taal nodig.

In dit onderzoek wordt onderzocht wat de mogelijkheden zijn van Controlled Natural Language (CNL) voor requirements. Een Controlled Natural Language is een subset van een natuurlijke taal. De grammatica en het aantal te gebruiken woorden is beperkt om ambiguïteit en complexiteit te verminderen of voorkomen (Schwitter, 2002). Het voordeel van een CNL taal is dat deze gebaseerd is op een natuurlijke taal. Door deze eigenschap is het lezen van een zin in een CNL even gemakkelijk als het lezen van een zin in een natuurlijke taal. Een persoon die het Engels beheerst is in staat om zonder veel training zinnen in een CNL te lezen.

Dit onderzoek wil een bijdrage leveren aan het onderzoek naar het gebruik van CNL talen voor requirements. In dit onderzoek ligt de nadruk op welke wensen betrokkenen van de requirements in de praktijk hebben voor een CNL.

1.2 Leeswijzer

Deze scriptie bestaat uit zes hoofdstukken. In hoofdstuk twee worden de probleemstelling en de onderzoeksvragen van het onderzoek beschreven. In hoofdstuk drie is de theorie beschreven die gebruikt is voor het onderzoek. Hoofdstuk drie bestaat uit drie onderdelen. In de eerste paragraaf wordt er achtergrond informatie gegeven over requirements. In deze paragraaf zijn ook vijf kwaliteitscriteria voor requirements beschreven. Op basis van deze criteria is het requirementsdocumenten onderzoek, de interviews en de enquête opgesteld. In de tweede paragraaf is beschreven wat de nadelen zijn van requirements die zijn opgesteld in een natuurlijke taal. In de derde paragraaf zijn de gecontroleerde talen beschreven die in dit onderzoek onderzocht zijn. In hoofdstuk vier is beschreven welke onderzoeksmethoden er gebruikt zijn om antwoorden te vinden op de onderzoeksvragen. In hoofdtuk vijf zijn de resultaten van het onderzoek weergegeven. In hoofdstuk zes is de conclusie van het onderzoek beschreven.

2. Onderzoeksdoel

Binnen de softwareontwikkeling vervullen requirements een belangrijke rol. In de requirements staat beschreven wat de te ontwikkelen softwareapplicatie moet kunnen. De requirements zijn gebaseerd op de wensen en eisen van de belanghebbenden. Een belanghebbende kan de opdrachtgever, toekomstige gebruiker of betrokkene zijn van de softwareapplicatie. Met de requirements wordt er als het ware een brug gelegd tussen de belanghebbenden met hun wensen, en het softwareontwikkelteam met wat zij moeten maken.

2.1 Probleemstelling

Het softwareontwikkelteam moet een softwareapplicatie opleveren die aan de requirements van de business voldoet (de Swart, 2010). Het softwareontwikkelteam kan dit alleen doen wanneer de wensen en eisen van de business goed zijn beschreven in de requirements. De wensen en eisen van de belanghebbenden moeten goed gespecificeerd zijn zodat deze begrepen worden door de applicatieontwikkelaars. Uit het onderzoek van PM Solutions (2011) blijkt dat dit vaak het probleem is. De requirements zijn vaak niet goed, maar dubbelzinnig en onnauwkeurig opgesteld.

In de meeste softwareprojecten worden de requirementsdocumenten opgesteld in een natuurlijke taal. Het gebruik van een natuurlijke taal heeft als voordeel dat elke persoon die betrokken is bij de requirements in staat is om deze te lezen; elk mens beheerst namelijk één of meerdere natuurlijke talen. Het nadeel van een natuurlijke taal is dubbelzinnigheid. Zinnen of zelfs woorden hebben in een natuurlijke taal vaak meer dan één betekenis. De 500 meest gebruikte woorden in het Engels hebben gemiddeld 23 verschillende betekenissen (Kamsties & Peach, 2000). Het gevolg hiervan is dat een requirement die beschreven is in een natuurlijke taal meestal error gevoelig en vaag is. Dit leidt tot requirements die ambigu, incompleet of onnauwkeurig zijn (Kamalrudin, Hosking, & Grundy, 2011). Requirements die ambigu zijn vormen een obstakel voor de ontwikkeling van een softwareapplicatie. Vooral ongewenste ambiguïteit is erg risicovol. Ambiguïteit in de requirements kan namelijk leiden van uiteenlopende verwachtingen van belanghebbenden en applicatieontwikkelaars tot onvoldoende of ongewenste implementatie van de requirements in de applicatie. Een requirement die ambigu is kan door twee verschillende ontwikkelaars op een geheel andere wijze worden uitgevoerd, hoewel de ontwikkelaars denken dat zij requirements precies gevolgd hebben (Berry, Kamsties, & Krieger, 2003). In sommige bedrijven worden reviewsessies gehouden om ambiguïteiten in requirementsdocumenten te identificeren. De gevonden ambiguïteiten worden samengevoegd en terug gestuurd naar de auteur voor correctie. Hierbij is de kwaliteit van het reviewproces afhankelijk van hoe effectief de reviewer is in het vinden van ambiguïteiten en andere requirementsfouten.

Formele talen zijn veel minder ambigu dan natuurlijke talen. Formele talen zijn dan ook voorgesteld om gebruikt te worden voor het opstellen van requirements. Het nadeel van formele talen is dat deze moeilijk te begrijpen zijn voor stakeholders. Formele talen veroorzaken hierdoor een afstand tot het applicatiedomein wat niet het geval is met een natuurlijke taal (Schwitter, 2010). Een manier om het gat tussen natuurlijke en formele taal te dichten is door gebruik te maken van een gecontroleerde natuurlijke taal (Controlled Natural Language (CNL)). Een CNL is een kunstmatige taal die gebruik maakt van een deel van de woordenschat en grammatica van een natuurlijke taal. Een gecontroleerde taal heeft beperkingen op de woordenschat, grammatica en semantiek om ambiguïteit in zinnen te voorkomen.

2.2 Onderzoeksvragen

In dit onderzoek is een onderzoek gedaan naar de eigenschappen waarover een CNL taal moet beschikken om op een gebruiksvriendelijke manier kwalitatieve requirements op te stellen. Met welke eigenschappen wordt de kwaliteit van de requirements verbetert en welke eigenschappen moet de CNL taal hebben om gebruiksvriendelijk te zijn voor al de verschillende groepen die betrokken zijn bij het requirement engineerings proces. Daarnaast is in dit onderzoek uit een voorgeselecteerd aantal CNL talen een taal gekozen die de beste eigenschappen heeft om daarmee in de praktijk de requirements op te stellen. De voorgeselecteerde talen zijn: Attempto Controlled English (ACE), Processable ENGlish (PENG), Common Logic Controlled English (CLCE) en Computer Processable Language (CPL).

De onderzoeksvraag van dit onderzoek is:

> *"Waar moet een gecontroleerde natuurlijke taal aan voldoen om de kwaliteit van software requirements in de praktijk op een gebruiksvriendelijke manier te verbeteren?"*

De deelvragen in dit onderzoek zijn:

1. *Hoe kan ambiguïteit op een betrouwbare manier worden opgespoord in de requirements?*
2. *Welke factoren beïnvloeden in de praktijk de kwaliteit van de requirements?*
3. *Wat zijn de wensen vanuit de praktijk voor een gebruiksvriendelijke gecontroleerde natuurlijke taal?*
4. *Welke van de vier geselecteerde gecontroleerde natuurlijke talen heeft de juiste eigenschappen (kwaliteit en gebruiksvriendelijk) om daarmee requirements te definiëren?*

2.3 Onderzoekscontext

De data voor het onderzoek is verzameld binnen het softwarebedrijf VitalHealth software BV. VitalHealth is gevestigd in de Verenigde Staten, India en Nederland. De vestigingslocatie in Nederland is Ede. VitalHealth is in het najaar van 2005 opgericht door Noaber Foundation in Nederland en Mayo Clinic in de VS. De missie van het bedrijf is om door softwareoplossingen de gezondheid van de mensen wereldwijd te verbeteren. Bij het bedrijf werken meer dan honderd werknemers. VitalHealth heeft een softwareplatform ontwikkeld waarop web-applicaties kunnen worden gebouwd. Een voorbeeld van een web-applicatie is Vital for diabetes. Met deze applicatie kan een zorgverlener een dossier bijhouden van diabetes patiënten. De patiënt heeft ook zelf de mogelijkheid om in een eigen portaal actief deel te nemen aan zijn behandeling. Het softwareplatform van VitalHealth vormt de basis van de web-applicaties. De applicaties worden gebouwd door applicatieontwikkelaars in de door VitalHealth ontwikkelde modelleer-tool. In deze tool maken applicatieontwikkelaars modellen voor de database, bussiness-rules, workflow en visuele frontend van de applicatie. De applicatiemodellen zijn xml-bestanden die door het VitalHealth platform worden getransformeerd naar web-applicaties.

3. Theoretisch kader

"Unintended ambiguity is the Achilles' heel of requirements ..."
(Daniel M. Berry, Erik Kamsties, and Michael M. Krieger)

In dit hoofdstuk is de theorie beschreven die in dit onderzoek gebruikt is. Het hoofdstuk bestaat uit drie onderdelen. Het eerste onderdeel beschrijft wat requirements zijn, en aan welke kwaliteitscriteria requirements moeten voldoen. In het tweede onderdeel staat beschreven hoe natuurlijke taal de kwaliteit van de requirements beïnvloed. In het derde onderdeel zijn de gecontroleerde talen die onderzocht zijn beschreven.

3.1 Software requirements

Requirements vormen de basis voor veel activiteiten die worden uitgevoerd tijdens de softwareontwikkeling. Hierbij valt te denken aan activiteiten zoals ontwerpen, bouwen en testen. Zonder de requirements kunnen veel activiteiten niet worden uitgevoerd. Het is bijvoorbeeld niet mogelijk om te testen of een applicatie daadwerkelijk voldoet aan de wensen van de opdrachtgever. Bij een softwareapplicatie zonder requirements is ook het doel van het systeem niet bekend. Tegelijkertijd is het onmogelijk om te definiëren waaraan de softwareapplicatie moet voldoen om in de behoeften van de gebruikers te voorzien.

3.1.1 Requirements definitie

De term requirement is een veelgehoorde algemene term. In de literatuur wordt er geen gemeenschappelijke definitie gebruikt voor deze term. Wiegers (2009) noemt dit een probleem die vaker voorkomt binnen de software industrie; het ontbreekt vaak aan gemeenschappelijke definities voor termen om activiteiten in het werk te beschrijven. De betekenis van de term requirement is niet altijd duidelijk. Er zijn verschillende definities te vinden die beschrijven wat de term requirement inhoudt. Een definitie die veel gehanteerd en geciteerd wordt is de definitie van de IEEE Standard Glossary of Software Engineering Terminology (1990):

1. *A condition or capability needed by a user to solve a problem or achive a objective.*
2. *A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.*
3. *A documentented representation of a condition or capability as in 1 or 2.*

In het eerste deel van de IEEE-definitie staat wat de user, of belanghebbende, nodig heeft om het probleem in het domein op te lossen of om een bepaald doel te bereiken. In het tweede deel van de definities staat dat het doel of de probleemoplossing bereikt moet worden met het systeem of een component daarvan. Wiegers (2009) combineert de punten 1 en 2 in de volgende definitie:

> *"A statement of a customer need or objective, or of a condition or capability that a product must possess to satisfy such a need or objective"*

Kulak en Guiney beschrijven een requirement op de volgende manier "*iets dat een computerapplicatie moet doen voor zijn gebruikers*". Het is een specifieke functie, eigenschap, kwaliteit of principe dat het systeem moet aanbieden om zijn bestaan te rechtvaardigen. Gebaseerd op de IEEE-definitie heeft De Swart (2010) de volgende definitie opgesteld:

> *"Een requirement is:*
> *a. Een behoefte aan geautomatiseerde ondersteuning: een proces of een verbetering daarin, die een belanghebbende uit de business (deels) met behulp van het systeem wil uitvoeren;*
> *b. Een eis aan een systeem: gedrag (functionaliteit) of kwaliteit die het systeem moet bezitten om in een behoefte te voorzien van een belanghebbende uit de business."*

Deze definitie geeft expliciet aan dat naast een eigenschap van een applicatie een requirement ook betrekking kan hebben op een proces of een verbetering daarvan.

Functionele en niet-functionele requirements
In de requirements staan eigenschappen beschreven waaraan een softwareapplicatie moet voldoen. De eigenschappen kunnen functioneel zijn, oplossen van het probleem in het domein, maar het kunnen ook eigenschappen zijn waaraan het systeem moet voldoen om in het domein te functioneren; zoals: "*het systeem moet voldoen aan standaard x*". In de requirementsdocumenten wordt dit verschil vaak gemaakt door functionele en niet functionele requirements van elkaar te scheiden (Kulak & Guiney, 2004).

De *functionele requirements* beschrijven het gewenste gedrag van de softwareapplicatie: de functies en de features van het systeem. In de functies staat beschreven wat een gebruiker met de softwareapplicatie moet doen en in de features staat beschreven welke eigenschappen de softwareapplicatie moet hebben om dit te bereiken. Een voorbeeld van een functionele requirement is: *"Het systeem moet de beschikbare kamers tonen van het hotel"*.
Een *niet-functionele requirement* is een kwaliteitseis waaraan het systeem moet voldoen. Met de niet functionele eisen worden de verborgen gebieden van de softwareapplicatie beschreven die belangrijk zijn voor de gebruiker, hoewel ze dit niet altijd realiseren. Met niet-functionele requirements wordt niet beschreven wat het systeem moet doen, er wordt geen functionaliteit van de softwareapplicatie mee beschreven zoals de naam al aangeeft. Niet functionele requirements zijn eisen die worden gesteld aan bijvoorbeeld gebruiksvriendelijkheid, schaalbaarheid of performance (de Swart, 2010) (Kulak & Guiney, 2004). In dit onderzoek ligt de focus vooral op de functionele requirements.

3.1.2 Requirements engineering
De doelen die bereikt moeten worden met een nieuwe softwareapplicatie zijn meestal niet vooraf gedocumenteerd. Vaak is de kennis hiervan alleen aanwezig in de hoofden van de opdrachtgevers of andere belanghebbenden. Om een softwareapplicatie te maken die voldoet aan de wensen van de opdrachtgever moeten de wensen en eisen zo correct en volledig mogelijk worden beschreven. Dit wordt gedaan tijdens de requirements engineerings (RE) fase. In de RE fase worden de wensen en eisen, die de opdrachtgever voor de softwareapplicatie heeft, omgezet naar requirements. In dit proces heeft de requirements engineer een belangrijke taak. De persoon in deze rol verzamelt de wensen van opdrachtgever, voert verschillende activiteiten uit die betrekking hebben op het kiezen en uitwerken van de wensen, onderzoekt de correctheid van de wensen en beschrijft en interpreteert deze (Palyagar & Richards, 2005). De wensen en eisen van de opdrachtgever zijn gebaseerd op domeinkennis. Dit kan technische-, functionele - of bedrijfskennis zijn. Om requirements op te stellen moet de requirements engineer zich de domeinkennis eigen maken.

<u>Rollen in het requirements enigneerings proces</u>
Bij de ontwikkeling van een softwareapplicatie zijn personen met verschillende rollen betrokken. Voor vrijwel elke rol zijn de requirements van belang bij het uitvoeren van hun activiteiten. In dit onderzoek onderscheiden wij vier verschillende rollen die belang hebben bij de requirements; dit zijn: stakeholder, requirements engineer, applicatieontwikkelaar en applicatietester.

Stakeholder: Onder deze groep vallen diverse personen. De eerste zijn de opdrachtgevers, hiervoor zijn de requirements van belang omdat door deze personen het softwareontwikkeltraject is gestart. Het doel van de opdrachtgever is om met de softwareapplicatie bepaalde bedrijfsdoelen te halen of problemen op te lossen. Onder de groep stakeholders vallen ook de gebruikers van de nieuwe softwareapplicatie. De requirements zijn belangrijk voor deze groep omdat de nieuwe applicatie in hoge mate beïnvloedt of de gebruiker zijn werk goed en prettig kan uitvoeren. In het vervolg van dit document zullen voor de rollen: opdrachtgever, gebruiker of andere belanghebbende in het domein de term stakeholder gebruikt worden, tenzij de specifieke rol bedoeld wordt.
Requirements engineer: De requirements engineers zijn de personen die de requirements voor de softwareapplicatie opstellen. De activiteiten die zij hierbij uitvoeren zijn, verzamelen van wensen en eisen, opstellen probleemstelling van het domein, kennis van het domein op doen en communiceren met de applicatieontwikkelaar over de requirements.

Applicatieontwikkelaar: De applicatieontwikkelaars bouwen de nieuwe softwareapplicatie of een component daarvan. Voor deze groep zijn de requirements belangrijk omdat hierin beschreven staat waaraan de nieuwe softwareapplicatie moet voldoen. De requirements vormen de basis van de activiteiten van deze groep: het systeemontwerp en de realisatie. Zonder requirements is het voor deze groep bijna niet mogelijk om een applicatie te bouwen die geschikt is voor de business.
Applicatietester: De applicatietester test de softwareapplicatie. De tester test of de requirements in de softwareapplicatie zijn geïmplementeerd. Voor deze groep zijn de requirements belangrijk omdat zij hiermee de applicatie testen. Zonder requirements is het voor deze groep niet mogelijk om te testen of de applicatie aan de wensen van de stakeholders voldoet.

3.1.3 *Requirements opstellen*

De requirements worden gedocumenteerd in een software requirements specification (SRS). Het SRS beschrijft zo volledig mogelijk het gedrag van de softwareapplicatie (Wiegers, 2009). In de praktijk is een SRS meestal een document, maar het kan ook een spreadsheet, requirementstool of een database zijn waarin de requirements zijn opgenomen.

Requirementsfout

De requirements vormen de basis voor de activiteiten die uitgevoerd worden tijdens de softwareontwikkeling. Het is belangrijk dat de requirements engineerings fase goed wordt uitgevoerd; een slordige of incomplete requirements analyse resulteert in een slecht fundament voor de verdere uitvoer van de softwareontwikkeling. Bij het opstellen van de requirements kunnen fouten worden gemaakt. Een fout in de software die als oorzaak één of meerdere requirements van slechte kwaliteit heeft is een requirementsfout. Oorzaken van requirementsfouten zijn: de requirements zijn incompleet, inconsistent of ambigu gedefinieerd of de requirement geeft niet precies de wens van de opdrachtgever weer. (Palyagar & Richards, 2005). Fouten in de requirements kunnen worden gevonden op ieder tijdstip tijdens de requirements engineerings fase, maar worden meestal gevonden wanneer de softwareapplicatie al gebouw of gebruikt wordt. De kosten voor het oplossen van requirementsfouten worden exponentieel groter met de volgende fase in de softwareontwikkeling. Een fout die X kost (waar X de kosten van werk en recources is) om deze op te lossen tijdens de requirements engineerings fase kost ongeveer 10*X in de ontwikkelfase en 200*X om op te lossen na de oplevering van de softwareapplicatie (Berry et al., 2003). De oorzaak van deze kostenstijging is dat met een requirementsfout een sneeuwbaleffect op gang komt die door elke fase van het ontwikkelproces heen rolt en nieuwe fouten oplevert: ontwerpfouten, fouten in de code en fouten in integraties met andere softwareapplicaties (Palyagar & Richards, 2005).

3.1.4 *Kwaliteitscriteria voor requirements*

In de IEEE Standaard 830 (1984) worden zeven kwaliteitscriteria beschreven waaraan een kwalitatief goed software requirements specification (SRS) moet voldoen. Hieronder staan vier van deze kwaliteitscriteria beschreven die in dit onderzoek gebruikt zijn. Deze vier criteria zijn uitgekozen omdat deze betrekking hebben op de kwaliteit van de requirements, en niet alleen op de kwaliteit van het requirementsdocument. Het criterium *modifiable* is bijvoorbeeld niet beschreven omdat dit een eigenschap is waaraan een kwalitatief goed document moet voldoen. Een requirementsdocument voldoet aan dit criterium wanneer er geen redundante requirements voorkomen in het document. Aan kwaliteitscriteria is in dit onderzoek ook het criterium implementatie onafhankelijk toegevoegd. Deze eigenschap staat niet beschreven in de IEEE Standaard. In de literatuur over requirements wordt dit criterium echter wel vaak genoemd. Omdat implementatie onafhankelijk op requirement niveau is te onderzoeken is deze opgenomen in dit onderzoek.

Eenduidig

Een requirement die eenduidig is kan maar op één manier worden uitgelegd. De requirement heeft maar één betekenis in het SRS document. In de IEEE Standaard 830 wordt beschreven dat een SRS document eenduidig is wanneer "*if, and only if, every requirement stated therein has only one interpretation*". Dit vereist dat minimaal elk uniek kenmerk van het systeem is beschreven met unieke termen in het SRS document. Wanneer dit niet mogelijk is, als een kenmerk meer dan één betekenis heeft in het SRS document, dan moet in een woordenlijst worden uitgelegd wat de betekenis van de term is in het SRS document.

In de meeste softwareprojecten zijn de requirementsdocumenten opgesteld in een natuurlijke taal. Het nadeel van een requirements die beschreven zijn in een natuurlijke taal is dat deze meestal error gevoelig en vaag zijn. Door het gebruik van natuurlijke taal in requirementsdocumenten kunnen er requirements voorkomen die ambigu zijn en daardoor op een verkeerde manier geïnterpreteerd worden (Luisa et al., 2004). In paragraaf 3.3 zal er dieper worden ingegaan op eenduidigheid in requirements.

Consistent
Het requirementsdocument is consistent wanneer er geen requirements in het document met elkaar conflicteren. In de IEEE Standaard 830 wordt beschreven dat een SRS document consistent is wanneer "*if and only if no set of individual requirements described in it conflict*". In de standaard worden er drie oorzaken genoemd van consistentieconflicten in een requirementsdocument.

- Twee of meer requirements beschrijven hetzelfde object in de reële wereld, maar voor dit object worden verschillende termen gebruikt.
- Het gespecificeerde object heeft conflicten met hetzelfde object in de reële wereld.
- Er is een logische of functioneel conflict tussen twee requirements.

Traceerbaar
Een requirement moet zowel voorwaarts als achterwaarts traceerbaar zijn. Een requirementsdocument is goed achterwaarts traceerbaar wanneer de herkomst van al de requirements duidelijk is. Bij elke requirement kan worden nagegaan wat de veranderingen zijn in elke fase van de softwareontwikkeling. Een requirementsdocument is goed voorwaarts traceerbaar wanneer bij alle requirements in het document de relaties naar andere requirements duidelijk zijn. Bij een wijziging in een requirement zijn de gevolgen voor de overige requirements dan na te gaan. Om de traceerbaarheid te verbeteren adviseert de IEEE Standaard 830 om requirements unieke namen of referentienummers te geven. Door een unieke naam of nummer kan een requirement worden getraceerd in het document. Hierdoor kan elk document dat refereert naar een specifieke requirement (bijvoorbeeld een ontwerpdocument) een exacte referentie naar de requirement maken.

Verifieerbaar
Een requirement is verifieerbaar wanneer een persoon of machine kan controleren dat de requirement in de softwareapplicatie is geïmplementeerd. In de volgenden voorbeelden staan requirements die niet goed verifieerbaar zijn: *"De softwareapplicatie moet goed werken"* of *"De software applicatie heeft een goed UI"*. Deze requirements kunnen niet goed worden geverifieerd omdat het woord *"goed"* niet nader is omschreven.
Aan het einde van de softwareontwikkeling vindt meestal een test plaats op basis van de requirements. Als er met deze test is vastgesteld dat al de requirements in de softwareapplicatie zijn geïmplementeerd dan kan de softwareapplicatie als af worden beschouwd. Voor deze testen is het belangrijk dat de requirements goed verifieerbaar zijn. Verifieerbare requirement zijn opgesteld in een duidelijke meetbare begrensde vorm.

Implementatie onafhankelijk
Wiegers (2009) vindt dat er bij het opstellen van de requirements geen ontwerp- of implementatiedetails opgenomen mogen worden. De details moeten gescheiden worden van de requirements zodat de requirements engineer zich kan focussen op de doelen waaraan de softwareapplicatie moet voldoen. Ook Kulak en Guiney (2004) vinden dat het cruciaal is om de requirements en het ontwerp te gescheiden:

> *"Anything that relates to how the system should operate, rather than what it needs to accomplish, is design. Design should not be part of requirements."*

Om duidelijk te maken wat ontwerp- en implementatiedetails zijn in requirements het volgende voorbeeld waarin een aantal implementatiedetails zijn verwerkt:

> *"Een knop in de menubalk van waaruit de inhoud van deze view kan worden geëxporteerd naar Excel, met inbouw van een filtermogelijkheid."* [1]

1 De geciteerde requirement is overgenomen uit een requirementsdocument van VitalHealth software.

In dit requirement ligt de focus meer op het *hoe* dan op het *wat*. Woorden zoals "*knop*", "*menubalk*" en "*Excel*" beschrijven hoe de requirement moet worden geïmplementeerd in de applicatie. Wanneer de requirement wordt herschreven op een implementatie onafhankelijke wijze dan resulteert dit in de volgende requirement: "*De inhoud van de view moet geëxporteerd worden naar een spreadsheet, hierbij moet het mogelijk zijn om de data van de view te filteren.*"

Kulak en Guiney (2004) beschrijven een aantal redenen waarom het belangrijk is om geen ontwerp- en implementatiedetails op te nemen in de requirements. Ten eerste vinden zij dat personen activiteiten moeten uitvoeren die passen bij de rol en vaardigheden die zij hebben. Personen die requirements verzamelen kunnen vaardigheden hebben die niet geschikt zijn om ontwerp en implementatie beslissingen te maken. Aan de andere kant hebben applicatieontwikkelaars misschien niet de vaardigheden die nodig zijn om requirements te verzamelen. Het uitvoeren van activiteiten zoals gebruikersinterviews, gesprekken met opdrachtgevers of het documenteren van de requirements. Ten tweede vinden Kulak en Guiney dat het opstellen van de requirements het begrijpen en documenteren van het probleem is dat speelt in het domein. Ontwerpen is de activiteit om dit op te lossen. De oplossing van het probleem moet komen nadat het probleem is geïdentificeerd, begrepen, gedocumenteerd en overeengekomen met de stakeholder. Samengevat noemen zij dit: "*Not knowing the problem that your design is solving is dangerous business*". De derde reden die Kulak en Guiney geven is gerelateerd aan het ontwikkelteam die de softwareapplicatie maakt. Wanneer een team een systeem ontwerpt zonder gedocumenteerde requirements dan is het goed mogelijk dat de leden van het team met verschillende doelen in gedachten aan de applicatie werken. Omdat zij niet een gemeenschappelijk document hebben van waaruit zij beginnen met de ontwikkeling van de applicatie. Dit kan problemen opleveren tijdens de ontwikkeling zoals Kulak en Guiney schrijven:

> *"These designs almost certainly will be incompatible and overlapping, causing integration problems, skipped requirements, scope creep, schedule issues and unhappy users".*

Easterbrook (2004) vindt dat het de moeite waard is om "*description of a problem*" te scheiden van de "*descripton of a solution*". Het scheiden van het probleem van de potentiële oplossing en het schrijven van een probleemstelling is nuttig. Bij het maken van een probleemstelling moet de requirements engineer het domein bestuderen, vragen stellen over de activiteiten die de applicatie moet supporten en een scope definiëren voor de nieuwe applicatie. Hierdoor moet de requirements engineer goed begrijpen wat de oorzaak van het probleem is voordat er nagedacht wordt over hoe het opgelost moet worden. Wanneer er direct wordt begonnen aan het ontwerpen van de applicatie kunnen dingen gemist worden die bij een grondige probleemanalyse wel naar boven gekomen waren. Als extra voordeel van het scheiden van probleem en oplossing noemt Easterbrook ook nog dat het makkelijker is om het systeem te testen; een oplossing is alleen correct wanneer het probleem in het domein daarmee wordt opgelost.

3.1.5 Oorzaken interpretatieproblemen requirements

Bij een requirements die op meerdere manieren uitgelegd kunnen worden ontstaan er verschillen tussen de verwachtingen van de stakeholders en de interpretatie van de ontwikkelaars. In dit onderzoek ligt de nadruk op dubbelzinnigheid als oorzaak voor interpretatieproblemen bij requirements. Er zijn echter ook andere oorzaken waardoor een ontwikkelaar een requirement verkeerd interpreteert. In deze paragraaf zullen twee oorzaken worden beschreven. Dit zijn: gebrek aan domeinkennis en het gebruik maken van requirementsnotaties die door gebruikers van de requirements niet begrepen worden.

<u>Requirements notatie</u>

Voor het opstellen van requirements zijn er diverse technieken beschikbaar: user stories, use cases, uml-diagrammen en diverse andere technieken. Voor elke techniek is er een eigen notatie beschikbaar. De rollen die betrokken zijn bij de requirements engineering hebben hun eigen voorkeur voor een notatietechniek. Het communiceren over een softwareapplicatie in termen van gedrag en functionaliteit zal de voorkeur hebben van de stakeholders. De stakholders spreken het liefst over concrete situaties die met de nieuwe applicatie moeten worden opgelost. Personen in deze rol hebben waarschijnlijk een voorkeur voor het opstellen van requirements met de use case techniek (Al-Rawas & Easterbrook, 1996). Met deze notatietechniek worden de requirements beschreven in een natuurlijke

taal. De applicatieontwikkelaar daarentegen heeft een voorkeur voor requirements die abstract en formeel zijn gedefinieerd. Hierbij loopt men algauw tegen een probleem aan; de opdrachtgevers hebben meestal geen kennis van formele talen en willen deze om verschillende redenen ook niet leren. In veel gevallen moet de requirements engineer bij het opstellen van de requirements een natuurlijke taal gebruiken zodat hij met de opdrachtgever kan communiceren.
De requirements engineer moet de requirements verwerken tot een functioneel ontwerp. Hierna is het de taak van de ontwikkelaar om de requirements die in een natuurlijke taal beschreven zijn om te zetten naar een formele taal waarmee de applicatie gemaakt wordt. Al-Rawas en Easterbrook (1996) hebben veldonderzoek gedaan naar de communicatieproblemen tussen de verschillende groepen die betrokken zijn bij het RE proces. Een situatie die zij vaak tegenkwamen bij grote softwareprojecten noemde zij het "over-de-muur-gooi" principe. De requirements engineer engineer stelt hierbij samen met de opdrachtgever de requirements op voor de applicatie. Omdat de ontwikkelaar nog niet betrokken is bij het RE proces kiezen de requirements engineer en de opdrachtgever samen een notatie uit waarmee ze de requirements opstellen. Wanneer de requirements zijn gedefinieerd wordt het opgestelde document zonder commentaar over de muur naar de softwareontwikkelaar gegooid. Het probleem hierbij is dat een softwareontwikkelaar zonder hulp de requirements moet begrijpen, zonder dat de ontwikkelaar kennis heeft van het domein waarin de requirements zijn opgesteld.

Gebrek aan domeinkennis
De applicatieontwikkelaar moet een applicatie maken die voldoet aan de wensen en eisen van de stakeholder. Om dit te doen heeft de applicatieontwikkelaar ondubbelzinnige en nauwkeurige requirements nodig. Zoals beschreven is het de taak van de requirements engineer om deze op te stellen. De requirements engineer moet op een correcte en volledige manier beschrijven wat de stakeholder wil. De requirements engineer en de applicatieontwikkelaar hebben meestal geen kennis van het betrokken vakgebied; zij hebben niet genoeg domeinkennis. De requirements engineer moet de taal van het vakgebied leren, bekend worden met de terminologie, concepten en procedures zodat hij de wensen en eisen van de stakeholder kan interpreteren. De applicatieontwikkelaar heeft domeinkennis nodig om requirements op een juiste manier te interpreteren. Tijdens de RE fase moet de requirements engineer voldoende domeinkennis op doen en communiceren aan de applicatieontwikkelaar zodat deze in staat is om de requirement goed te begrijpen.
De requirements engineerings fase wordt gekenmerkt door de intensiteit en het belang van communicatie tussen de verschillende personen die betrokken zijn bij de ontwikkeling van de nieuwe softwareapplicatie. Curtis (1988) heeft veldonderzoek gedaan naar het falen van softwareprojecten. Hiervoor heeft hij personeel van zeventien grote softwareprojecten geïnterviewd. Eén van de conclusies uit dit onderzoek is dat communicatieproblemen een grote factor zijn voor het vertragen of falen van softwareprojecten. AL-Rawas en Easterbrook (1996) voegen hieraan toe dat dit vooral het geval is bij "socio-technical" softwareapplicaties. Dit zijn softwareapplicaties die gebouwd worden in complexe organisatorische omgevingen. De organisatorische domeinen waarin dergelijke software wordt ingevoerd zijn vaak ingewikkeld en daarom moeilijk om volledig te begrijpen.

3.2 Natuurlijke taal en requirements

Het requirementsdocument moet nauwkeurig, consistent en compleet zijn. In de meeste softwareprojecten wordt het requirementsdocument opgesteld in een natuurlijke taal. Een natuurlijke taal is een taal die door een aantal mensen als hun moedertaal wordt beschouwd. Natuurlijke talen zijn talen zoals het Engels of het Nederlands. Natuurlijke talen zijn de meest krachtige kennisrepresentatie talen die er bestaan (Schwitter, 2004). Natuurlijke talen worden geleerd in de vroege kinderjaren en zijn zo geoptimaliseerd dat ze alle aspecten van de menselijke communicatie kunnen ondersteunen, gebruik makend van één notatie. Natuurlijke talen dienen als hun eigen metataal. Hierdoor hebben natuurlijke talen een grotere flexibiliteit en expressieve kracht dan enige formele taal.

De universiteit van Trento in Italië heeft een online survey[2] gehouden om onderzoek te doen naar het percentage van de Software Requirements Specification (SRS) documenten die worden beschreven in een natuurlijke taal (Luisa et al., 2004). De resultaten van dit onderzoek zijn:

- *71,8% van de documenten worden opgesteld in een algemene natuurlijke taal.*
- *15,9% van de documenten worden opgesteld in een gestructureerde natuurlijke taal.*
- *5,3% van deze documenten worden opgesteld in een geformaliseerde taal.*

Uit de uitkomst van dit onderzoek blijkt dat de meeste requirementsdocumenten worden opgesteld in een natuurlijke taal. Mich en Garigliano (2000) geven hier een aantal redenen voor. De eerste reden is dat de communicatie tijdens het softwareontwikkelingstraject verloopt via een natuurlijke taal. Bij het verzamelen en analyseren van de requirements is het essentieel om het probleem van het domein te begrijpen voordat de requirements worden opgesteld. Hiervoor moet de domeinanalist communiceren met de stakeholders. De communicatie met deze personen verloopt doorgaans in een natuurlijke taal. Ook Yang (2010) geeft aan dat communicatie met stakeholders een belangrijk voordeel is om requirements in een natuurlijke taal op te stellen. Het gebruik van een natuurlijke taal helpt requirements engineers om met stakeholders en applicatieontwikkelaars te communiceren gedurende het hele softwareontwikkelingstraject. De tweede reden is dat input documenten voor de requirements vaak in een natuurlijke taal zijn gemaakt. Een onderzoek heeft aangetoond dat de meerderheid (78%) van de documenten die gebruikt worden als input voor de requirements opgesteld zijn in een natuurlijke taal. Als derde reden wordt output genoemd. Een requirementsdocument die opgesteld is in een natuurlijke taal kan gebruikt worden voor diverse andere doelen. Voorbeelden hiervan zijn: het kan worden gebruikt als input document voor software architecten, testers en voor personen die de handleiding schrijven of het kan ook gebruikt worden als contractdocument tussen de klant en leverancier of als een informatiebron voor de projectmanager (Fabbrini, Fusani, Gnesi, & Lami, 2001).

3.2.1 *Dubbelzinnige requirements*

In hoofdstuk twee is het resultaat van een onderzoek van PM Solutions (2011) beschreven. Een oorzaak die genoemd wordt voor het falen van softwareprojecten is dubbelzinnige requirements. Dubbelzinnige requirements laten ruimte voor verschillende en mogelijk ongewenste interpretatie. Kamsties (2005) geeft drie oorzaken waardoor requirements dubbelzinnig worden. De eerste oorzaak is incomplete requirements. Bij requirements die incompleet zijn is de betekenis van de requirements onduidelijk. Hoe completer de requirement is, hoe minder ambigu deze wordt. De tweede oorzaak is het gebrek aan overeenstemming; conflicterende individuele meningen kunnen leiden tot een verschillende interpretatie van de requirements. De derde oorzaak is de vorm van representatie van de requirement. Informele requirements zijn onvermijdelijk ambigu terwijl formele requirements aanzienlijk minder ambigu zijn. Deze dubbelzinnigheid wordt veroorzaakt door de zwakte in een natuurlijke taal. Een natuurlijke taal is van zichzelf ambigu.

Het gebruik van natuurlijke taal in requirementsdocumenten heeft veel nadelen. Een requirement beschreven in een natuurlijke taal is meestal error gevoelig en vaag. Ambiguïteit in requirements kan potentieel gevaarlijk zijn omdat dit leidt tot slechte requirements. Uit een onderzoek van Hui Yang blijkt dat ambiguïteit veel voorkomt in requirements die opgesteld zijn in een natuurlijke taal. In het onderzoek zijn 11 requirementsdocumenten onderzocht. In deze documenten zijn in 3404 zinnen linguïstische ambiguïteiten gevonden; dat is 12.68% van al de zinnen (het waren er 26829) (Yang et al., 2010). In een natuurlijke taal hebben de meeste woorden verscheidene betekenissen. Door de vele betekenissen kunnen zinnen op meer dan één manier worden geïnterpreteerd (Kamsties & Peach, 2000).

3.2.2 *Ambiguïteit in requirements*

Kamsties (2000) vindt dat requirements die ambigu zijn een probleem vormen tijdens de softwareontwikkeling. De reden die hij hiervoor geeft is dat stakeholders zich er meestal niet van bewust zijn wanneer er een ambiguïteit voorkomt in de requirements. Een ontwikkelaar die een requirement leest kan deze onbewust op een andere manier interpreteren dan de stakeholder. De

2 Deze survey is deel van een online marktonderzoek programma van de Computer Sciences Department van de Trento University

consequentie hiervan kan zijn dat de ontwikkelaar software ontwikkeld die niet naar de wens van de stakeholder is; terwijl hij denkt dat hij de requirements gevolgd heeft. Ook componenten die ontwikkeld worden kunnen falen in de interactie met andere componenten omdat dezelfde requirement werd toegewezen aan verschillende componenten, en de verschillende ontwikkelaars deze op verschillende manieren hebben geïnterpreteerd.
Ambiguïteit kan ook gezien worden als een feature van een natuurlijke taal. Door het toelaten van ambiguïteit is een natuurlijke taal extra expressief. De uitdrukkingsmogelijkheden van een natuurlijke taal zijn door ambiguïteit extra groot. Door het bewust toepassen van ambiguïteit in requirements kan een requirements engineer extra expressieve requirements maken. Een softwareontwikkelaar kan vaak niet afleiden uit de requirement of een ambiguïteit bewust of onbewust is toegepast, hierdoor is het bewust toepassen van ambiguïteit in requirements niet zonder risico.

Voor ambiguïteit in requirements zijn er verschillende definities beschreven in de software engineerings literatuur. Zoals al beschreven geeft de IEEE Standaard 830 (1984) bij het kwaliteitscriterium eenduidigheid de volgende definitie voor ambiguïteit:

> *"An SRS is unambigious if, and only if, every requirement stated therein has only one interpretation."*

De definitie houdt in dat een SRS alleen niet ambigu is wanneer al de requirements in de SRS ook niet ambigu zijn. Berry en Kamsties (2003) merken bij deze definitie op dat er geen requirements zijn die niet ambigu zijn, er is namelijk altijd iemand die de requirement op een andere manier interpreteert dan iemand anders. Er zijn echter genoeg bruikbare requirements die door de meerderheid van de personen op dezelfde manier worden geïnterpreteerd: de meerderheid van de stakeholders, gebruikers en applicatieontwikkelaars. Deze requirements kunnen dan ook goed in de softwareapplicatie worden geïmplementeerd, en overeenkomstig wat de meeste personen verwachten. Mullery (1996) geeft aan dat een zekere mate van ambiguïteit toegestaan mag worden in software requirements. Het maken van een SRS waarin geen ambiguïteiten voorkomen kost veel tijd. De tijd die dit kost moet wel in overeenkomst zijn met de totale tijd die besteed wordt aan de ontwikkeling van de softwareapplicatie.

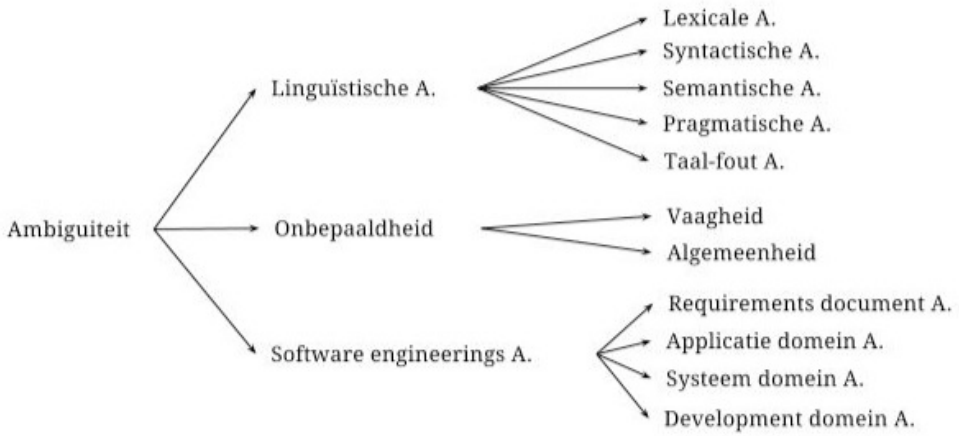
Kamsties, Berry en Paech (2001) definiëren ambiguïteit op de volgende manier:

> *"We define a requirement as ambiguous if it has multiple interpretations despite the reader's knowledge of the RE context."*

Door deze definitie baseren zij het ambigu zijn van een requirement niet op de kennis die een lezer van het domein heeft. Het maakt ook niet uit of een auteur de ambiguïteit bewust of onbewust heeft geïntroduceerd. In dit onderzoek is gebruik gemaakt van deze definitie. Er is voor de definitie gekozen omdat bij het controleren van een requirement op ambiguïteit het bijna onmogelijk is om te bepalen of een ambiguïteit bewust of onbewust is geïntroduceerd en of de domeinkennis van de lezer toereikend is om de requirement juist te interpreteren.

3.2.3 Typen ambiguïteit

Ambiguïteit kan op verschillende wijze voorkomen in requirements. Berry (2008) onderscheid drie groepen van ambiguïteiten. De eerste groep is de linguïstische ambiguïteit, in deze groep staan de traditionele typen ambiguïteiten beschreven. De tweede groep in onbepaaldheid. Dit komt voor wanneer requirement te vaag of algemeen zijn. Naar deze groep is in dit onderzoek geen onderzoek gedaan. De derde groep zijn de software engineerings ambiguïteiten. Deze ambiguïteiten kunnen voor komen bij zinnen die syntactische en semantische juist zijn gedefinieerd, maar die in het requirements engineerings domein dubbelzinnig zijn. De typen ambiguïteiten zullen in de volgende paragrafen verder worden uitgewerkt. In figuur 4.1 is een overzicht gemaakt van de verschillende ambiguïteiten.



Figuur 3.1. Verschillende typen ambiguïteiten. Aangepast en overgenomen van (Berry, 2008)

Linguïstische ambiguïteit

De traditionele typen ambiguïteiten zijn: lexicale, syntactische, semantische, pragmatische en taalfout ambiguïteit. In het hoofdstuk *"Ambiguity in Requirements Specification"* in het boek *"Perspective of software requirements"* hebben Berry en Kamsties (2003) een overzicht gegeven welke typen ambiguïteiten kunnen optreden. In deze paragraaf is gebruik gemaakt van de indeling en theorie die zij voor ambiguïteitstypen hanteren.

Lexicale ambiguïteit

Lexicale ambiguïteit treedt op wanneer een woord meer dan één betekenis heeft. Lexicale ambiguïteit kan weer worden onderverdeeld in drie subtypen: homonymie, polysemie en systematische polysemie.

Homonymie treedt op wanneer een woord dezelfde schrijfwijze heeft als een ander woord maar niet dezelfde betekenis en etymologie. Dit houdt in dat de betekenis van een woord in de loop van de tijd is veranderd. Een voorbeeld in het Nederlands hiervan is het woord stout. Vroeger was de betekenis van dit woord moedig, tegenwoordig is dit ondeugend.

Polysemie treedt op bij een woord met dezelfde schrijfwijze maar met een verschillende betekenis. Een voorbeeld hiervan is het woord groen. Dit woord kan een kleur betekenen maar ook in sommige contexten jeugdig en onervaren.

Systematische polysemie treedt op wanneer de reden voor de polysemie een verwarring is tussen klasse van wat wordt bedoeld met een woord. Deze vorm van ambiguïteit kan op de volgende manieren voorkomen: De betekenis van een woord is verwarrend tussen de klassen en instantie. Een voorbeeld hiervan is de zin: *"ik vind deze jas leuk"*, het woord jas kan verwijzen naar een specifieke individuele jas of naar het type jas die wordt leuk gevonden. Een andere vorm is dat de betekenis van een woord verwarrend is tussen proces en product. Een voorbeeld hiervan zijn de woorden "building" en "writing". Deze woorden kunnen een zelfstandig naamwoord of een werkwoord zijn, de woorden kunnen verwijzen naar een proces of een product. Bij een systematische polysemie kan de betekenis van een woord verwarrend zijn tussen het gedrag en de dispositie. De zin *"dit is een snelle auto"* kan verwijzen naar het gedrag of naar de klasse van de auto bijvoorbeeld een Ferrari.

Syntactische ambiguïteit

Syntactische ambiguïteit, ook wel structurele ambiguïteit genoemd, treedt op wanneer een gegeven volgorde van woorden meer dan één grammaticale structuur kan hebben, de grammaticale structuur met elk een verschillende betekenis. Syntactische ambiguïteit wordt onderverdeeld in analytische, attachment, coördinatie of elliptische ambiguïteit.

Analytische ambiguïteit komt voor wanneer een rol in een zin meerdere betekenissen kan hebben. Een voorbeeld hiervan is de zin: *"De Engelse geschiedenis leraar"*. De betekenissen van deze zin zijn: *"De geschiedenis leraar uit Engeland"* of *"De geschiedenis leraar die gespecialiseerd is in Engeland"*.

Attachment ambiguïteit treedt op wanneer een bepaalde syntactische bestanddeel van een zin, zoals een voorzetsel, kan worden bevestigd aan twee delen van een zin. Een voorbeeld hiervan is: *"De politie schoot op de relschoppers met geweren"*. Het deel met geweren kan bij het zelfstandig naamwoord relschoppers horen of bij het werkwoord schoot. De betekenissen van de zin zijn: *"De politie schoot op*

relschoppers die geweren hebben" of "*De politie schoot met geweren op relschoppers".*
Coördinatie ambiguïteit komt voor wanneer er meer dan één conjunctie, "*en*" of " *of*", wordt gebruikt in een zin of wanneer een conjunctie wordt gebruikt met een beperkend bijwoord. Voorbeeld van de eerste soort is: *"Ik zag Peter en Paul en Mary zag mij"*. Deze zin kan zijn: "*Ik zag (Peter en Paul) en Mary zag mij* of *"Ik zag Peter en (Paul en Mary) zagen mij"*. Een voorbeeld van de tweede soort is *"Jonge man en vrouw"*. Deze zin kan de betekenis hebben van *"Jonge man en jonge vrouw"* of *"jonge man en een vrouw"*.
Elliptische ambiguïteit is een gat in een zin die wordt veroorzaakt door weglating van een lexicaal of syntactisch noodzakelijk bestanddeel. Een voorbeeld hiervan in een zin is: *"Peter kent een rijkere man dan Paul"*. Deze zin heeft twee betekenissen: "*Peter kent een man die rijker is dan Paul*" of *"Peter kent een man die rijker is dan al de mannen die Paul kent"*.

Semantische ambiguïteit
Semantische ambiguïteit komt voor wanneer een zin meer dan één manier heeft om gelezen te worden in de context maar de zin geen lexicale of structurele ambiguïteit bevat. Semantische dubbelzinnigheid kan worden veroorzaakt door: (1) de coördinatie ambiguïteit, (2) referentiële ambiguïteit, en (3) scope dubbelzinnigheid. Coördinatie ambiguïteit is in de vorige paragraaf beschreven. Referentiële ambiguïteit kan voorkomen binnen een zin of tussen een zin en zijn discours context; dit wordt beschreven in het volgende item.
Scope ambiguïteit doet zich voor bij woorden zoals: elke, elk alle, sommige en bij een negatie zoals niet. Scope ambiguïteit treedt op wanneer bij deze woorden het niet duidelijk is tot waar de scope rijkt. Als voorbeeld de volgende zin: *"Alle applicatieontwikkelaars hebben een voorkeur voor de programmeertaal"*. De kwantoren *alle* en *een* hebben op twee manieren met elkaar een verbinding. Wanneer de scope van *een* de scope van *alle* includeert, dan betekent deze zin dat al de applicatieontwikkelaars dezelfde programmeertaal als hun voorkeur hebben. Wanneer de scope van *alle* de scope van *een* includeert dan betekent deze zin dat elke applicatieontwikkelaar een voorkeur heeft, misschien een verschillende, voor een programmeertaal.

Pragmatische ambiguïteit
Pragmatische ambiguïteit treedt op wanneer een zin verschillende betekenissen heeft in de context waarin de zin geuit wordt. Pragmatische ambiguïteit kan op verschillende manieren voorkomen. Een subtype van pragmatische ambiguïteit is de referentiële ambiguïteit.
Referentiële ambiguïteit komt voor wanneer een anafoor of verwijzing kan refereren naar meer dan één element. Een anafoor is een element van de zin die verwijst naar een element die voorkomt in de zin of in een andere zin. Het element waarnaar de anafoor refereert wordt de antecedent genoemd. De antecedent moet voorkomen in een eerder of latere zin in de tekst. Een voorbeeld van een referentiële ambiguïteit is: *"De server zal de software installeren voordat deze weg is"*. De antecedent van de anafoor deze kan server en software zijn.

Taalfout ambiguïteit
Een taalfout ambiguïteit treedt op wanneer een grammaticale, interpunctie, woord keuze of een andere fout in het gebruik van de taal leidt tot een tekst die op diverse manieren geïnterpreteerd kan worden. Berry (2008) geeft drie soorten van taalfout ambiguïteiten. De eerste die hij vaak tegenkomt is het op een verkeerde manier gebruiken van de woorden *only* en *also*, of in het Nederlands *alleen* en *ook*. Berry geeft hierbij de volgende requirement als voorbeeld: *"The spam filter only marks the email it considers to be spam"*. In deze requirement is het niet duidelijk of de auteur bedoelt dat de spamfilter alleen mail markeert wat spam is of dat alleen de spam filter alleen de mail markeert die spam is. Een ander voorbeeld die Berry (2003) geeft is de volgende zin "*I only smoke marlboro*". In deze voorbeeldzin staat het woord *only* waarschijnlijk op de verkeerde plaats, wanneer de auteur bedoelt "*I smoke only marlboro*" en zo zullen de meeste personen deze zin interpreteren. Wanneer het de intentie is van de auteur om aan te geven dat hij alleen marlboro rookt en deze bijvoorbeeld niet eet dan klopt deze zin wel.
Het tweede voorbeeld is het door elkaar gebruiken van meervoud en enkelvoud in een zin. Een voorbeeld van een requirement waarbij dit gebeurt is de volgende: *"De gebruikers van het systeem zijn de arts en de patient"*. De betekenis van deze requirement is dat het systeem bestaat uit niet meer dan twee gebruikers: de arts en de patiënt. De intentie van de auteur van deze requirement zal waarschijnlijk zijn geweest om te beschrijven dat de gebruikers van het systeem de artsen en de

patiënten zijn.
Als derde veel voorkomende fout noemt Berry nog het gebruik van een voornaamwoord waarbij de referent niet duidelijk is, als voorbeeld de zin "*This prevents security breaches*". Het is niet duidelijk waarnaar het woord *this* verwijst.

Software engineering ambiguïteit
Uit een onderzoek van Kamsties (2005) blijkt dat in requirementsdocumenten de meest voorkomende vorm van ambiguïteit software engineerings ambiguïteit is. Linguïstisch ambiguïteit blijkt veel minder aanwezig te zijn in de requirementsdocumenten. De requirementsdocumenten die Kamsties onderzocht heeft bevatten 4 linguïstisch en 34 software engineerings ambiguïteiten. Kamsties beschrijft drie domeinen waarbinnen in het software engineerings proces ambiguïteit voorkomt (geciteerd en vertaald uit het Engels uit *"Understanding Ambiguity in Requirements Engineering"* (2005)):

- Een *requirementsdocument ambiguïteit* treedt op wanneer een requirement diverse interpretaties kan hebben binnen het document waarin de requirement beschreven is. Een requirement is zelden op zichzelf staand. Meestal heeft een requirement impliciete of expliciete verwijzingen naar andere requirements. Dit wil zeggen dat de lezer weet hoe een requirement moet worden geïnterpreteerd omdat de requirement een verband heeft met een andere requirement in het document. Document ambiguïteit kan het gevolg zijn van verkeerd gebruik van een zelfstandig voornaamwoord.
- Een *applicatie domein ambiguïteit* treedt op wanneer een requirement meerdere interpretaties kan hebben binnen het domein waarin de applicatie ontwikkeld wordt.
- Een *systeem domein ambiguïteit* treedt op wanneer een requirement meerdere interpretaties heeft binnen de kennis die een lezer heeft over het systeem domein.

In dit onderzoek ligt de nadruk op linguïstische ambiguïteit. Software engineerings ambiguïteit is lastig te detecteren zonder domeinkennis.

3.2.4 *Ambiguïteiten vinden*

In sommige bedrijven worden reviewsessies gehouden om ambiguïteiten in requirementsdocumenten te identificeren. De gevonden ambiguïteiten worden verzameld en teruggestuurd naar de auteur voor correctie. De kwaliteit van het reviewproces is afhankelijk van hoe effectief de reviewer ambiguïteiten en andere requirementsfouten kan vinden. Voor een menselijke reviewer, of zelfs een groep van hen, is het moeilijk om al de ambiguïteiten in een tekst te identificeren. Een menselijke reviewer leest snel over een ambiguïteit in een zin heen. Dit komt omdat een menselijk reviewer de eerste interpretatie die hem te binnen schiet aanneemt als de juiste. De reviewer is zich meestal niet bewust van andere mogelijke betekenissen van de zin (Popescu, Rugaber, Medvidovic, & Berry, 2008).

Kamsties, Berry en Paech (2001) hebben onderzoek gedaan naar technieken om ambiguïteiten in requirements te detecteren. De techniek die zij gebruiken is een checklist waarmee per requirement wordt gecontroleerd of deze ambigu is. Dit is de volgende checklist:

Item	Description
Lexical Ambiguity, Polysemy	Does a word in a requirement have serveral possibly related meanings? Be aware that lexical ambiguity arises in particular from the actual of a word in an RE context
Systematic Polysemy	A systematic polysemy applies to a class of words: (1) The *object-class ambiguity* arises when a word in a requirement can refer either to a class of objects or to just a particular object of the same class. (2) The process–product ambiguity arises when a word can refer either to a process or to a product of the process. (3) The volatile–persistent ambiguity arises when a word refers to either a volatile or a persistent property of an object.
Referential Ambiguity	Can a phrase in a requirement refer to more than one object in other requirements? Check pronouns (it), definite noun phrases (the road), and ellipses (... if not ...)
Discourse Ambiguity	Does a requirement have several interpretations in relation to other requirements? This ambiguity arises when (1) words such as first, before, between, after, and last are used and can refer to several elements and when (2) adjectives, verbs, or noun phrases refer to more than one condition described before.

Domain Ambiguity	Is the requirement ambiguous with respect to what is known about the application, system, or development domain?

Tabel 3.1 Ambiguïteiten checklist overgenomen van (Kamsties et al., 2001)

In de checklist zijn vier linguïstische ambiguïteitstypen opgenomen. Het laatste item in de lijst is een software engineerings ambiguïteit. In de checklist staan niet al de linguïstische ambiguïteitstypen die beschreven zijn in het voorgaande hoofdstuk. Kamsties, Berry en Paech hebben de typen ambiguïteit in de checklist opgenomen waarvan zij gezien hebben dat deze vaak voorkomen in requirementsdocumenten.

3.3 Controlled Natural Language

Formele talen hebben een goed gedefinieerde syntax, ondubbelzinnige semantiek en ondersteunen automatische redenering. Door deze eigenschappen zijn formele talen al vaak voorgesteld als oplossing voor de ambiguïteitsproblemen in requirements. Het grote nadeel van formele talen is dat ze vaak moeilijk te begrijpen zijn voor domeinspecialisten en stakeholders. Een manier om de kloof tussen natuurlijke en formele taal te overbruggen is om gebruik te maken van een gecontroleerde natuurlijke taal, Controlled Natural Language (CNL).

Een CNL is gebaseerd op een natuurlijke taal, meestal op het Engels, maar bij een CNL zijn er beperkingen op de grammatica, woordenschat en semantiek. Het doel van een CNL is om de ambiguïteit en complexiteit van natuurlijke talen te verminderen. Rolf Schwitter van het *"Center for Language Technology"* van de Macquarie University in Australië heeft veel onderzoek gedaan naar CNLs. De definitie die hij geeft is: *"Controlled Natural Language is een subset van een natuurlijke taal. De grammatica en het aantal te gebruiken woorden is beperkt om ambiguïteit en complexiteit te verminderen of te voorkomen"* (Schwitter, 2002). Kort samengevat is een CNL een kunstmatige taal die gebruik maakt van een deel van de woorden en grammatica van een natuurlijke taal. Omdat een CNL afgeleid is van een natuurlijke taal is het voor personen die de natuurlijke taal beheersen makkelijker om deze te leren, gebruiken, lezen en schrijven in vergelijking met andere technische talen zoals een programmeertaal of een formele taal.

CNLs worden onderverdeeld in twee typen: de talen die de leesbaarheid voor menselijke lezers verbeteren en de talen die een betrouwbare automatische semantische analyse van de taal mogelijk maken.
De eerste type CNL talen (ook wel *"simplified languages"* of "*technical languages"* genoemd) zijn ontwikkeld voor de industrie. Het doel van deze talen is om de kwaliteit van technische documentatie te verhogen en om automatische vertaling van technische documenten makkelijker te maken. Bij deze talen worden de schrijver van een document een aantal algemene restricties opgelegd; zoals: *"schrijf korte en grammaticaal makkelijke zinnen"*, *"gebruik zelfstandige naamwoorden in plaats van voornaamwoorden"*, *"gebruik determinatoren"*, "*maak actieve zinnen in plaats van passieve zinnen"* en gebruik een voor-gedefinieerde woordenschat zonder synoniemen.
De tweede type van CNL talen hebben formele logica als basis. De talen hebben een formele syntax en semantiek. Doordat de talen gebaseerd zijn op formele logica kunnen ze ondubbelzinnig worden overgezet naar een bestaande formele taal zoals eerste order logica. De talen kunnen gebruikt worden als kennis representatie talen. Wanneer de talen gebruikt worden met een parser kan bij het schrijven van deze talen automatische consistentie en redundantie controles worden uitgevoerd (Kaljurand, 2007). Omdat de zinnen in deze vorm van CNL formeel zijn kan een zin maar één betekenis hebben. Hierdoor is het voor een parser mogelijk om de zinnen te verwerken. De focus in dit onderzoek ligt op de tweede type CNLs. Met de tweede type van CNLs is het mogelijk om requirements op te stellen zonder ambiguïteit.

In de twintigste eeuw is er veel onderzoek gedaan om beperkingen op natuurlijke talen te leggen zodat de communicatie tussen mens-mens en mens-computer verbeterd. De eerste onderzoeken die gedaan werden hadden als doel om de communicatie tussen mensen over de hele wereld te verbeteren. Een van de eerste van deze talen was Basic English. De taal is gepresenteerd in 1932. De taal had een grammatica met restricties en maakte gebruik van 850 Engelse woorden. In de jaren zeventig werden gecontroleerde subsets van Engels gebruik voor meer technische toepassingen.

Bedrijven zoals Caterpillar, Boeing, IBM en anderen definieerde hun eigen subset van het Engels dat ze gebruikte voor het maken van technische documentatie. Het doel was om de ambiguïteit te verminderen in technische documentatie en om misverstand bij het lezen van de documenten tegen te gaan. In de jaren zeventig werden ook de natuurlijke taal interfaces voor databases populair hierdoor richtte de studie van CNL zich meer binnen deze context. Doordat er binnen dit onderwerp ook veel onderzoek gedaan werd naar Natural Language Processing (NLP) en hier in die jaren nog veel onoverkomelijke problemen mee waren dreef ook het onderzoek naar CNL langzaam weg uit deze context, hoewel het onderzoek al in de jaren zeventig was begonnen naar CNL heeft het tot midden jaren negentig geduurd voordat talen zoals Attempto Controlled English in de praktijk werden toegepast. Dit gedeelte over de geschiedenis van gecontroleerde talen is een samenvatting van een beschrijving van de historie van CNLs door Kuhn (2010).

3.3.1 CNL talen met elkaar vergelijken

Er zijn veel gecontroleerde natuurlijke talen. Kuhn (2012) heeft bijvoorbeeld 95 verschillende gecontroleerde talen met elkaar vergeleken. Kuhn is als onderzoeker betrokken bij het Attempto project die de gecontroleerde taal ACE hebben ontwikkeld. Om de 95 gecontroleerde talen met elkaar te vergelijken heeft Kuhn een classificatieschema ontwikkeld met de naam PENS. PENS is een acroniem en de vier letters staan voor *Precision*, *Expressiveness*, *Naturalness*, and *Simplicity*. Aan de vier dimensies geeft Kuhn in zijn PENS schema per taal een cijfer van één tot vijf. Hierbij is vijf de hoogst mogelijke score en één de laagste score. Een taal die bijvoorbeeld op precision vijf scoort is erg nauwkeurig.
De dimensie *precision* in het schema geeft aan in welke mate de betekenis van een zin direct kan worden afgeleid van de tekstuele vorm. Een natuurlijke taal is onnauwkeurig; er is veel context informatie nodig om de betekenis van een zin te begrijpen. Formele talen zijn erg nauwkeurig omdat de betekenis van de zin direct kan worden omgezet vanuit de symbolen die gebruikt worden. De CNL talen zitten tussen een natuurlijke en formele taal in. In zijn schema geeft Kuhn met de P van precision aan in hoeverre de taal dichter bij een natuurlijke of formele taal zit.
De dimensie *expressiveness* geeft de mate van expressiviteit van een taal aan. Hiermee worden de talen vergeleken op het aantal proposities die gebruikt kunnen worden in de CNL taal. Een taal X is expressiever dan taal Y wanneer taal X alles van taal Y kan beschrijven maar niet andersom.
De dimensie *naturalness* geeft in het schema aan in welke mate de taal op een natuurlijke taal lijkt in termen van leesbaarheid en begrijpelijkheid. Het grote nadeel van formele talen is dat ze vaak nogal moeilijk te begrijpen zijn voor domeinspecialisten en stakeholders. Omdat een CNL is afgeleid van een natuurlijke taal is het voor personen die de natuurlijke taal beheersen makkelijker om deze te leren, gebruiken, lezen en schrijven in vergelijking met andere technische talen zoals een programmeertaal of een formele taal.
De *simplicity* dimensie in het PENS schema beschrijft hoe eenvoudig of complex het is om de taal om te zetten naar een formele taal. Met deze dimensie meet Kuhn hoeveel inspanning het kost om de taal in een computer programma te implementeren.

Om de CNL talen met elkaar te vergelijken heeft Wyner (2010) met een groep onderzoekers een lijst opgesteld met belangrijke eigenschappen voor CNL talen. Deze lijst met eigenschappen is opgesteld in 2009 tijdens een workshop bij de Controlled Natural Language CNL conferentie[3]. Het doel van deze workshop was om de belangrijke eigenschappen voor CNL talen op te stellen als hulp bij het maken van nieuwe CNL talen en om CNL talen met elkaar te vergelijken. In het rapport is onderscheid gemaakt tussen drie grote secties van eigenschappen: algemene eigenschappen, ontwerp eigenschappen en linguïstische eigenschappen.

3 8-10 juni 2009, Marettimo Island, Italy (http://attempto.ifi.uzh.ch/site/cnl2009)

3.3.2 *Geselecteerde CNL talen*

Om de scope van het onderzoek te beperken is gekozen om voorafgaande aan het onderzoek vier CNL talen te selecteren. In hoofdstuk vier is beschreven hoe deze talen zijn geselecteerd. De volgende CNL talen zijn onderzocht:

1. Attempto Controlled English (ACE)
2. Processable ENGlish (PENG)
3. Common Logic Controlled English (CLCE)
4. Computer Processable Language (CPL)

Clark (2010) beschrijft dat er twee verschillende filosofieën zijn bij het ontwikkelen van een gecontroleerde natuurlijke taal. Bij een *"naturalist"* benadering wordt de gecontroleerde taal ontwikkeld als een eenvoudiger vorm van de natuurlijke taal waarin ambiguïteit nog steeds voorkomt. Bij deze benadering zijn meerdere interpretaties van een zin mogelijk. Voorbeelden van deze benadering zijn CPL en SBVR.
Bij een *"formalist"* benadering lijkt de gecontroleerde taal meer op een formele taal of programmeertaal. In de taal komt minimale ambiguïteit voor. De taal is makkelijker in gebruik dan een formele taal. Het doel van deze talen is om formele talen gebruiksvriendelijke te maken. Voorbeelden van deze benadering zijn ACE, PENG en CLCE.

3.3.3 *Attempto Controlled English (ACE)*

In de wetenschappelijke literatuur is er veel gepubliceerd over Attempto Controlled English (ACE). Al meer dan 14 jaar wordt er actief ontwikkelt aan ACE door het Attempto team, een onderzoeksgroep van de universiteit van Zurich[4]. In 1995 is ACE voor het eerst geïntroduceerd door Fuchs en Schwitter sindsdien zijn er al veel wetenschappelijke artikelen over verschenen. Door het grote aantal jaren dat er ontwikkelt is aan ACE en de vele publicaties die er verschenen zijn, is ACE één van de meest bekende CNL in de academische wereld geworden (Kuhn, 2010).
Het Attempto onderzoek is gestart met de visie om de voordelen van natuurlijke en formele taal met elkaar te combineren. Hiermee willen de ontwerpers tegemoet komen aan de behoefte van de domeinspecialisten. Het uitgangspunt bij de ontwikkeling van ACE is dat natuurlijke taal op een nauwkeurige manier gebruikt kan worden. Voorbeelden van het nauwkeurig gebruik van een taal zijn juridische taal en de zogenaamde gecontroleerde talen die gebruikt worden voor technische documentatie en automatische vertaling. De regels die achter deze talen liggen zijn vaak ad-hoc en niet formeel. Met deze talen rekening houdend is het attempto project gestart om de specificatie taal ACE te ontwikkelen (Fuchs, Schwertel, & Schwitter, 1990).

ACE is een subset van het standaard Engels met een domein specifieke woordenschat en een grammatica die beperkt is met een aantal constructie- en interpretatieregels (Fuchs, Schwertel, & Schwitter, 1999). Met ACE kunnen gebruikers nauwkeurige specificatie maken in de termen van het applicatiedomein. ACE specificatie kunnen door een computer verwerkt worden en ondubbelzinnig vertaald worden naar een formele taal. De zinnen die gemaakt zijn met ACE zijn tekstuele weergave van formele zinnen in de logica. Doordat ACE zinnen gebaseerd zijn op natuurlijke taal lijken de zinnen niet formeel, maar schijn bedriegt. De ACE zinnen zijn formeel en hierdoor verwerkbaar door een computer (Fuchs & Schwitter, 1996).
ACE is een nauwkeurige gedefinieerde subset van het Engels die door een computer kan worden vertaald naar eerste orde logica. Hierdoor is ACE een eerste orde logica taal met als syntax een subset van het Engels waardoor ACE leesbaar is voor mens en computer. Door de Attempto groep is een tool ontwikkeld waarmee ACE zinnen gemaakt kunnen worden. De tool, Attempto Parsing Engine (APE), kan ACE zinnen vetalen naar eerste orde logica. De tool maakt hiervoor gebruik van de discourse representation structures (DRS) techniek. De zinnen die gemaakt worden met ACE worden door het Attempto systeem vertaald in DRS en optioneel in de logische programmeertaal Prolog. Een DRS is een gestructureerde vorm van eerste orde predikaat logica die discourse referenten bevat die de discourse representeert, en de condities voor deze discours referenten (Schwitter & Fuchs, 1996). Elke

4 http://attempto.ifi.uzh.ch/site/

zin wordt vertaald in het kader van de voorgaande zinnen. Hierdoor kunnen zinnen die verwijswoorden bevatten verwerkt worden in de discours van de opgegeven tekst.

De constructie- en interpretatieregels van ACE zijn makkelijk te gebruiken en te herinneren doordat deze veel op de Engelse grammaticaregels lijken. Om de taal te leren is alleen basiskennis nodig van de Engelse grammatica. Fuchs (1990) schrijft dat aan de andere kant wel in gedachten moet worden gehouden dat ACE een formele taal is die, net zoals andere formele talen, geleerd moet worden. Bedrijven zoals Boeing en Caterpillar maken al jaren gebruik van gecontroleerde talen voor het maken van technische documentatie. De ervaring van deze bedrijven is dat een gecontroleerde taal in een paar dagen geleerd kan worden. De gebruiker is in een aantal weken competent om de taal te gebruiken. Hierdoor, schrijft Fuchs, claimen wij dat domeinspecialisten niet zoveel moeite hoeven te doen om de ACE regels te leren toe te passen dan ze moeten doen om een formele taal te leren.
In de praktijk blijkt dat het aanleren van de constructie- en interpretatieregels twee dagen kost. Om vloeiend met ACE te leren werken kost uiteraard meer tijd (Fuchs, Kaljurand, & Kuhn, 2008). In een experiment van Bernstein en Kaufmann (2010) moesten CS studenten gebruik maken van ACE in plaats van SQL als query taal. De studenten waren in staat om in twee tot drie dagen ACE te leren en daarmee queries te maken.

Syntax
De syntax van ACE wordt beschreven in minder dan twintig constructieregels. De constructieregels beschrijven de syntax van ACE. De regels beperken de vorm van al de toelaatbare zinnen in ACE. Elke ACE zin is een syntactisch acceptabele Engelse zin, maar niet iedere Engelse zin is een ACE zin.

Woordenschat
Om een tekst te kunnen analyseren moet de betekenis van een woord door een computer begrepen worden. ACE beschikt over een eigen woordenschat. Hierdoor is het mogelijk dat de teksten die geschreven zijn in ACE verwerkt kunnen worden door een computer. In ACE wordt onderscheid gemaakt tussen twee groepen van woorden:

- Voor-gedefinieerde functiewoorden; dit zijn: determinatoren, voorzetsels, voornaamwoorden en voegwoorden.
- Domein specifieke content woorden; dit zijn: zelfstandige naamwoorden, werkwoorden, bijvoeglijke naamwoorden en bijwoorden.

De functiewoorden zitten ingebouwd in ACE en kunnen niet gewijzigd worden. Voorbeelden van functiewoorden zijn: "*if*", "*every*", "*or*" en "*is*". De content woorden kunnen door een domeinspecialist worden toegevoegd. Deze woorden kunnen vrij worden gedefinieerd. Het is alleen niet toegestaan dat content woorden spaties bevatten. Voor domeinwoorden kunnen synoniemen worden toegevoegd. Door deze vorm van werken is het mogelijk dat er in een zin lexicale ambiguïteit voorkomt. De verantwoordelijkheid om dit te voorkomen ligt bij de domein expert, die moet beslissen welke woorden er worden toegevoegd en welke betekenis deze woorden hebben in de context van de zin (Fuchs & Schwitter, 1996).

In de constructieregels staat de grammatica van ACE beschreven. Hieronder volgen een aantal voorbeelden van belangrijke regels die betrekking hebben op hoe woorden in ACE gebruikt mogen worden; de lijst is overgenomen van Fuchs (1990).

- Werkwoorden moeten worden gebruikt in de onvoltooid tegenwoordige tijd, de actieve vorm, aantonende wijs of de derde persoon enkelvoud.
- Geen modale werkwoorden *(may, can, must etc.)* of intensionele werkwoorden *(hope, know, believe etc)*.
- Geen modale bijwoorden *(possibly, probably etc)*.
- ACE geeft de gebruiker de mogelijkheid om synoniemen te definiëren *(alarm signal button, signal button)* en afkortingen *(ABS betekent alarm signal button)*.
- Content woorden kunnen eenvoudig zijn (*train)* of een samenstelling *(alarm signal button, alarm-signal button)*.

Grammatica
De grammatica van ACE is beschreven in de constructieregels. In ACE zijn er twee soorten zinnen toegestaan: vragende en declaratieve zinnen. Iedere declaratieve zin moet, net zoals in een natuurlijke taal, eindigen met een punt. Een declaratieve zin kan een enkelvoudige of een complexe / samengestelde zin zijn (Fuchs, Kaljurand, & Kuhn, 2009). Een enkelvoudige zin in ACE heeft de volgende algemene structuur:

onderwerp + werkwoord + [complement] + {toevoegsels}

Iedere zin in ACE heeft een onderwerp en een werkwoord. Een complement is nodig voor een transitief werkwoord en een distransitief werkwoord, toevoegsels zijn altijd optioneel (Pretorius & Schwitter, 2009). Als voorbeeld de volgende requirement:

"A patient inserts a password into the login form".

In deze requirement is "*a patient*" het onderwerp van de zin. Het werkwoord "inserts" geeft aan welke handeling de patiënt moet verrichten. Het werkwoord "inserts" is een transitief werkwoord omdat het complement "*a password*" aan dit werkwoord verbonden is. Als laatste wordt er nog als toevoegsel bijgevoegd waarin de handeling wordt verricht; in "the login form".

Een enkelvoudige zin bestaat uit een zelfstandig naamwoord en een werkwoord of bestaat uit een "er is / zijn" en een zelfstandig naamwoord. De zin "*de machine werkt*" is een voorbeeld van een enkelvoudige zin. Deze zin bestaat uit een zelfstandig naamwoord "*de machine*" en het werkwoord "*werkt*". Een samengestelde zin wordt recursief gevormd door ten minsten één constructor. Een constructor is een coördinaat, onderschikkend voegwoord of een kwantificering. Een voorbeeld van een samengesteld zin is *"Als de machine werkt dan hoeft Jan niet te wachten"*.

Elke vraag eindigt met een vraagteken in ACE. De zin *"Moet Jan wachten?"* is een voorbeeld van een ja/nee vraag zin. De zin *"Wie wacht er?"* is een voorbeeld van een waarom vraagzin. Een opdrachtzin begint in ACE met een geadresseerde en eindigt met een uitroepteken. Een voorbeeld van een opdrachtzin is *"Jan wacht!"*. Een ACE tekst moet tenminste uit één zin bestaan met een zelfstandig naamwoord waarnaar kan worden verwezen door een anaforische referentie. Elke anaforische referentie is een zelfstandig naamwoord zin of is een voornaamwoord, variabelen of een eigen naam (Fuchs et al., 2009).

De betekenis van een anafoor of een verwijswoord wordt in ACE op een syntactische wijze afgehandeld. Omdat ACE geen kennis heeft van de betekenis van de woorden worden verwijswoorden op een syntactische wijze afgehandeld. Wanneer er in een ACE zin een verwijswoord voorkomt wordt in de voorgaande tekst gezocht waarnaar het verwijswoord verwijst. In het meest eenvoudige geval is het verwijswoord een persoonlijk voornaamwoord en de voorgaande zin is een zelfstandig naamwoord zin waarnaar verwezen wordt. ACE zoekt in de voorgaande tekst naar verwijzingen die hetzelfde nummer en geslacht hebben als het verwijswoord. Bij het gebruik van verwijswoorden moet een gebruiker hiermee rekening houden (Fuchs & Schwitter, 1996). In ACE zijn verwijswoorden alleen toegestaan als de verwijzingen kunnen worden gevonden in de voorgaande tekst (Kuhn, 2010).
Om meervoud, verkleining, samenstelling, verleden tijd en nog andere statussen aan te geven zijn in een natuurlijke taal morfologische regels van toepassing. Door deze regels kan een kleine wijziging in een woord een zin een andere betekenis geven. In ACE is het alleen toegestaan om zinnen in de tegenwoordige tijd te maken. Het gebruik van meervoud wordt wel ondersteunt met ACE, bij telwoorden mag bijvoorbeeld gebruik worden gemaakt van meervoud (book/books).

<u>Semantiek</u>
ACE heeft twintig interpretatieregel. In de regels staat beschreven hoe de betekenis van een zin tot stand komt. Elke ACE zin heeft maar één betekenis. Omdat ACE gebaseerd is op het Engels kan een ACE zin in het Engels verscheidene betekenissen hebben. In de interpretatieregels staat beschreven welke van de betekenissen de juiste is volgens ACE. Kort gezegd wordt er met de interpretatieregels de semantiek bepaald van de ACE zinnen. De interpretatieregels voorkomen ambiguïteit in ACE zinnen die niet kan worden opgelost met de constructieregels. Een aantal belangrijke interpretatieregels zijn (Fuchs et al., 1990):

- Werkwoorden duiden gebeurtenissen aan (*"press"*) of een status (*"be"*).
- De tekstuele orde van werkwoorden bepaalt de standaard orde van de gebeurtenissen en de statussen.
- Voorzetsel groepen in een aanvullende positie wijzigen altijd het werkwoord en geeft meer informatie over de locatie of de gebeurtenis (*"the driver {stops the train in a station}"*).
- Een anaforische referentie of verwijswoord is mogelijk met voornaamwoorden of met naamwoord zinnen.
- De scope van een kwantor (*a, there is a, every, for every*) strekt zich uit tot het einde van een zin, elke volgende kwantor is automatische in de scope van de vorige kwantor.

Vertaling
De zinnen die gemaakt worden met ACE kunnen worden omgezet naar eerste orde logica (FOL). In ACE is het mogelijk om verwijzing te maken naar andere zinnen. Om verwijswoorden op een correcte manier om te zetten is het noodzakelijk dat de manier waarop de woorden naar elkaar verwijzen ook wordt omgezet naar een FOL. In predicaat logica worden twee zinnen gezien als twee verschillende formules. Als voorbeeld de volgende zinnen *"A passenger enters a train"* en *"The train leaves a station"*. Dit levert de volgende twee zinnen in predicaat logica op:

$$\exists X \exists Y (passenger(X) \bigwedge train(Y) \bigwedge enter(X,Y))$$
$$\exists U \exists W (train(U) \bigwedge station(W) \bigwedge leave(U,W))$$

In de twee voorbeeldzinnen is er geen referentie tussen *"a train"* en *"the train"* terwijl dit over hetzelfde object gaat. In ACE is dit probleem opgelost met de discourse representation theory (DRT). Met deze methode worden anaforische referenties op een systematische manier met elkaar gecombineerd naar één propositie. DRS is een variant van predicaat logica die een tekst weergeeft als een logische eenheid, een discourse representation structure (DRS) genoemd. Elk deel van een ACE zin heeft een aantal logische voorwaarden voor de DRS met behulp van voorgaande zinnen als context om anaforische verwijzingen mogelijk te maken. Een DRS kan gezien worden als drs(U,Con) waar U een lijst met discourse referenten is en Con een lijst met condities is voor de discourse referenten. De discourse referenten zijn gekwantificeerde variabelen die staan voor objecten in het opgegeven domein. De voorwaarde vormen beperkingen waar aan de discourse refrenten moeten voldoen om een DRS waar te maken. Complexe DRS condities worden recursief gebouwd vanuit andere DRS's. Als voorbeeld: "if then (DRS1,DRS2), or (DRS1,DRS2,...), not(DRS)" (Fuchs et al., 1990).

<u>Voorbeeld requirement in ACE</u>
Om te tonen hoe ACE werkt zal met een voorbeeld worden weergegeven hoe een requirement kan worden omgezet naar ACE. Het voorbeeld requirement is:

"Members are assigned to one practice, but can also work in other practices with the same network."

In dit requirement zijn twee verschillende requirements verwerkt. Dit zijn de volgende:

1. *Members are assigned to one practice.*
2. *Members can work in other practices with the same network.*

In de tweede requirement is het niet duidelijk of de members werken met hetzelfde netwerk als waar zij normaal mee werken. Het kan bijvoorbeeld zijn dat zij bij de andere praktijk werken waar zij kunnen in loggen op het netwerk waarop zij normaal werken. De tweede requirement kan ook als een regel worden geïnterpreteerd. Members mogen alleen bij een andere praktijk gaan werken als die praktijk gebruik maakt van hetzelfde netwerk als waar zij normaal op werken. De volgende interpretaties zijn mogelijk:

1. *Members can work in {other practices} with {the same network}.*
2. *Members can work in {other practices with the same network}.*

Met ACE kunnen de volgende requirements worden gemaakt gebaseerd op de twee mogelijke betekenissen van de requirement. Het verschil zit in het gebruik van het woord "*with*". Bij de tweede requirement is dit woord vervangen door "*that has";* dit zijn de volgende mogelijkheden:

1. *The members are assigned to one practice. The member can work in a other practice with the same network.*

2. *The members are assigned to one practice. The member can work in a other practice that has the same network.*

Bij het eerste voorbeeld wordt door de APE[5] tool de volgende paraphrase gegenereerd:

- *There are at least 2 members X1.*
- *The members X1 are assigned to a practice.*
- *There is a member X1.*
- *It is possible that the member X1 works with a same network in an other practice.*

In deze paraphrase vallen een paar dingen op. In de requirement is gebruik gemaakt van meervoud bij het definiëren van de member. Hierdoor herkent de APE tool dat het gaat om minstens twee verschillende members die "*assigned*" zijn "*to a practice*". Het woord "*can*" is door APE omgezet naar *"It is possible that*". Bij het tweede deel van de requirement is de minimal attachment regel uitgevoerd; er is gekozen voor de meest eenvoudige syntactische structuur als betekenis van de requirement. Bij het tweede voorbeeld wordt door de APE tool de volgende paraphrase gegenereerd:

- *There are at least 2 members X1.*
- *The members X1 are assigned to a practice.*
- *There is a member X1.*
- *It is possible that the member X1 works in an other practice X2 and that the other practice X2 has a same network.*

In deze paraphrase valt op dat de laatste zin anders is dan bij de voorgaande parapharse. Dit komt doordat in de invoer *"that has"* is gebruikt in plaats van "*with*". Door APE wordt nu de right association regel toegepast.

3.3.4 *Processable ENGlish (PENG)*

Een CNL taal die vergelijkbaar is met ACE is Processable ENGlish (PENG). PENG is ontwikkeld door een onderzoeksgroep van het *Centre for Language Technology* van de Macquarie University in Sydney. Rolf Schwitter, één van de ontwikkelaars van ACE, is al tien jaar nauw betrokken bij het PENG project. Doordat Schwitter aan het begin heeft gestaan van zowel ACE als PENG zijn er in deze twee talen veel overeenkomsten te vinden.
PENG is een machine georiënteerde CNL taal die ontworpen is voor niet-specialisten om nauwkeurige specificatie teksten te schrijven in een schijnbaar informele notatie. Om te garanderen dat de taal om een efficiënte manier gebruikt wordt is bij deze taal ook een teksteditor ontwikkeld. Voor de teksteditor is een intelligent feedback mechanisme ontwikkeld dat de schrijver door het schrijf proces heen leidt. Hierdoor is het mogelijk dat er goed gevormde zinnen ontstaan die zonder ambiguïteit kunnen worden omgezet naar eerste-orde logica door discourse representation structures (DRS) (Schwitter, 2004).

PENG kan worden gebruikt als een specificatie en kennisrepresentatie taal. Specificatieteksten die geschreven zijn in PENG lijken informeel, net zoals natuurlijk Engels, maar in tegenstelling tot natuurlijk Engels is de PENG taal ontworpen om dezelfde nauwkeurigheid van een formele taal te hebben. Dit is gedaan door alleen zinsconstructies in PENG toe te staan die niet ambigu zijn en door alleen bepaalde werkwoordsvormen toe te staan. In PENG kunnen werkwoorden alleen gebruikt worden in de tegenwoordige tijd en de aantonende wijs. Hulpwerkwoorden (zoals *kan* en *moet*) en intensionele werkwoorden (zoals *geloof* en *wil*) zijn niet toegestaan in PENG omdat bij deze woorden kennis van het domein nodig is (Schwitter & Tilbrook, 2006). Bij het ontwerpen van PENG is goed nagedacht zodat PENG makkelijk voor mensen te lezen en te gebruiken is.

Syntax

PENG maakt gebruik van een strikte subset van het Engels. In de gecontroleerde grammatica en lexicon staat beschreven waaraan PENG zinnen moeten voldoen. In tegenstelling tot andere CNL talen is bij het gebruik van PENG geen kennis nodig van de regels voor grammatica en woorden. De tool die gebruikt wordt met PENG herkent de regels terwijl de zinnen worden ingevoerd. Hierdoor kan de tool voorstellen doen aan de gebruiker hoe een zin opgesteld moet worden. De lexicon van

5 Bij de voorbeelden is gebruik gemaakt van de online APE-tool: http://attempto.ifi.uzh.ch/site/tools/

PENG bestaat uit domeinwoorden die ondertussen bij het gebruik van de tool kunnen worden toegevoegd aan de lexicon.

Woordenschat
De gecontroleerde lexicon van PENG bestaat uit een basis woordenschat en een gebruiker gedefinieerde woordenschat. In de basis woordenschat zijn de meest voorkomende woorden opgenomen vanuit het Engels. Dit zijn de volgende soort woorden: eigenamen, soortnamen, werkwoorden, bijvoeglijke naamwoorden en bijwoorden. Daarnaast zijn er in de basis lexicon ook voor-gedefinieerde functiewoorden opgenomen. In natuurlijke taal zijn functiewoorden de woorden met weinig inhoudelijke betekenis. De functiewoorden dienen in de eerste plaats voor een goede syntactische structuur. Dit zijn de volgende soort woorden: determinatoren (een bij een zelfstandig naamwoord horend functiewoord), conjuncties (dit zijn de woorden die zinnen aan elkaar verbinden), voorzetsels, voegwoorden, ontkenningen en ondubbelzinnig makende structuren. In de basis lexicon zijn ook woorden opgenomen die niet gebruikt mogen worden in PENG zinnen. Dit zijn vooral intensionele woorden. Intensionele woorden zijn de woorden die niet direct op de werkelijkheid slaan maar andere woorden omschrijven. De gebruikers lexicon kan door de gebruiker worden uitgebreid met domein specifieke content woorden. Het uitbreiden kan worden gedaan terwijl de gebruiker bezig is met het schrijven van de zin. Met de PENG tool kan een gebruiker synoniemen, acroniemen en afkortingen definiëren voor domeinwoorden (Schwitter, 2004) (Schwitter & Tilbrook, 2006).

Grammatica
De grammatica van PENG definieert hoe woorden gecombineerd moeten worden om enkelvoudige, complexe en vragende zinnen te maken. In het geval van requirements zijn vooral de enkelvoudige en complexe zinnen van belang.
Elke enkelvoudige zin in PENG heeft een hiërarchie die bestaat uit woorden en bestanddelen. Elk woord is zelf ook een bestanddeel. Verschillende bestanddelen kunnen samengevoegd worden op een gecontroleerde wijze om zo een enkelvoudige PENG zin te vormen (Schwitter & Tilbrook, 2006). Op het hoogste niveau worden PENG zinnen op de volgende manier samengesteld:

Onderwerp + predicator + complement + {toevoegsels}

De structuur van de PENG zinnen is hetzelfde als van de zinnen die gemaakt worden in ACE. Het volgende voorbeeld geeft aan hoe een PENG zin is opgebouwd.

"The administrator reboots the webserver every day."

In deze zin is *"the administrator"* het onderwerp van de zin. Het werkwoord *"reboots"* geeft de handeling aan die de administrator verricht. Het werkwoord *"reboots"* is een transitief werkwoord omdat het complement *"the webserver"* aan dit werkwoord verbonden is. In het toevoegsel van de zin wordt beschreven wanneer de handeling wordt verricht, *"every day"*.
Met PENG kunnen enkelvoudige en complexe zinnen worden opgesteld. In de volgende voorbeelden is te zien hoe er van een enkelvoudige zin een complexe zin wordt gemaakt.

1. *The user logs in.*
2. *The user logs in on the webpage.*
3. *The user logs in on the webpage every hour.*
4. *The user logs in on the webpage every hour and adds a new patient.*

In zin één is de meest eenvoudige structuur van een PENG zin te zien. De zin bestaat uit een onderwerp en een werkwoord. In zin twee wordt aan het werkwoord het complement *"on the webpage"* toegevoegd. In zin drie wordt er aan de zin een toevoegsel toegevoegd: de handeling wordt elk uur verricht. In de vierde zin wordt de zin complexer omdat er gebruik wordt gemaakt van een koppelwoord. In zin vier worden er twee activiteiten met elkaar verbonden.

Semantiek
In tegenstelling tot ACE is het bij PENG niet nodig dat de auteur kennis heeft van de grammaticaregels. PENG maakt gebruik van een tool (ECOLE) die vooruit kijkt terwijl de auteur een zin opstelt. Na ieder woord die de auteur in de tool intypt geeft de tool aan welk soort syntactische constructie de auteur kan gebruiken om de zin verder op te stellen. Op deze manier wordt de auteur door de zinnen heen geleidt en is het niet expliciet nodig dat de auteur kennis van de regels heeft. De

tool genereert en garandeert niet alleen dat de auteur goed gevormde zinnen opstelt, maar biedt ook de nodige structuur voor de semantische interpretatie van de zin (Schwitter, 2002). Door deze functionaliteit is het gebruik van de PENG tool te vergelijken met een ontwikkeltool voor het programmeren van software code.

Om ambiguïteit in zinnen te voorkomen beschikt PENG over een aantal interpretatieregels zodat elke zin maar één betekenis heeft. PENG heeft zeven interpretatie principes. De principes worden in deze paragraaf beschreven. De uitleg van de principes komt uit het artikel *"English as a formal specification language"* van Schwitter (2002).
Anaforaprincipe: In PENG kan gebruik worden gemaakt van anaforen of anders genoemd verwijswoorden. Verwijswoorden refereren altijd naar het meest recent toegankelijke zelfstandige naamwoord die geschikt is. Dit houdt in dat een woord van hetzelfde nominale type is, tenminste hetzelfde bijvoeglijke naamwoord heeft, een prepositie of een appositie (een zinsdeel zonder gezegde dat direct achter het woord of de woordgroep waar het naar verwijst staat) naam als het verwijzende zelfstandige naamwoord heeft. Wanneer er geen antecedent in de tekst gevonden kunnen worden dan wordt het bestaan van de entiteit veronderstelt. Zelfstandige naamwoorden zijn toegankelijk vanuit elke positie in de tekst.
Modificatie principe: In PENG worden voorzetsel zinnen als modifier gebruikt voor het dichtstbijzijnde voorgaande werkwoord zinsdeel en niet voor het zelfstandig naamwoord zinsdeel ("minimal attachtment"). Als voorbeeld de zin "*The wolf {catches the bird in the garden}*" heeft in PENG de betekenis van "*The wolf catches {the bird that is in the garden}*".
Distributie principe: Wanneer een complement van een (negatief) werkwoord bestaat uit een coördinatie van een zin dan geldt het werkwoord voor al de delen van de zin. Als voorbeeld de zin "*The bird is yellow and green*". Deze zin heeft in PENG de betekenis *"The bird is yellow and [is] green"*. Als voorbeeld ook de volgende zin: "*The wolf does not eat a frog and a bird*" die in PENG deze betekenis heeft: "*The wolf does nor eat a frog and [does not eat] a bird.*"
Bewerkingsvolgorde principe: Op dezelfde manier als in de eerste orde predicaat logica wordt er in PENG gebruik gemaakt van de bewerkingsvolgorde; welke elementen in een zin bij elkaar horen. In PENG wordt gebruik gemaakt van de volgende hiërarchie:

"*Negatie -> Conjunctie -> Disjunctie -> Implicatie*".

Coördinatie principe: Standaard hoort een samengevoegde werkwoord zin bij het hoofdzinsdeel en niet bij de relatieve zin. Als voorbeeld de volgende zin waarin dit duidelijk wordt: *"The wolf {catches a bird that is yellow} and {eats a worm}"*.

Scope principe: In PENG staat de scope van een kwantor vast. De scope begint bij de tekstuele positie van het gekwantificeerd zelfstandig naamwoord en loopt door tot het einde van de zin. De constructor voor *"for every"* en *"there"* geeft de auteur de mogelijkheid om het gekwantificeerd zelfstandig naamwoord te verplaatsen om deze op een zodanige wijze een bredere scope te geven. Als voorbeeld de zin *"A wolf eats every bird"* moet opnieuw op de volgende manier worden geformuleerd om het gekwantificeerd zelfstandig naamwoord een bredere scope te geven: "*For every bird there is a wolf that eats the bird*".

Temporele orde principe: De tekstuele orde van werkwoorden bepaalt de standaard orde van de onderliggende eventualiteiten. Een temporele subordinator zoals "*while*" of "*before*" kan de standaard orde veranderen. Omdat PENG zinnen standaard een lineaire temporele orde hebben heeft de zin "*The fox chases a cat and eats a bird*" dezelfde betekenis als de zin "*Before the fox eats a bird, the fox chases a cat*".

<u>Voorbeeld requirement in PENG</u>
In deze paragraaf zal een voorbeeld requirement worden omgezet naar PENG. Hiermee wordt aangetoond dat de ambiguïteit met PENG wordt opgelost. Het voorbeeld requirement is de volgende:

"The different tests and measurements done by the technician can be different in multiple practices."

Het voorbeeld requirement is ambigu en vaag. In deze requirement komt twee keer een coördinatie ambiguïteit voor. In de requirements is het is niet duidelijk of de beperkende bijwoorden *"different"* en *"done"* van toepassing zijn op beide woorden *"test"* en *"measurements"* of dat hier gelezen moet worden *"different tests"* en *"measurements done by ..."*. De betekenis van het requirement kan zijn *"The*

[different test] and [measurements done by the technician] can be different in multiple practices" of *"The [different test done by the technician] and [different measurements done by the technician] can be different in multiple practices"*.
Deze requirement is omgezet naar PENG. In PENG zijn er twee zinnen van gemaakt omdat het over twee verschillende feiten gaat. In de requirement in PENG wordt uitgedrukt dat dezelfde technician de testen en measurements uitvoert. In PENG is de requirement als volgt:

"A technician can measure on more than 1 practice. The technician can test on more than 1 practice."

In de eerste zin is gebruik gemaakt van *"a"* om een nieuw object te introduceren. In de tweede zin wordt gebruik gemaakt van *"The"* om te verwijzen naar het object en aan te geven dat het om dezelfde technician gaat. In de requirement is het zinsdeel "*different in multiple practices*" vervangen in "*more than 1 practice*".

3.3.5 *Common Logic Controlled English (CLCE)*

Common Logic Controlled English is ontworpen door John Sowa. De taal is ontworpen als human interface taal voor de Common Logic ISO[6] standaard. CLCE maakt zelf geen deel uit van deze standaard, maar maakt alleen gebruik van de semantiek van de standaard (Schwitter, 2010). John Sowa heeft CLCE nog niet volledig uit ontwikkeld. Sommige delen van de taal zijn nog in ontwikkeling.

Sowa heeft CLCE ontwikkeld met als doel om een taal te ontwikkelen die zo dicht mogelijk bij nauwkeurig en precies Engels zit. Doordat zinnen in CLCE bijna formeel zijn is het mogelijk om de zinnen automatische te vertalen naar een eerste orde logica (FOL). De syntax die gebruikt wordt in CLCE is gelijk aan het soort Engels wat gebruikt wordt in softwaredocumentatie en wiskunde boeken. Sowa geeft aan dat iedereen die Engels kan lezen ook in staat is om teksten te lezen die gemaakt zijn met CLCE zonder speciale training. Het moeilijkste is om CLCE zinnen op te schrijven die voldoen aan de semantische restricties van FOL. De restricties zijn gelijk aan de restricties die worden gebruikt in database query talen, software design of formele specificatie talen. Voorbeelden hiervan zijn SQL, UML en OWL. Deze talen kunnen automatische worden vertaald naar en van FOL. Het is hierdoor ook mogelijk om deze talen te vertalen naar een zin in CLCE. Ondanks de algemene opzet van CLCE is het niet de intentie om van CLCE een officiële standaard te maken. De standaard is eerste orde logica. CLCE is gewoon een handige notatie om FOL makkelijker te lezen en te schrijven. Hierdoor kan CLCE worden gebruikt als taal voor het lezen van softwaredocumentatie (Sowa, 2004).

Syntax

CLCE is opgesteld volgens de Extended Backus-Naur Form (EBNF) structuur die gespecificeerd is door de Internationale Standaard ISO/IEC 14977. EBNF is een metasyntax notatie die gebruikt wordt om op een formele manier programmeer en andere formele talen te beschrijven. EBNF schrijft voor om drie soorten van grammaticaregels te definiëren: lexicale, syntactische en declaratieregels. In de lexicale regels zijn de categorieën van woorden gedefinieerd die voor mogen komen in een CLCE zin. In de syntactische regels is gedefinieerd hoe de woorden met elkaar gecombineerd moeten worden tot een CLCE zin. De declaratieregels beschrijven hoe de statements gedefinieerd moeten worden die de CLCE mapping naar FOL vaststelt (Sowa, 2007).

Woordenschat

De woorden die gebruikt worden in CLCE kunnen worden onderverdeeld in twee groepen. Er is een kleine groep met daarin een aantal gereserveerde woorden. De gereserveerde woorden worden op dezelfde manier gespeld als de Engelse woorden, maar de betekenis van de woorden is beperkt tot één betekenis of een klein aantal betekenissen die af te leiden zijn uit de syntax. Er zijn tien soorten gereserveerde woorden:

1. Booleaanse operatoren: *not, and, either, or, neither, nor, if ,then.*
2. Kwantoren: *a, an, some, something, someone, every, everything, everyone, no, nothing, no one.*
3. Speciale werkwoorden: *is, has, have, does.*
4. Vraagzinnen: *who, what, when, where, wich.*

6 http://cl.tamu.edu

5. Betrekkelijk voornaamwoord: *that.*
6. Lidwoord: *the.*
7. Koppelwoord: *and*.
8. Speciale koppelwoorden: *none, others, nothing else, no one els*.
9. Argumenten: *of, than, as.*
10. Speciale deelzinnen: *there is, such that, only if, of and only if, it is false that, it is true that.*

Naast de gereserveerde woorden kan in CLCE gebruikt worden gemaakt van domeinwoorden.

Grammatica
De zinnen in CLCE gebruiken een subset van het Engels. De zinnen die gemaakt worden in CLCE lijken op het natuurlijk Engels. In CLCE heeft elke zin één of meerdere clausules. Elke clausule heeft een hoofd werkwoord. Met CLCE kunnen enkelvoudige, complexe of samengestelde zinnen worden gemaakt. Een zin kan declaratief, vragend, of een regel zijn.

De grammatica van CLCE is gebaseerd op standaarden van de eerste orde logica. Bij het maken van een CLCE moeten de regels van de logica worden gevolgd. Zinnen die gemaakt zijn met CLCE lijken als het om de structuur van de zinnen gaat erg op een eerste orde logica zin. Een voorbeeld hiervan is de volgende zin. Het voorbeeld is overgenomen van Sowa (2004):

CLCE: *Some person is between a rock and a hard place*
FOL: $\exists x \in person \exists y \in rock \exists z \in place(Between(x, y, z) \bigwedge Hard(z))$

Dit voorbeeld maakt gebruik gemaakt van de existentiekwantor om het woordje "*some*" uit te drukken. Na het woord "*between*" komt een lijst van twee items die iets zeggen over de plaats van de persoon.

Semantiek
De fundamentele semantische beperking in CLCE is dat de betekenis van elke CLCE zin wordt gedefinieerd door de vertaling naar een eerste orde logica (Sowa, 2004). De betekenis van de omgezette zin in eerste orde logica is dus de betekenis die een CLCE zin heeft.

Voorbeeld requirement in CLCE
In deze paragraaf wordt een voorbeeld gegeven van een requirement in CLCE. De volgende requirement komt uit één van de requirementsdocumenten van VitalHealth:

> *"The summary overview will display the latest medications and information in a single view."*

In het voorbeeld requirement komt ambiguïteit voor van het type coördinatie ambiguïteit. In het requirement kan het woord *"latest"* ook worden verbonden aan *"information"*. Het omzetten van het requirement naar CLCE levert het volgende resultaat op:

> *"A summary_overview displays the latest_medications in a single view and latest_information in a single view"*

In CLCE wordt coördinatie ambiguïteit voorkomen door het bijvoeglijke naamwoord te verbinden aan het zelfstandig naamwoord; daarom zijn "*latest_medications*" en "*latest_information*" geïntroduceerd.

3.3.6 *Computer Processable Language (CPL)*
Computer Processable Language (CPL) is een CNL taal die gebaseerd is op het Engels. CPL is ontworpen voor het Amerikaanse luchtvaartbedrijf Boeing. De taal is een volwassen uitgebreide taal die wordt gebruikt in menig project (Schwitter, 2010). In tegenstelling tot de andere CNL talen is in CPL ambiguïteit op kleine schaal toegestaan. Hierdoor kan CPL gezien worden als een wat minder strikte CNL taal (Kuhn, 2010).

Clark, één van de onderzoekers die CPL ontwikkeld hebben, noemt gebruiksvriendelijkheid één van de uitdagingen die nog verbeterd moeten worden aan CPL. Een gebruiker moet ervaren zijn om goed met CPL te kunnen werken. Bij het opstellen van een CPL zin in de tool, die gebruikt wordt voor CPL, moet de gebruiker nog rekening houden met de semantiek van de zin. Net zoals elke interpretatie tool is de tool nog niet 100% perfect waardoor de gebruiker moet leren om zinnen zo op te stellen zodat deze door de tool op een juiste manier worden geïnterpreteerd. Wanneer een zin niet goed door de

tool wordt geïnterpreteerd moet de gebruiker in staat zijn om de zin op dusdanige wijze aan te passen zodat deze wel de juiste betekenis krijgt (Clark, Harrison, Jenkins, Thompson, & Wojcik, 2005).

Syntax

Om zinnen op te stellen in CPL moeten gebruikers een set van richtlijnen volgen. In de richtlijnen is beschreven hoe woorden en structuren in een zin gebruikt moeten worden.

Woordenschat

Om lexicale ambiguïteit in woorden tegen te gaan gebruikt CPL WordNet als ontologie. WordNet is een grote database waarin Engelse woorden zijn opgeslagen zoals: zelfstandig naamwoorden, werkwoorden, voorzetsels en bijwoorden. Deze woorden zijn gegroepeerd in sets van cognitieve synoniemen. Bij het invoeren van een woord in CPL wordt uit de WordNet database de meest in de buurt komende betekenis opgezocht. De betekenis die het meest in de buurt van het woord komt wordt gepresenteerd aan de gebruiker voor verificatie of voor correctie. Clark (2005) beschrijft dat deze benadering het meest adequaat is voor het doel wat ze willen bereiken met CPL.

Grammatica

Zinnen die gemaakt zijn met CPL hebben een zelfde structuur als de zinnen die met PENG en ACE worden gemaakt. Een basis zin in CPL heeft de volgende structuur:

Onderwerp + werkwoord + complement + {toevoegsels}

In een CPL zin is een complement een verplicht onderdeel van de zin. Een complement is nodig voor een transitief werkwoord en een distransitief werkwoord. Een toevoegsel is optioneel in een CPL zin. In de structuur van een CPL zin mag gebruik worden gemaakt van verschillende linguïstische structuren, maar voornaamwoorden zijn niet toegestaan in CPL (Clark et al., 2005).

Met CPL kunnen er drie vormen van zinnen worden gemaakt (Clark et al., 2010) (Schwitter, 2010): feittypen, vragende zinnen en regels. Feittypen hebben de volgende structuur:

There is | are [NG]
[NG] werkwoord [NG] [PP]*
[NG] is | are passief werkwoord [door NG] [PP]*
voorbeeld: *"A man picks up a large box from a table"*

Met NG wordt een naamwoord groep of naamwoordelijk zinsdeel bedoelt. Dit is in de syntax een groep van woorden met daarin een zelfstandig naamwoord, bijvoeglijk naamwoord of een voornaamwoord. De werkwoorden in deze zinnen kunnen hulp werkwoorden of een woordgroep zijn die een werkwoord in zich hebben. De zelfstandige naamwoorden in de zinnen kunnen worden veranderd door andere zelfstandige naamwoorden, voorzetsels en bijvoeglijke naamwoorden. Voor vragende zinnen accepteert CPL vijf vormen. De meest voorkomende vormen zijn:

What is [NP]?
Is het waar dat [zin]?

Voor regels accepteert CPL het volgende zins patroon:

ALS zin [EN zin]* DAN zin [EN zin]*
voorbeeld *"IF a person is carrying an entity that is inside a room THEN (almost) always the person is in the room"*

Zinnen kunnen aan elkaar worden verbonden met het woordje *"en"*. De grammatica van CLP ondersteunt enkelvoudige zinnen, voorzetsel groepen, samengestelde zelfstandige naamwoorden, rangtelwoorden, eigenamen, bijvoeglijke naamwoorden, voornaamwoorden, verwijzingen en een beperkt gebruik van conjuncties.

Om zinnen met CPL op te stellen is er een set van richtlijnen beschikbaar. Sommige van deze richtlijnen zijn aanbevelingen om ambiguïteit en misinterpretatie te voorkomen (Clark et al., 2007). Hieronder een paar voorbeeld richtlijnen:

- *Houdt zinnen zo kort en simpel als mogelijk.*
- *Gebruik één clausule per zin.*
- *Gebruik "a" om een object te introduceren en "the" om naar hetzelfde object te refereren.*

- *Gebruik "first" en "second" om twee items van hetzelfde soort van elkaar te scheiden.*
- *Gebruik "There is a ..." wanneer het nodig is om een object te introduceren.*

De volledige lijst met richtlijnen en voorbeelden staat in de handleiding van CPL. CPL beschikt over een aantal grammatica richtlijnen om ambiguïteit en moeilijk te interpreteren structuren te voorkomen. Clark geeft hier het volgende voorbeeld van:

"Assume you are told that the best estimate of the mass of Planet X is 440 kg"

Een valide herformulatie in CPL is voor deze zin de volgende:

"The mass of Planet X is 400 kg"

In de herformulatie is gebruik gemaakt van de richtlijn: korte zinnen en geen imperatieven.

Semantiek

CPL zinnen worden geïnterpreteerd door deze om te zetten naar een formele kennis representatie (KR) taal. De kennis representatie taal waar CPL gebruik van maakt is de Knowledge Machine (KM). Clark beschrijft KM als een volwassen, geavanceerde taal met een goed gedefinieerde semantiek. KM wordt door verschillende projecten gebruikt.
Om de betekenis van een zin vast te stellen moet een gebruiker de CPL zin door de CPL interpretator halen. Op basis van KM bepaalt de CPL interpretator de betekenis van de zin. Nadat dit gedaan is geeft de interpretator de betekenis weer aan de gebruiker. De gebruiker kan de betekenis accepteren of de zin herformuleren.

Voorbeeld requirement in CPL

Om een voorbeeld requirement in CPL te maken is de tool van CPL nodig. De tool reduceert de ambiguïteit en bepaalt de betekenis van de requirement. De tool is niet online beschikbaar waardoor het moeilijk is om een goed voorbeeld te geven. Het volgende voorbeeld is overgenomen van Clark (2007):

"An object is thrown with a horizontal velocity of 20 m/s from a cliff that is 125 m above level ground"

Omgezet naar CPL heeft de tekst de volgende vorm:

"An object is thrown from the top of a cliff.
The initial horizontal velocity of the object is 20 m/s
The initial vertical velocity of the object is 0 m/s
The height of the cliff is 125 m"

In de CPL tekst is de zin omgezet naar vier aparte zinnen. CPL schrijft namelijk voor dat een zin maar mag bestaan uit één clausule. In het voorbeeld is ook te zien dat een object wordt geïntroduceerd door het woordje "a" om te verwijzen naar het object wordt later in de tekst gebruik gemaakt van "*the*".

4. Methode

In dit hoofdstuk wordt beschreven hoe de data verzameld en geanalyseerd is tijdens het onderzoek.

4.1 Onderzoeksmethode

In deze paragraaf staat beschreven welke methoden er gebruikt zijn om de data te verzamelen en te analyseren. De methoden zijn per onderzoeksactiviteit beschreven. In het onderzoek zijn de onderzoeksactiviteiten opeenvolgend uitgevoerd. Bij de onderzoeksactiviteiten zijn de resultaten uit de voorgaande activiteit gebruikt om de methode uit te werken. Bij de enquête is bijvoorbeeld gebruik gemaakt van de resultaten van het requirementsdocumenten onderzoek en de interviews. Door deze structuur worden er in dit hoofdstuk resultaten gebruikt die pas in het volgende hoofdstuk beschreven zijn. Hierdoor kan het lastig zijn om de methoden die beschreven zijn in dit hoofdstuk te begrijpen.

4.1.1 Requirementsdocumenten

Om de kwaliteit van de requirements te verbeteren is het van belang om te weten welke factoren invloed hierop hebben. Met dit doel zijn er een aantal requirementsdocumenten van VitalHealth onderzocht. Het doel was om vast te stellen wat de kwaliteit van de bestaande requirements is. De requirements die onderzocht zijn komen uit requirementsdocumenten van drie verschillende softwareprojecten. De requirements worden bij VitalHealth per project opgesteld door een consultant of applicatieontwikkelaar. Een applicatieontwikkelaar kan namelijk ook de rol van requirements engineer op zich nemen.
VitalHealth software bestaat ruim vijf jaar. In de laatste jaren is de kwaliteit van de requirements engineering fase binnen VitalHealth sterk verbeterd. Bij de selectie van de requirementsdocumenten is hier rekening mee gehouden. Voor het onderzoek zijn alleen requirementsdocumenten geselecteerd die niet meer dan anderhalf jaar oud zijn. Met deze documenten is een betrouwbare meting van de kwaliteit van de bestaande requirements gedaan.
Aanvankelijk was het plan om van een groot, middelgroot en klein softwareproject de requirementsdocumenten te onderzoeken. Door deze spreiding kan het verschil in kwaliteit opgevangen worden die er kan zitten tussen requirements van grote en kleine softwareprojecten. Bij de selectie van de projecten bleek echter dat er bij de kleine softwareprojecten niet of nauwelijks aan requirements engineering wordt gedaan. Hierdoor zijn er twee grotere softwareprojecten gekozen. In het onderzoek worden deze projecten project 1 en project 2 genoemd[7]. Daarnaast is er nog een middelgroot project gekozen; dit is project 3. Per project is er één requirementsdocument onderzocht; in totaal drie documenten. Het eerste document bestond uit 119 requirements. Het tweede document bestond uit 39 requirements en laatste document bestond uit 108 requirements. Bij elkaar zijn er in totaal 266 requirements onderzocht.

Opzet

De requirementsdocumenten zijn onderzocht op kwaliteit op basis van de vijf kwaliteitscriteria die beschreven zijn in hoofdstuk 3. De nadruk bij het requirementsdocumenten onderzoek lag op het criterium eenduidig. De documenten zijn dan ook voornamelijk gecontroleerd op dit criterium. In de requirementsdocument is onderzocht hoeveel requirements er ambigu zijn.
Bij de selectie van de requirementsdocumenten bleek dat binnen VitalHealth de requirements worden opgesteld door stukken tekst waarin meerdere requirements zijn verwerkt. Omdat het voor de analyse van belang is dat er per requirement wordt gecontroleerd wat de kwaliteit is, zijn de stukken tekst

7 Wegens bedrijfsvertrouwelijke informatie zijn de namen van de projecten veranderd.

omgezet naar requirements per regel. Het volgende voorbeeld is gehaald uit één van de requirementsdocumenten. In het voorbeeld is te zien dat requirements zijn verwerkt in de tekst.

> *"***2.1.1 Optometrist**
> *The Optometrist has access to all the features needed to maintain patient medical health records. The core process for the optometrist is the Exam. The Optometrist role has the permissions to sign an Exam."*

Uit dit voorbeeld zijn drie requirements gehaald. Dit heeft het volgende resultaat:

1. *The Optometrist has access to all the features needed to maintain patient medical health records.*
2. *The core process for the optometrist is the Exam.*
3. *The Optometrist role has the permissions to sign an Exam.*

Op deze manier zijn uit de drie documenten de requirements gehaald die gecontroleerd zijn op ambiguïteit.

In hoofdstuk 3 zijn er drie groepen ambiguïteiten beschreven (linguïstische, onbepaaldheid en requirement engineering). In dit onderzoek zijn de requirements voornamelijk onderzocht op de groep linguïstische ambiguïteiten. Software engineerings ambiguïteiten zijn lastig te detecteren zonder domeinkennis. Om software engineerings ambiguïteit te detecteren moeten er andere onderzoeksmethode worden gebruikt dan die gebruikt zijn voor het vinden van linguïstische ambiguïteiten.

In hoofdstuk 3 zijn er verschillende typen linguïstische ambiguïteiten beschreven. In de documenten is onderzocht welk type het meest voorkomt in de requirements. Om ambiguïteiten in de requirements te detecteren en classificeren is er gebruik gemaakt van een ambiguïteiten checklist. Kamsties, Berry en Paech (2001) hebben een checklist ontwikkeld (de checklist is beschreven in hoofdstuk 3) waarmee ambiguïteiten gedetecteerd en geclassificeerd kunnen worden. Met de checklist kan er per requirement worden gecontroleerd of deze ambigu is. De checklist bestaat uit vijf verschillende items. De eerste vier items (polysemie, systematische polysemie, referentiële ambiguïteit, discourse ambiguïteit) zijn linguïstische ambiguïteiten. Het laatste item (domein ambiguïteit) is een software engineerings ambiguïteit. Om de requirements op het laatste criterium te controleren is er domeinkennis nodig. Domeinkennis was niet aanwezig in het onderzoek daarom zijn de requirements niet volledig op dit punt gecontroleerd.
Het detecteren van ambiguïteit in requirements is een lastige klus. Kamsties, Berry en Paech hebben in hun onderzoek een experiment uitgevoerd met de checklist die zij ontwikkeld hebben. De resultaten van dit experiment laten zien dat het moeilijk is om al de ambiguïteiten in een requirementsdocument te vinden. Het resultaat van hun experiment was dat er maar 18% van al de ambiguïteiten in een document werden gevonden. In het experiment werd 4,5 uur gezocht door een team van drie reviewers. Om de betrouwbaarheid van dit deel van het onderzoek te waarborgen zijn de volgende activiteiten uitgevoerd. Door één persoon zijn de requirements die in bijlage "A.1" staan onderzocht op kwaliteit. Hierbij is gebruik gemaakt van de ambiguïteiten checklist uit tabel 4.1. De checklist is omgezet en vertaald naar het Nederlands:

Ambiguïteit type	**Beschrijving**
Polysemie	Heeft een woord in de requirement meer dan één betekenis. Wees ervan bewust dat een lexicale ambiguïteit ontstaat in het bijzonder bij de daadwerkelijke betekenis van een woord in de context van de RE. voorbeelden: een voorbeeld hiervan is het woord groen; dit kan een kleur betekenen maar ook in sommige contexten jeugdig en onervaren.
Systematische polysemie	Bij systematische polysemie is de betekenis van een woord verwarrend tussen klassen, bv. instantie en class ("Ik vind deze jas leuk", dit kan verwijzen naar een individuele jas of naar het type jas), tussen proces en product (Het zelfstandig naamwoord verwijst naar een proces of naar een product van het proces, voorbeelden building, writing) en tussen gedrag en dispositie (bv. "Dit is een snelle auto", kan verwijzen naar het gedrag of naar de klasse van de auto zoals een Ferrari) en de volatile–persistent ambiguïteit komt voor wanneer een woord verwijst naar een blijvende of een tijdelijke eigenschap van een object.

Referentiële ambiguïteit	Kan een woord of een deel van de zin in een requirement verwijzen naar meer dan één object in de requirements of in een andere. Controleer op voornaamwoorden (persoonlijke, bezittelijke, aanwijzende), zelfstandige naamwoorden en *ellips*. *Voorbeelden*: "*Anke heeft maar één zus en die woont al jaren in Noorwegen. Ze zou het wel wat gezelliger vinden om wat dichter bij haar in de buurt te wonen, maar ze kan het prachtige Scandinavische landschap niet missen."* In de tweede zin van het voorbeeld is niet duidelijk naar wie er verwezen wordt, Anke of Haar zus.
Discourse ambiguïteit	Heeft een requirement meer dan 1 interpretatie in relatie tot andere requirements? Deze ambiguïteit komt voor wanneer (1) woorden zoals, eerste, vorige, tussen, na en laatste worden gebruikt die kunnen verwijzen naar verschillende elementen. En wanneer (2) bijvoeglijke naamwoorden, werkwoorden of zelfstandige naamwoord zinnen verwijzen naar meer dan één conditie die beschreven is in het voorgaande.
Domein ambiguïteit	Is de requirement ambigu in relatie tot de kennis van de applicatie, systeem of het ontwikkeld domein.

Tabel 4.1 Ambiguïteiten checklist overgenomen uit (Kamsties et al., 2001) vertaald in het Nederlands

Twee andere personen hebben dezelfde lijst met requirements uit bijlage "A.1" gereviewd met bovenstaande checklist. De resultaten hiervan zijn te vinden in de tabel in bijlage "A.2". In de resultaten tabel is te zien welke ambiguïteiten de respondenten hebben gevonden en hoe zij deze hebben geclassificeerd. Uit deze resultaten blijkt dat de respondenten niet precies dezelfde ambiguïteiten hebben gevonden in de requirements. Wanneer er dezelfde ambiguïteit in een requirement gevonden werd, is deze wel op dezelfde manier geclassificeerd. Hieruit blijkt dat de resultaten van de checklist betrouwbaar zijn bij het classificeren van de ambiguïteiten. Met de checklist worden alleen niet door verschillende personen dezelfde ambiguïteiten gevonden.

Na de reviewsessie is met de respondenten besproken hoe ze de ambiguïteiten hebben gevonden en geclassificeerd. Het doel van de reviewsessie was om te achterhalen waarom een respondent een ambiguïteit in de requirement niet vindt. Met de respondenten zijn ook de ambiguïteiten in de requirements besproken die zij zelf niet gevonden hadden. Na uitleg over de ambiguïteit zagen de respondenten in de meeste gevallen wel dat de requirement ambigu was. De reden waarom ze de ambiguïteit in de eerste plaats niet hadden gevonden is omdat ze er overheen gelezen hadden. Omdat respondenten niet dezelfde ambiguïteiten in requirements vinden is de checklist aangepast zodat bij gebruik meer ambiguïteiten in requirements worden gevonden. Het aanpassen is gedaan op basis van de feedback van de eerste twee respondenten. De feedback van de respondenten was als volgt:

1. *Voor elke type ambiguïteit in één zin beschrijven wat wordt bedoeld met het type.*
2. *De voorbeelden in de checklist consistenter weergeven en meer voorbeelden toevoegen.*
3. *Extra ambiguïteit; dit is: "De betekenis is verwarrend doordat enkelvoud en meervoud door elkaar worden gehaald".*

Op basis van deze feedback is de checklist aangepast. In tabel 4.2 staat de aangepaste checklist.

Type	**Beschrijving**
Polysemie	*Een woord kan meer dan één betekenis hebben.* Bij polysemie heeft een woord meer dan één betekenis in de requirement. Let hierbij op dat er lexicale ambiguïteit kan ontstaan in het bijzonder bij de betekenis van een woord in de context van het RE domein. Voorbeeld: Het woord *groen;* dit kan een kleur betekenen maar ook in sommige contexten jeugdig en onervaren.
Systematische polysemie	Systematische polysemie komt voor op de volgende manieren: De betekenis van een woord is verwarrend tussen klassen en instantie. Voorbeeld: "*Ik vind deze jas leuk"*. Dit kan verwijzen naar een individuele kas of naar het type jas. De betekenis van een woord is verwarrend tussen proces en product. Voorbeeld: "*building*" of *"writing"*. Het zelfstandig naamwoord verwijst naar een proces of naar een product. De betekenis van een woord is verwarrend tussen gedrag en dispositie. Voorbeeld: *"Dit is een snelle auto"*, kan verwijzen naar het gedrag of naar de klasse van de auto zoals een Ferrari.

	De betekenis van een woord is verwarrend tussen een blijvende of een tijdelijke eigenschap van een object. De betekenis is verwarrend doordat enkelvoud en meervoud door elkaar worden gehaald.
Referentiële ambiguïteit	De verwijzing naar een woord of naar een deel van de zin is verwarrend. Een woord of een deel van de zin in een requirement verwijst naar meer dan één object in de requirement. Controleer op voornaamwoorden (persoonlijke, bezittelijke, aanwijzende), zelfstandige naamwoorden en ellips. Voorbeeld: "*Anke heeft maar één zus en die woont al jaren in Noorwegen. Ze zou het wel wat gezelliger vinden om wat dichter bij haar in de buurt te wonen, maar ze kan het prachtige Scandinavische landschap niet missen."* In de tweede zin van het voorbeeld is niet duidelijk naar wie er verwezen wordt, Anke of haar zus.
Discourse ambiguïteit	Meer dan één interpretatie op basis van de verwijzing naar voorgaande requirements. Deze ambiguïteit komt voor wanneer: - Woorden zoals, eerste, vorige, tussen, na en laatste worden gebruikt die kunnen verwijzen naar verschillende elementen. - bijvoeglijke naamwoorden, werkwoorden of zelfstandige naamwoord zinnen verwijzen naar meer dan één conditie die beschreven is in een voorgaande requirement.
Domein ambiguïteit	Is de requirement ambigu in relatie tot de kennis van de applicatie, systeem of het ontwikkeld domein. Voorbeeld: "*De deuren van een lift openen niet bij een verdieping, tenzij de lift stilstaat op die verdieping*". In deze requirement kan de actie openen worden uitgevoerd door de hardware of software van de lift. Wanneer dit door de hardware gedaan wordt is het niet meer nodig dat dit in de software wordt geïmplementeerd.

Tabel 4.2 Aangepaste ambiguïteiten checklist.

Aan een derde respondent zijn de requirements uit bijlage "A.1" voorgelegd om met de vernieuwde checklist ambiguïteiten mee te vinden. In bijlage "A.2" is het resultaat te zien. Uit deze resultaten blijkt dat de respondent nog niet dezelfde ambiguïteiten vindt in de requirements. De respondent heeft de ambiguïteiten die hetzelfde waren wel gelijk geclassificeerd als de andere twee respondenten. Uit deze resultaten blijkt dat het erg lastig is om via een checklist al de ambiguïteiten in een requirementsdocument te vinden.

Na het afronden van het onderzoek naar de checklist is door één persoon de rest van de requirements in de documenten geanalyseerd. Dit is gedaan met de aangepaste checklist. Nadat al de ambiguïteiten gevonden en geclassificeerd waren is er nog een extra controle gedaan op de gevonden ambiguïteiten. Bij de gevonden ambiguïteiten is de theorie uit hoofdstuk drie naast de classificatie van de ambiguïteit gelegd. Hieruit bleek dat veel syntactische ambiguïteiten geclassificeerd werden als referentiële ambiguïteit. De oorzaak hiervoor is dat het type syntactische ambiguïteit niet in de checklist opgenomen was. Omdat de syntactische en referentiële ambiguïteiten op elkaar lijken is deze fout in de eerste fase van het onderzoek over het hoofd gezien. Op basis van deze bevinding is syntactische ambiguïteit als type toegevoegd aan de checklist. De uiteindelijke checklist is te vinden in het hoofdstuk resultaten.

Analyse

Het percentage requirements die ambigu zijn in een document, is in dit onderzoek gebruikt om de kwaliteit van een document te bepalen. Het vaststellen van het percentage is gedaan door al de requirements die ambigu zijn (een requirement is ambigu wanneer er één of meerdere ambiguïteiten in voorkomen) per document bij elkaar op te tellen. Daarna is berekend welk percentage van de requirements ambigu is. Dit is het aantal requirements die ambigu zijn ten opzichte van het totaal aantal requirements in het document.

Voor de selectie van de CNL talen is het van belang om te weten welke type ambiguïteit het meeste voorkomt in de requirementsdocumenten. Per linguïstische ambiguïteit is vastgesteld van welk type de ambiguïteit is. Het vaststellen van het type ambiguïteit kon gedaan worden op basis van de classificatie die er per ambiguïteit gedaan is met de checklist. De eerste twee items (polysemie en systematische polysemie) in de checklist zijn van het type lexicale ambiguïteit. Het derde item (referentiële ambiguïteit) zit tussen de typen semantische en pragmatische ambiguïteit in. Bij een ambiguïteit waarbij de referentie in een zin niet duidelijk is hoort bij het type semantische ambiguïteit. Bij een ambiguïteit waarbij de referentie naar een deel in een andere zin niet duidelijk hoort bij een

pragmatische ambiguïteit. Het vierde item (discourse ambiguïteit) is een pragmatische ambiguïteit van het type referentiële ambiguïteit. De laatste twee items (syntactische en taalfout ambiguïteit) in de checklist horen bij het linguïstische ambiguïteitstype met dezelfde naam. Op basis van deze indeling zijn de linguïstische ambiguïteiten geanalyseerd.

Tijdens het requirementsdocumenten onderzoek is er ook gekeken naar andere opvallende zaken in de documenten die met de andere vier criteria te maken hebben. Deze opvallende zaken zijn beschreven in het hoofdstuk resultaten.

4.1.2 *Interviews*

Na de afronding van het requirementsdocumenten onderzoek zijn er diverse interviews afgenomen. De interviews zijn afgenomen met als doel om twee vragen te beantwoorden. De eerste vraag is: welke factoren beïnvloeden de kwaliteit van de requirements. Deze vraag is deels te beantwoorden uit de resultaten van het requirementsdocumenten onderzoek. In het interview zijn op basis van deze resultaten vragen gesteld om achterliggende oorzaken te achterhalen. De tweede vraag is: waar moet een gecontroleerde taal aan voldoen om gebruiksvriendelijk te zijn. Het antwoord hierop is van belang bij de selectie van de gecontroleerde taal.

Het interview is afgenomen bij vijf personen met verschillende rollen. De eerste twee interviews zijn afgenomen bij personen die requirements opstellen. Het derde en vierde interview is afgenomen bij personen die softwareapplicaties ontwikkelen. Het vijfde interview is afgenomen bij een persoon die de softwareapplicatie test. De vijf respondenten zijn geselecteerd op basis van hun werkervaring. De respondenten werken minimaal een jaar bij VitalHealth.
Bij de respondenten is een semi-gestructureerd interview afgenomen. Bij een semi-gestructureerd interview worden de vragen voorbereid, maar tijdens het gesprek wordt de volgorde bepaald waarop de vragen worden gesteld. De vragen van de interviews zijn voorbereid; deze zijn na te lezen in bijlage B interviews. Het interview is afgenomen volgens de STARR methode. STARR (Situatie, Taak, Actie, Resultaat en Reflectie) is een gedragsgerichte interviewmethode die gebruikt wordt om te achterhalen hoe respondenten hebben gehandeld in situatie waarbij zij betrokken zijn geweest. De STARR methode is gekozen omdat er met een gedragsgerichte methode achterhaald kan worden waarom respondenten in het verleden bepaalde keuzes gemaakt hebben. Daarnaast geven situaties uit de praktijk respondenten een houvast om vragen op een concrete manier te beantwoorden.

<u>Interview opzet</u>
De vragen van het interview zijn gemaakt op basis van drie bronnen: requirementsdocumenten onderzoek, de vijf kwaliteitscriteria en de Wyner eigenschappenlijst. Bij de interviews is deels Grounded theory methode toegepast. Sommige onderwerpen die de respondenten noemen zijn nieuw of er zijn nieuwe inzichten op de onderwerpen vanuit de andere onderzoeksactiviteiten. Deze nieuwe onderwerpen zijn getoetst bij een volgend interview. In het vervolg van deze paragraaf is aangegeven welke vragen of coderingen later zijn toegevoegd.

Requirements engineerings proces
VitalHealth heeft verschillende software productlijnen. Elke productlijn heeft zijn eigen projecten en werknemers. Per productlijn kan het verschillend zijn hoe de requirements worden verzameld en opgesteld. Hierdoor kunnen er per productlijn verschillende factoren zijn die de kwaliteit van de requirements beïnvloeden. Om in kaart te brengen of bepaalde factoren optreden bij één of meerdere productlijnen is er aan de respondenten gevraagd in welke projecten zij werkzaam zijn geweest in het afgelopen jaar. In hoofdstuk 3 zijn er een aantal rollen beschreven die gebruik maken van requirements. In het interview is aan de respondent gevraagd welke rol hij in het project vervult heeft. Door deze vraag kan er bij de analyse onderscheid worden gemaakt in de specifieke behoefte die een rol heeft voor requirements en gecontroleerde talen.
Nadat de respondent kort iets over de projecten heeft verteld is hem gevraagd welke taken hij uitvoert binnen het project. Hierbij is de respondent gevraagd op welke manier hij te maken heeft met requirements in zijn dagelijkse werkzaamheden. De respondent is gevraagd welke activiteiten hij met betrekking tot de requirements in de projecten heeft uitgevoerd. Gebaseerd op voorgaande interviews is aan de laatste twee respondenten gevraagd of de requirements die zij gebruiken speciaal voor

stakeholders of voor ontwikkelaars zijn opgesteld. In de eerste twee interviews gaven requirements engineers aan dat zij het lastig vinden om hier een goed evenwicht in te vinden.

In hoofdstuk drie zijn een vijf kwaliteitscriteria voor requirementsdocumenten beschreven. Aan de hand van deze criteria zijn er vragen opgesteld voor de respondenten. Aan de respondenten is gevraagd waarom bepaalde keuzes worden gemaakt bij het opstellen van de requirements en of dit hun voorkeur heeft.

Implementatie onafhankelijk
De requirements in de requirementsdocumenten worden procedureel beschreven. De requirements beschrijven hoe de softwareapplicatie moet gaan werken. In plaats van procedureel kunnen requirements ook declaratief worden opgesteld. Bij requirements die declaratief zijn opgesteld wordt beschreven wat het systeem moet oplossen. In de interviews is aan de respondenten gevraagd welke vorm bij hun de voorkeur heeft. Aan de respondenten is ook gevraagd waarom dit bij hun de voorkeur heeft en wat zij hiervan de voor- of nadelen vinden. De vragen zijn gesteld om te achterhalen of de requirements engineers er bewust voor kiezen om de requirements implementatie afhankelijk op te stellen. Bij het onderzoek naar de requirementsdocumenten kwam naar voren dat er vaak geen onderscheid gemaakt wordt tussen de requirements en het functioneel ontwerp. Aan de respondenten is gevraagd of deze conclusie klopt en wat de redenen hiervoor zijn.

Traceerbaar
In de requirementsdocumenten zijn de requirements opgesteld in een beschrijvend stukje tekst. In één stuk tekst worden er meerdere requirements opgenomen. Aan de requirements engineers is gevraagd of zij deze manier van noteren makkelijker te begrijpen vinden voor de ontwikkelaar dan requirements die opgesteld zijn in een lijst. Deze vraag is gesteld om te achterhalen of de requirements engineers er bewust voor kiezen om de requirements op te stellen in stukken tekst. Aan de respondenten die ontwikkelaar of tester zijn, is bij deze vraag gevraagd welke manier hun voorkeur heeft.

Eenduidig
Bij de requirementsdocumenten die onderzocht zijn is geen verklarende woordenlijst opgenomen. In één van de document was alleen een lijst opgenomen met verklaringen van gebruikte afkortingen. Gebaseerd op deze constatering zijn de respondenten hierover bevraagd. Door de betekenis van domeinwoorden niet te beschrijven kan het voorkomen dat stakeholders of applicatieontwikkelaars bepaalde woorden in de requirements niet begrijpen. Aan de requirements engineer is gevraagd of dit wel eens voorkomt. Gerelateerd aan dit onderwerp is aan de requirements engineers gevraagd hoe zij ervoor zorgen dat domeinkennis, die nodig is om bepaalde woorden te begrijpen, overdragen op de applicatieontwikkelaars. Aan applicatieontwikkelaars is gevraagd of het wel eens voorkomt dat zij bepaalde woorden in de requirements niet begrijpen.

In het requirementsdocumenten onderzoek zijn veel ambiguïteiten in de requirements gevonden. In het interview is aan de respondenten gevraagd hoe vaak het voorkomt dat requirements op een verkeerde manier worden geïnterpreteerd. Daarbij is ook gevraagd of zij zelf voorbeelden kunnen geven van situatie waarin dit voorkwam. Het doel van deze vragen is om te achterhalen of de requirements engineers zich bewust zijn van ambiguïteiten in requirements en wat de invloed daarvan is op de softwareontwikkeling.

Verifieerbaar
Aan de respondenten is gevraagd hoe er gecontroleerd wordt of de requirements in de softwareapplicatie zijn geïmplementeerd.

Requirements kwaliteit verbeteren
Aan de respondenten zijn een aantal kaartjes gegeven met daarop de kwaliteitscriteria. Per criterium is aan de respondenten gevraagd of de huidige requirements hieraan voldoen. Om te achterhalen welke criteria de respondenten zelf belangrijk vinden is aan de respondenten gevraagd om de kaartjes op volgorde te leggen van welke zij erg belangrijk vinden tot welke zij minder belangrijk vinden.

Eigenschappen die belangrijk zijn voor de gebruiksvriendelijkheid van een gecontroleerde taal.
Aan de respondenten is een korte uitleg gegeven over gecontroleerde talen. Daarna is aan de respondent gevraagd hoeveel tijd hij wil investeren in het leren van een CNL taal die voldoet aan de kwaliteitseigenschappen van requirements die hij in de vorige vraag als belangrijk heeft opgegeven.

Uit de eigenschappenlijst van Wyner zijn een aantal eigenschappen gehaald en voorgelegd aan de respondenten. Dit zijn algemene en linguïstische eigenschappen van een CNL taal. De eigenschappen zijn aan de respondenten voorgelegd en er is gevraagd welke eigenschappen zij belangrijk vinden voor de gebruiksvriendelijkheid van een gecontroleerde taal. Een aantal van deze eigenschappen gaan over linguïstische eigenschappen voor requirements. Bijvoorbeeld het gebruik van beeldspaak, aan de respondenten is gevraagd of zij hier gebruik van maken bij het maken van de huidige requirements en of zij deze eigenschap belangrijk vinden om te blijven gebruiken wanneer zij met een gecontroleerde taal requirements gaan opstellen. Naast de linguïstische eigenschappen is de respondenten ook een aantal andere eigenschappen van een gecontroleerde taal voorgelegd, met hierbij de vraag hoe belangrijk zij deze vinden.

Als afsluiting van het interview is de respondenten nog een aantal vragen voorgelegd om te achterhalen hoeveel kennis de respondenten hebben over formele en gestructureerde talen.

Interview analyse
De interviews zijn digitaal opgenomen en volledig uitgeschreven. De interviews zijn terug te vinden in bijlagen *"B. Interviews"*. Bij het uitschrijven van de interviews zijn sommige zinnen iets aangepast om de leesbaarheid te verbeteren. Woorden zoals "uhm" en stiltes zijn niet opgenomen in de transcriptie. De uitgeschreven interviews zijn van codes voorzien. In de transcripties zijn de codes in de kantlijn gezet. De zinsdelen die bij de code horen zijn onderstreept. Een code bestaat uit een variabele die een bepaalde waarde kan hebben. De code kan gezien worden als het onderwerp waarover gesproken wordt in het interview. De waarde is de mening, activiteit of dergelijk van de respondent over het onderwerp. In deze paragraaf staan de codes beschreven die gebruikt zijn. De codes zijn dik gedrukt weergegeven. De waarden die bij de codes horen staan onder de code. In de transcripties worden de codes in de kantlijn op de volgende manier weergegeven: "**<code>**.*<waarde>*".

VitalHealth heeft verschillende software productlijnen. Elke productlijn heeft zijn eigen projecten en werknemers. Per productlijn kan het verschillend zijn hoe de requirements worden verzameld en opgesteld. Hierdoor kunnen er per productlijn verschillende factoren zijn die de kwaliteit van de requirements beïnvloeden.

Code:	Project
Waarde:	Bij VitalHealth zijn er vijf productlijnen. Dit zijn: *Keten Informatie Systeem (KIS)*, *patiëntenportalen (PP)*, *QuestManager (QM)*, *Electronic Health Record (EHR)* en *maatwerkprojecten (MP)*.

In hoofdstuk 3 zijn een aantal rollen beschreven die gebruik maken van requirements. Een doel van dit onderzoek is om te definiëren welke eigenschappen voor de verschillende rollen belangrijk zijn voor de requirements en de gecontroleerde taal.

Code:	Rol
Waarde:	*Stakeholder*, *Requirements Engineer (RE)*, *Applicatieontwikkelaar* en *Applicatietester*.

In de interviews hebben de respondenten verschillende manieren genoemd waarop requirements verzameld worden. Om een stukje achtergrond bij de requirements te hebben is in de interviews gekeken hoe de wensen en eisen worden verzameld. Deze code is later toegevoegd gebaseerd op wat respondenten in de interviews aangeven.

Code:	Verzamelen
Waarde:	*mondeling stakeholder, mailcontact.*

De opgestelde requirements kunnen op verschillende manieren bewaard worden. De requirements kunnen worden opgenomen in een document, spreadsheet, requirementstool of een database. Deze code is later toegevoegd gebaseerd op wat respondenten in de interviews aangeven.

Code: Aanlevering
Waarde: *document, mail, mondeling via requirements engineer, supportsysteem.* (de waarde zijn opgesteld op basis inhoud van de interviews)

De requirements worden door verschillende rollen gebruikt. Elke rol heeft een andere achtergrond. Hierdoor zijn requirements die door stakeholders te begrijpen zijn niet direct te begrijpen voor de applicatieontwikkelaars. Uit de eerste twee interviews bleek dat de respondenten het moeilijk vinden om de requirements op te stellen zodat rol deze begrijpt. De respondenten hebben aangeven dat zij het lastig vinden om een goed evenwicht hierin aan te brengen. Zien andere respondenten dit ook als een moeilijkheid.

Code: Doel rol
Waarde: De requirements kunnen voor *beide* rollen, voor de *stakeholder* of *ontwikkelaar* specifiek of voor *geen* één rol worden opgesteld.

Bij het opstellen van de requirements kunnen er verschillende notatietechnieken worden gebruikt. Elke techniek heeft zijn eigen notatie om requirements mee te beschrijven. Bij gebrek aan kennis van de notatietechniek die gebruikt wordt is het lastig om een requirement te begrijpen. In hoofdstuk drie wordt meer beschreven over de problemen die hierbij kunnen optreden. Aan de respondenten is gevraagd of zij gebruik maken van een notatietechniek.

Code: Notatietechniek
Waarde: De respondent geeft aan dat er *niet* of *wel* gebruik van een notatietechniek wordt gemaakt.

In hoofdstuk 3 zijn een aantal kwaliteitscriteria voor requirementsdocumenten beschreven. Per eigenschap zijn er aan de respondent vragen gesteld in de interviews. Per eigenschap zijn er ook een aantal codes opgesteld.

Eenduidig
In het interview zijn respondenten over eenduidigheid bevraagd.

Code: Eenduidig
Waarde: De respondent vindt de huidige requirements *wel* of *niet* eenduidig. De respondent vindt deze eigenschap *belangrijk* of *onbelangrijk*.

In de IEEE Standaard 830 (1984) wordt geadviseerd om een verklarende woordenlijst op te nemen in een requirementsdocument. In de interviews is gevraagd waarom woordenlijsten niet altijd worden opgenomen in de documenten.

Code: Woordenlijst
Waarde: De respondent geeft aan of hij *wel*, *niet* of *soms* verklarende woordenlijsten tegenkomt in de requirementsdocumenten.

Ambiguïteit in requirements kan potentieel gevaarlijk zijn omdat het kan leiden tot slechte requirements. Aan de respondenten is gevraagd of zij in de praktijk weleens meemaken dat er woord op een andere manier wordt geïnterpreteerd. Er is onderscheid gemaakt tussen stakeholder en applicatieontwikkelaars om duidelijk te krijgen welke groep hier het meest mee te maken heeft.

Code: Lexicale Ambiguïteit stakeholder *(Lex. stakeholder)*
Waarde: De respondent geeft aan dat dit *wel, niet* of *soms* voorkomt.

Code: Lexicale Ambiguïteit ontwikkelaar *(Lex. ontwikkelaar)*
Waarde: De respondent geeft aan dat dit *wel, niet* of *soms* voorkomt.

De stakeholder kan de requirement op een andere manier interpreteren dan de requirements engineer of de applicatieontwikkelaar. In het interview is de respondenten gevraagd of ze in de praktijk meemaken dat requirements verkeerd worden geïnterpreteerd.

Code: Interpretatie
Waarde: De respondent geeft aan dat dit *wel, niet* of *soms* voorkomt.

Voor het anders interpreteren van requirements kunnen er diverse redenen zijn. Benoemt de respondent in het interview een reden waarom er in de praktijk requirements anders worden geïnterpreteerd. De waarde zijn opgesteld op basis van wat er in de interviews genoemd wordt.

Code: Reden Interpretatie
Waarde: *domeinkennis, zinsstructuur, incompleet, geen inzicht* en *slecht lezen.* (de waarde zijn opgesteld op basis inhoud van de interviews)

Ambiguïteit heeft invloed op de kwaliteit van de requirements. De kwaliteit van de requirements heeft invloed op de kwaliteit van de softwareapplicatie. Aan de respondenten is in het interview gevraagd of zij in de praktijk voorbeelden zijn waarbij ambiguïteit invloed op de kwaliteit van de softwareapplicatie had.

Code: Invloed Ambiguïteit
Waarde: De respondent geeft aan dat ambiguïteit *wel, niet* of *soms* invloed heeft.

Omdat ambiguïteit in de requirements invloed heeft op de kwaliteit van de softwareapplicatie is het belangrijk om te zorgen dat dit zo weinig mogelijk voorkomt in de requirements. Worden er bij het opstellen van de requirements activiteiten uitgevoerd om ambiguïteit te voorkomen.

Code: Ambiguïteit Voorkomen
Waarde: De respondent geeft aan dat dit *wel, niet* of *soms* wordt uitgevoerd.

Om een requirement op een juiste manier te interpreteren heeft een lezer kennis van het applicatie domein nodig. Hoe wordt aan de applicatieontwikkelaar domeinkennis overgedragen.

Code: Domeinkennis
Waarde: De respondent noemt *hoe* domeinkennis wordt overgedragen.

Implementatie onafhankelijk
De requirementsdocumenten worden bij VitalHealth niet implementatie onafhankelijk opgesteld. In de interviews zijn de respondenten hierover bevraagd.

Code: Implementatie onafhankelijk *(implementatie)*
Waarde: De respondent vindt de huidige requirements *wel* of *niet* implementatie onafhankelijk. De respondent vindt deze eigenschap *belangrijk* of *onbelangrijk*.

De requirements zijn vaak procedureel beschreven; de requirements beschrijven hoe de softwareapplicatie moet gaan werken. In het interview is aan de respondenten gevraagd of dit hun voorkeur heeft.

Code: Procedureel
Waarde: De respondent geeft aan dat het procedureel beschrijven zijn duidelijke *voorkeur* of *afkeur* heeft. De respondent is *positief* of een *negatief* over een eigenschap van procedureel beschrijven.

In plaats van procedureel kunnen requirement ook declaratief worden opgesteld. In dat geval wordt in de requirements beschreven wat het systeem moet oplossen.

Code: Declaratief
Waarde: De respondent geeft aan dat het declaratief beschrijven zijn duidelijke *voorkeur* of *afkeur* heeft. De respondent is *positief* of *negatief* over een eigenschap van het declaratief beschrijven.

Bij het onderzoek naar requirementsdocumenten kwam naar voren dat er vaak geen onderscheid gemaakt wordt tussen de requirements en het functioneel ontwerp. Aan de respondenten is gevraagd of deze conclusie klopt.

Code: Functioneel
Waarde: De respondent geeft aan dat er *niet* of *wel* onderscheid wordt gemaakt.

Traceerbaar
In de interviews zijn de respondenten over de traceerbaarheid van de requirements bevraagd.

Code: Traceerbaar
Waarde: De respondent vindt de huidige requirements *wel* of *niet* traceerbaar. De respondent vindt deze eigenschap *belangrijk* of *onbelangrijk*.

In de requirementsdocumenten werden de requirements opgesteld in een beschrijvend stukje tekst. In één stuk tekst werden meerdere requirements opgenomen. In het interview is aan de respondenten gevraagd of deze notatie hun voorkeur heeft.

Code: Tekstueel
Waarde: De respondent geeft aan dat het tekstueel opstellen van requirements zijn duidelijke *voorkeur* of *afkeur* heeft.

In plaats van het opnemen van requirements in een stuk tekst kunnen requirements ook worden opgenomen in een lijst. Elke requirement is een apart item in de lijst. In het interview is aan de respondenten gevraagd of deze notatie hun voorkeur heeft.

Code: Lijst
Waarde: De respondent geeft aan dat het opstellen van requirements in een lijst zijn duidelijke *voorkeur* of *afkeur* heeft.

Consistent
In de interviews zijn de respondenten over de consistentie van de requirements bevraagd.

Code: Consistent
Waarde: De respondent vindt de huidige requirements *wel* of *niet* consistent. De respondent vindt deze eigenschap *belangrijk* of *onbelangrijk*.

Verifieerbaar
In de interviews zijn de respondenten over de verifieerbaarheid van de requirements bevraagd.

Code: Verifieerbaar
Waarde: De respondent vindt de huidige requirements *wel* of *niet* verifieerbaar. De respondent vindt deze eigenschap *belangrijk* of *onbelangrijk*.

Traceerbaar
In de interviews zijn de respondenten over de traceerbaarheid van de requirements bevraagd.

Code: Traceerbaar
Waarde: De respondent vindt de huidige requirements *wel* of *niet* traceerbaar. De respondent vindt deze eigenschap *belangrijk* of *onbelangrijk*.

Gecontroleerde taal leren
Om een gecontroleerde taal goed te gebruiken moet de gebruiker tijd investeren om deze te leren. In de interviews is aan de respondenten gevraagd hoeveel tijd ze willen investeren om een CNL te leren gebruiken.

Code: tijd
Waarde: De respondent wil *wel* of *niet* tijd investeren om met de CNL te leren werken.

De tijd die geïnvesteerd moet worden om een CNL te leren is afhankelijk van de ervaring van een persoon met formele talen of gecontroleerde talen. In de interviews is aan de respondenten gevraagd hoeveel ervaring zij hebben met formele en gecontroleerde talen.

Code: Formeletaal *(formeel)*
Waarde: De respondent heeft *wel* of *geen* ervaring.

Code: Gestructureerde taal *(gestructureerd)*
Waarde: De respondent heeft *wel* of *geen* ervaring.

Eigenschappen voor een gecontroleerde taal
Uit de lijst van Wyner zijn een aantal eigenschappen voorgelegd aan de respondenten. De eigenschappen zijn onderverdeeld in linguïstische en algemene taal eigenschappen. In de interviews worden de volgende linguïstische eigenschappen genoemd: beeldspraak, gezegde, spreekwoord, afwisseling in structuur van zinnen, tussenzinnen en opsommingen of verwijzingen. Deze eigenschappen zijn ook als code gebruikt bij de analyse. Met de volgende waarde; de respondent geeft per eigenschap aan of deze *wel* of *niet* gebruikt wordt in de requirements. De respondent geeft aan dat de *eigenschap* zijn *voorkeur* of *afkeur* heeft.

In de interviews worden de volgende algemene eigenschappen genoemd: tool ondersteuning, meertalig, eenvoudig te leren en beschikbaarheid van een eenvoudige handleiding. De respondent geeft aan dat de *eigenschap* zijn *voorkeur* heeft.

4.1.3 Enquête
Om een grotere groep met respondenten te bereiken zijn er naast de interviews ook enquêtes afgenomen. De vragen voor de enquête zijn opgesteld op basis van de eerste drie interviews. Met de enquête zijn de respondenten over dezelfde onderwerpen bevraagt als in de interviews.

De vragenlijst is verstuurd naar personen die de rol van requirements engineer of applicatieontwikkelaar hebben binnen een VitalHealth project. Voor de vragenlijst zijn er 35 personen uitgenodigd. De personen zijn requirements engineer, applicatieontwikkelaar of zijn direct betrokken bij één van de VitalHealth softwareprojecten. De enquête is ingevuld door achttien personen. Dit is ongeveer de helft van de personen die een uitnodiging hebben ontvangen. De enquete is online afgenomen met behulp van de formulierentool van Google drive. De 35 personen zijn uitgenodigd met een email. In de email stond een link waarmee de respondenten direct toegang hadden tot het online enquêteformulier.

Opzet enquête
De vragen die gesteld zijn aan de respondenten zijn na te lezen in bijlage *"C. enquête"*. De vragen zijn gemaakt op basis van vier input bronnen: het requirementsdocumenten onderzoek, de eerste drie gehouden interviews, de vijf kwaliteitscriteria voor requirements en de Wyner eigenschappenlijst voor gecontroleerde talen.

VitalHealth heeft verschillende software productlijnen In de enquête is aan de respondenten gevraagd om in te vullen in welke projecten zij in het afgelopen jaar werkzaam zijn geweest. Bij de analyse van de enquête is hierdoor na te gaan of bepaalde factoren VitalHealth breed of productlijn specifiek zijn. Zoals beschreven in hoofdstuk 3 hebben personen in een softwareproject verschillende rollen. De respondenten hebben ook verschillende rollen. In de enquête kunnen de respondenten opgeven welke rol zij hebben. De respondenten kunnen kiezen uit de rollen die in hoofdstuk 3 zijn genoemd. Uit de interviews is gebleken dat sommige personen meer dan één rol hebben in het project. De rol van requirements engineer applicatieontwikkelaar wordt soms gecombineerd. De respondent werd in de enquête de mogelijkheid geboden om meer dan één rol te kiezen.

In hoofdstuk 3 is beschreven dat de requirements engineers de requirements opstellen. In de vierde vraag van de enquête is de respondent gevraagd of hij wel eens requirements opstelt. Met deze vraag wordt onderscheid aangebracht tussen de requirements engineers en de applicatieontwikkelaars. De requirements engineers en de applicatieontwikkelaars krijgen een apart set aan vragen voorgelegd. Aan de requirements engineers is naar de ervaring gevraagd die zij hebben met requirements in combinatie met de stakeholder en de applicatieontwikkelaar. Aan de applicatieontwikkelaar is

gevraagd naar de ervaring die zij hebben met requirements in combinatie met de requirements engineer.

Requirements opstellen
De requirements die de requirements engineer opstelt kunnen gebruikt worden door verschillende rollen. De mate waarmee de een rol gebruik maakt van de requirements kan verschillen. Aan de requirements engineers is gevraagd in welke mate de stakeholder, collega requirements engineers en applicatieontwikkelaars gebruik maken van hun requirements.
De requirements worden gebruikt door verschillende rollen. De personen in een rol hebben een andere achtergrond dan personen in een andere rol. De stakeholders hebben meer domeinkennis van de softwareapplicatie. De applicatieontwikkelaars hebben meer technische kennis van de softwareapplicatie. Voor de requirements engineer kan het lastig zijn om de software requirements op te stellen zodat deze goed te begrijpen zijn door al de rollen. In de interviews gaven requirements engineers aan dat zij het lastig vinden om een goede balans tussen de twee rollen te hebben. Aan de requirements engineers is in de enquete gevraagd hoe bij hun de verhouding ligt. De respondenten hebben dit aangeven op een schaal van één tot zeven. Bij nummer één houden zij alleen rekening met de stakeholders bij het opstellen van de requirements. Bij nummer zeven houden zij alleen rekening met de applicatieontwikkelaars bij het opstellen van de requirements.

Requirements en communicatie
Door ambiguïteit in een requirements kan het voor een stakeholder of applicatieontwikkelaar moeilijk zijn om een requirement te begrijpen. In de enquête is gevraagd hoe er in de praktijk wordt gecommuniceerd over de requirements. Zijn communicatieproblemen er de oorzaak van dat requirements verkeerd worden begrepen door de stakeholder of de applicatieontwikkelaar. Aan de respondenten die requirements engineer zijn, is gevraagd hoe vaak zij op een bepaalde manier met de stakeholder en de applicatieontwikkelaar communiceren. De volgende manieren van communicatie zijn voorgelegd aan de respondenten: mondeling contact, mail contact en communicatie via het VitalHealth Support systeem. De keuze voor deze drie manieren is gebaseerd op de interviews. In de interviews hebben de respondenten verschillende manieren genoemd om te communiceren. In de enquête was het voor de respondenten mogelijk om ook nog andere manieren op te geven waarop zij communiceren. Aan de applicatieontwikkelaars is gevraagd hoe vaak zij op de genoemde manieren communiceren met de requirements engineer. Het doel achter deze vraag is om te achterhalen of de mate van communicatie te maken heeft met de kwaliteit van de requirements. Is het zo dat er voor requirements van slechte kwaliteit meer communicatie nodig is. In de vragen over communicatie is gebruik gemaakt van een 5-puntsschaal: nooit, 1x per maand, 1x per twee weken, 1x per week en meerdere malen per week.

Gebruik van requirements
Om goed over de requirements te communiceren is het belangrijk dat de verschillende rollen in het softwareproject in het bezit zijn van de requirements. Wanneer de een persoon niet in het bezit is van de laatste versie van de requirements kan dit veel miscommunicatie opleveren. Aan de requirements engineers is in de enquête gevraagd of de stakeholder en de applicatieontwikkelaars in het project in het bezit zijn van de laatste versie van de requirements. Wanneer blijkt dat personen in het softwareproject vaak niet in het bezit zijn van de requirements kan dit een oorzaak zijn waarom requirements slecht begrepen worden. Aan de applicatieontwikkelaars is dezelfde vraag gesteld. Bij deze vragen gebruik gemaakt van een 4-puntsschaal: nooit, soms, vaak en altijd.

In de interviews gaf een respondent aan dat de requirements vaak niet of slecht worden gelezen. Om de requirements te begrijpen is het nodig om deze door te lezen. Om te achterhalen of dit vaak voorkomt is in de enquête aan de respondenten gevraagd of het voorkomt dat de stakeholders, applicatieontwikkelaars of zijzelf de requirement die zij aangeleverd krijgen niet doorlezen. Deze vraag is te beantwoorden met ja of nee. Om te achterhalen hoe nauwkeurig de requirements worden doorgelezen is aan de respondenten gevraagd om op een schaal van één tot zeven aan te geven hoe nauwkeurig de requirements worden gelezen. Bij één worden de requirements zeer onnauwkeurig gelezen en bij zeven worden de requirements zeer nauwkeurig gelezen. Aan de respondenten is

gevraagd om het antwoord op deze vraag toe te lichten. Hiermee kan achterhaald worden wat de reden is waarom de requirements op een bepaald niveau gelezen worden.

Ambiguïteit in requirements
Lexicale ambiguïteit treedt op wanneer een woord meerdere betekenissen heeft. In de enquête is aan de requirements engineers gevraagd hoe vaak zij in de praktijk vragen krijgen van een stakeholder of applicatieontwikkelaar die een woord niet begrijpt. Aan de applicatieontwikkelaars is gevraagd hoe vaak zij woorden tegenkomen die zij niet begrijpen. Een requirement die eenduidig is kan maar op één manier worden uitgelegd. In de enquête is aan de requirements engineers gevraagd hoe vaak zij in de praktijk vragen krijgen van een stakeholder of applicatieontwikkelaar die een requirement verkeerd interpreteert. Aan de applicatieontwikkelaars is gevraagd hoe vaak zij requirements tegenkomen die zij anders interpreteren. Bij deze vragen gebruik gemaakt van een 4-puntsschaal: nooit, soms, vaak en altijd.

Requirements kunnen verkeerd worden geïnterpreteerd. In de enquête zijn vier oorzaken genoemd waardoor requirements verkeerd worden geïnterpreteerd. Aan de respondenten is gevraagd in welke mate zij denken dat één van de bewuste oorzaken de oorzaak is van interpretatieproblemen. De volgende oorzaken zijn in de enquête genoemd:

1. Een woord is in de requirements voor meerdere uitleg vatbaar.
2. De zinsstructuur in de requirement is verwarrend waardoor de requirement voor meerdere uitleg vatbaar is.
3. De requirement is verwarrend met een andere requirement in hetzelfde requirementsdocument.
4. De requirement is verwarrend met een andere requirement in het domein waarin de applicatie wordt gemaakt.

De eerste oorzaak is een lexicale ambiguïteit; een woord heeft meer dan één betekenis. De tweede oorzaak is een syntactische of semantische ambiguïteit. De derde oorzaak kan veroorzaakt worden door een pragmatische of software engineerings ambiguïteit. De laatst genoemde oorzaak is een software engineerings ambiguïteit. Bij deze vragen gebruik gemaakt van een 4-puntsschaal: nooit, soms, vaak en heel vaak.

Domeinkennis
In de interviews werd door respondenten gebrek aan domeinkennis genoemd als één van de oorzaken voor het verkeerd interpreteren van requirements. In de enquête zijn drie activiteiten beschreven om domeinkennis over te dragen. Aan de requirements engineers is gevraagd in welke mate zij de drie activiteiten toepassen in de praktijk. Bij deze vragen is gebruik gemaakt van een 4-puntsschaal: nooit, soms, vaak en altijd. De drie activiteiten zijn: mondelinge toelichting, schriftelijke toelichting bij de requirements en het opnemen van een verklarende woordenlijst in de requirementsdocumenten. Aan de applicatieontwikkelaars is gevraagd hoe vaak requirements engineers deze activiteiten toepassen.

Notatietechniek
Bij het opstellen van de requirements kunnen er verschillende notatietechnieken worden gebruikt. Elke techniek heeft zijn eigen notatie om requirements mee te beschrijven. In hoofdstuk drie wordt beschreven over de problemen die hierbij kunnen optreden. In de enquête is aan de respondenten gevraagd of zij in de praktijk meemaken dat er een notatietechniek wordt gebruikt voor de requirements en zo ja of zij willen invullen welke techniek er gebruikt wordt. Daarnaast is de respondenten in een meerkeuze vraag gevraagd waarom er voor de bewuste notatietechniek gekozen is.

Requirements eigenschappen
Uit het requirementsdocumenten onderzoek bleek dat de requirements vaak implementatie afhankelijk worden opgesteld. De requirements zijn vaak procedureel beschreven. In de enquête is aan de respondenten gevraagd of procedureel de voorkeur heeft ten opzichte van declaratief.
In de requirementsdocumenten werden de requirements opgesteld in een beschrijvend stukje tekst. In

één stuk tekst werden meerdere requirements opgenomen. In de enquête is aan de respondenten gevraagd of deze notatie hun voorkeur heeft. Er is gevraagd of de respondenten hun vraag willen toelichten.

Requirements kwaliteitseigenschappen
Aan de respondenten is in de enquête gevraagd welke kwaliteitscriteria voor requirements zij belangrijk vinden. In de enquête worden over de vijf kwaliteitscriteria vragen gesteld. Per eigenschap is de respondent gevraagd om aan te geven hoe belangrijk hij deze eigenschap vindt voor de requirements. De antwoorden konden gegeven worden op een 5-puntsschaal met de opties: zeer onbelangrijk, onbelangrijk, neutraal, belangrijk en zeer belangrijk.

Eigenschappen gecontroleerde talen
De manier waarop gecontroleerde talen gebruikt moeten worden lijkt op het gebruik van formele of gestructureerde talen. Voor personen die ervaring hebben met één van deze talen is het makkelijker om een gecontroleerde taal te leren gebruiken. In de enquête is aan de respondenten gevraagd of zij ervaring hebben met gestructureerde of formele talen. Deze vraag is gesteld om te achterhalen moeilijk een gecontroleerde taal mag zijn om ook gebruiksvriendelijk te blijven.
Het leren gebruiken van een gecontroleerde taal kost tijd. Aan de respondenten is in de interviews gevraagd hoeveel tijd zij willen investeren om een gecontroleerde taal te leren.
Uit de eigenschappenlijst van Wyner en op basis van de interviews zijn aan de respondenten vragen gesteld in welke mate zij bepaalde eigenschappen belangrijk vinden voor een gecontroleerde taal. De antwoorden konden gegeven worden op een 5-puntsschaal met de opties: zeer onbelangrijk, onbelangrijk, neutraal, belangrijk en zeer belangrijk.
Aan de respondenten zijn de volgende eigenschappen gevraagd: moet de gecontroleerde taal beeldspraak, gezegde, spreekwoorden, afwisseling in structuur van zinnen, tussenzinnen, opsommingen of verwijzingen ondersteunen en moet de taal worden ondersteund door een tool.

<u>Analyse enquête</u>
In de enquête komen open en meerkeuzevragen voor. Voor de meerkeuze vragen zijn hoofdzakelijk 4 en 5-puntsschalen gebruikt. In de meeste gevallen is de 4-puntsschaal gebruikt om de respondenten in te laten vullen in welke mate een activiteit wordt uitgevoerd (nooit, soms, vaak en altijd). De 5-puntsschaal is vooral gebruikt om naar de mening van een respondent te vragen (zeer onbelangrijk, onbelangrijk, neutraal, belangrijk en zeer belangrijk).

De meerkeuzevragen met de 4-puntsschaal zijn geanalyseerd door dezelfde antwoorden van de respondenten bij elkaar op te tellen en te delen door het totaal aantal antwoorden. Het resultaat was een percentage waarin te zien is hoe vaak een antwoord is gegeven ten opzichte van de andere drie antwoorden. De meerkeuzevragen naar de mening van een respondent met een 5-puntsschaal zijn geanalyseerd door het gemiddelde te berekenen. De 5-puntsschaal die gebruikt werd in de vragen is ordinaal. In dit onderzoek is er vanuit gegaan dat de afstand tussen twee opeenvolgende antwoordcategorieën zoals ‘belangrijk’ en ‘zeer belangrijk’ en tussen ‘zeer onbelangrijk’ en ‘onbelangrijk’ gelijk is. Door deze aanname kan voor de analyse gebruik worden gemaakt van analysemethode voor een intervalschaal. Aan de vijf verschillende antwoorden is een gewicht gegeven van 1 tot 5. De opties in het antwoord worden op de volgende manier gewogen: zeer onbelangrijk (1), onbelangrijk (2), neutraal (3), belangrijk (4) en zeer belangrijk (5). Om het gemiddelde te berekenen wordt de gewogen waarde van een antwoord vermenigvuldigd met de frequentie hoe vaak het antwoord is ingevuld. Deze getallen worden bij elkaar opgeteld en gedeeld door het aantal respondenten dat het antwoord heeft ingevuld. Het gemiddelde antwoord van een vraag is de gewogen waarde die het dichts bij het gemiddelde getal ligt. Als voorbeeld het gemiddelde getal 4,2. Dit getal ligt het dichts bij de gewogen waarde 4 van ‘belangrijk’. Het gemiddelde antwoord is in dit geval dan ook ‘belangrijk’. Om de spreiding van de antwoorden aan te geven is ook de standaarddeviatie berekend.

In de open vragen in de enquete hebben respondenten opgegeven waarom zij een bepaald antwoord hebben gekozen in de voorgaande meerkeuze vraag; in de enquête volgden de open vragen op de meerkeuze vragen om de respondent te vragen naar de reden waarom zij een antwoord gekozen

hebben. In de analyse zijn de antwoorden op deze open vragen gebruikt om de antwoorden van de meerkeuzevragen te verduidelijken.

4.1.4 *Selectie van de gecontroleerde talen*

Om de scope van het onderzoek te beperken zijn er aan het begin van dit onderzoek vier CNL talen geselecteerd die interessant leken voor het maken van requirements. De CNL talen zijn geselecteerd op basis van artikelen in de wetenschappelijke literatuur. Er is een inventarisatie gedaan in verschillende wetenschappelijke zoeksystemen en databanken naar artikelen over CNLs. Dit waren de volgende zoeksystemen: IEEE Xplore[8], SpringerLink[9], ACM Digital Libary[10]. Er is gezocht naar publicaties over Controlled Natural Languages. De zoekopdrachten die gebruikt zijn in de zoeksystemen zijn *"controlled natural language"* en de afkorting "*cnl*". De zoekopdrachten resulteerde in een lijst met publicaties. Per zoeksysteem zijn de eerste 250 resultaten onderzocht. Binnen deze lijst zijn alleen de artikelen geselecteerd die in het Engels of Nederlands zijn geschreven. Van de lijst met publicatie zijn eerst de titels doorgenomen en beoordeeld op de relevantie voor het onderzoek. De publicaties vielen af wanneer vanuit de titel op te maken viel dat de publicatie geen relatie had tot CNL. De titels met daarin de woorden *"Controlled Natural Language"* of "*CNL*" zijn direct geselecteerd. Wanneer niet vanuit de titel op te maken viel of de publicatie over CNL ging is de samenvatting of het gehele artikel door gelezen. Daarnaast is er een speciale selectie toegepast op artikelen die een specifieke CNL taal benoemde of beschreven. Artikelen waarin geen CNL taal werd benoemd vielen af. Om deze tweede selectie te maken was het nodig dat de artikelen eerst werden doorgelezen.

De volgende talen zijn geselecteerd voor dit onderzoek, in hoofdstuk drie is meer te lezen over deze talen.

1. Attempto Controlled English (ACE)
2. Processable ENGlish (PENG)
3. Common Logic Controlled English (CLCE)
4. Computer Processable Language (CPL)

<u>Opzet</u>

De vier CNL talen zijn met elkaar vergeleken om de taal te selecteren die het meest in aanmerking komt voor het opstellen van requirements. De talen zijn met elkaar vergeleken op basis van de eigenschappen die de taal heeft. Per taal is onderzocht of deze wel of niet bepaalde eigenschappen heeft. Het onderzoek is gedaan met een eigenschappenlijst. De eigenschappenlijst is opgesteld op basis van de vier dimensies die Kuhn (2012) noemt en de eigenschappenlijst voor CNL talen van Wyner (2010). De eigenschappenlijst bestaat uit vier onderwerpen: nauwkeurigheid, expressiviteit, natuurlijk karakter en eenvoud. Per onderwerp zijn er een aantal eigenschappen gedefinieerd. De eigenschappen komen uit de lijst van Wyner, interviews of enquête.
Per eigenschap kunnen in de lijst twee of drie antwoorden worden ingevuld. Een taal die volledig over de eigenschap beschikt heeft een plus gekregen voor de eigenschap. Een taal die niet volledig over de eigenschap beschikt heeft een plus / min gescoord op de eigenschappen. Een taal waarin de eigenschap niet voorkomt heeft een min. In de eigenschappenlijst is beschreven hoe er per eigenschap beoordeelt moet worden wat de score voor een taal is.

Nauwkeurigheid (Precision)

De nauwkeurigheid dimensie in het PENS schema geeft aan in welke mate van de tekstuele vorm van een zin direct de betekenis kan worden afgeleid. Een natuurlijke taal is erg onnauwkeurig; er is veel context informatie nodig om de betekenis van een zin te begrijpen. Formele talen zijn erg nauwkeurig omdat de betekenis van de zin direct kan worden omgezet vanuit de symbolen die gebruikt worden. In tabel 4.3 staan vier typen ambiguïteiten. Op basis van deze vier typen worden de CNL talen

---

8 http://ieeexplore.ieee.org

9 http://www.springerlink.com

10 http://dl.acm.org

vergeleken hoe nauwkeurig deze zijn. Wanneer door de CNL taal wordt voorkomen dat al de ambiguïteitstypen in de requirements komen is de CNL erg nauwkeurig.

Eigenschap	Antwoord mogelijkheid
Lexicale ambiguïteit	- In de CNL taal wordt niets gedaan om lexicale ambiguïteit te voorkomen. Al de woorden van een natuurlijke taal zijn toegestaan. +/- In de CNL taal wordt polysemie en systematische polysemie op een beperkte schaal voorkomen door gebruik te maken van voor gedefinieerde woorden en domeinwoorden. Domein woorden kunnen door de gebruiker worden ingevoerd in een woordenlijst om de betekenis te beperken. + In de CNL taal is lexicale ambiguïteit niet mogelijk. De CNL taal beschikt over een eigen voor gedefinieerde woordenschat of maakt gebruik van een corpus waarin de betekenis van woorden beperkt is tot één betekenis.
Syntactische ambiguïteit	- In de CNL taal wordt niets gedaan om syntactische ambiguïteit te voorkomen. In de syntax van de taal zijn geen beperkingen aangebracht om deze vorm van ambiguïteit te voorkomen. +/- In de CNL taal worden twee van de vier subtypen syntactische ambiguïteiten voorkomen. De vier subtypen zijn analytische, attachment, coördinatie en elliptische ambiguïteit. + In de CNL taal kan geen syntactische ambiguïteit voorkomen. Al de vier de subtypen syntactische ambiguïteit worden voorkomen met de CNL taal.
Semantische ambiguïteit	- In de CNL taal wordt niets gedaan om semantische ambiguïteit te voorkomen. In de semantische regels van de taal zijn geen beperkingen aangebracht om deze vorm van ambiguïteit te voorkomen. +/- In de CNL taal worden één van de twee subtypen semantische ambiguïteit voorkomen. De twee subtypen zijn scope ambiguïteit en referentiële ambiguïteit in de context van de zin. + In de CNL taal kan geen semantische ambiguïteit voorkomen. De twee subtypen semantische ambiguïteit worden voorkomen met de CNL taal.
Pragmatische ambiguïteit	- In de CNL taal wordt niets gedaan om pragmatische ambiguïteit (alleen subtype referentiële ambiguïteit) te voorkomen. In de zinnen van de CNL taal kan gebruik worden gemaakt van verwijswoorden zonder dat hier syntactische of semantische regels voor zijn. +/- In de CNL taal mag geen gebruik worden gemaakt van verwijswoorden. Het risico op referentiële ambiguïteit is hierdoor erg laag. + Referentiële ambiguïteit kan niet voorkomen in de CNL taal. In de CNL taal zijn syntactische of semantische regels die de betekenis van een verwijswoord beschrijven.

Tabel 4.3 Eigenschappen en indicatoren voor de nauwkeurigheid van een CNL taal.

Expressiviteit (Expressiveness)

De expressiviteit dimensie in het PENS schema beschrijft het aantal proposities die gebruikt kunnen worden met een CNL taal. Een taal X is expressiever dan een taal Y wanneer taal X alles kan beschrijven van taal Y maar niet andersom. In tabel 4.4 staan de eigenschappen die gemaakt zijn op basis van de resultaten van de interviews en enquête. De indicatoren komen uit de eigenschappenlijst van Wyner en uit de resultaten van de interviews en enquête.

Eigenschap	Antwoord mogelijkheid
Maakt de CNL gebruik van een eigen woordenlijst	- De CNL maakt geen gebruik van een eigen woordenlijst. +/- In de CNL zijn voor gedefinieerde woorden beschikbaar die gebruikt mogen worden. + De CNL maakt gebruik van een Corpus.
Kunnen er woorden worden toegevoegd aan de woordenlijst	- Aan de woordenlijst mogen geen woorden worden toegevoegd. + Aan de woordenlijst mogen domeinwoorden worden toegevoegd.
Meervoud / enkelvoud	- In de CNL taal kan geen gebruik worden gemaakt van woorden die meervoud zijn. + In de taal zijn er regels om woorden enkelvoud of meervoud te maken.
Heeft de CNL taal regels	- In de CNL taal zijn er geen regels voor nominalisatie.

voor nominalisatie.	+	In de CNL taal zijn er regels voor nominalisatie. In de CNL taal is het mogelijk om een woord te veranderen in een zelfstandig naamwoord.
Is een andere schrijfwijze van het woord toegestaan.	+	In de CNL taal moeten woorden altijd op dezelfde manier worden geschreven.
	-	In de taal mogen woorden op een andere manier worden geschreven. Voorbeelden hiervan is het gebruik van afkortingen.
Kunnen er verwijzingen worden gemaakt.	-	In de CNL taal kunnen geen verwijzingen worden gemaakt.
	+/-	In de taal kunnen instantie van een object op meerdere plaatsen gebruikt worden. Voorbeeld "A user opens the view. The user close the view". *"A user"* en *"the user"* is hetzelfde object alleen een andere instantie.
	+	In de CNL taal kunnen verwijzingen worden gemaakt. De CNL taal ondersteunt anaforen die verwijzen naar antecedent in dezelfde zin of in de tekst.
Kunnen er opsommingen worden gemaakt	-	Het is niet mogelijk om met de taal opsommingen te maken.
	+/-	Het maken van opsommingen op de standaard manier (gescheiden door komma of opsomming) wordt niet ondersteund maar er zijn wel andere mogelijkheden beschikbaar in de taal om opsommingen te maken.
	+	In de taal kunnen opsommingen worden gemaakt op de volgende manier: Het maken van een opsomming in een zin gescheiden door een komma of het maken van een lijst.
Zijn tussenzinnen toegestaan.	-	Tussenzinnen zijn niet toegestaan in de CNL taal.
	+	Een structuur zoals "De gebruiker, dit is een persoon in rol x, moet inloggen op het systeem" is toegestaan in de CNL.
Is beeldspraak toegestaan.	-	In de CNL is geen beeldspraak toegestaan.
	+	In de CNL taal is beeldspraak toegestaan. Onder beeldspraak vallen de volgende soorten: idiomen, metaforen en uitdrukkingen.
Zijn gezegde toegestaan.	-	Een gezegde is niet toegestaan in de CNL taal.
	+	Een gezegde is toegestaan in de CNL taal.
Zijn spreekwoorden toegestaan.	-	Een spreekwoord is niet toegestaan in de CNL taal.
	+	Een spreekwoord is toegestaan in de CNL taal.

Tabel 4.4 Hoe expressief is de CNL.

Natuurlijk karakter (Naturalness)

De naturalness dimensie in het PENS schema beschrijft in welke mate de taal op een natuurlijke taal lijkt in termen van leesbaarheid en begrijpelijkheid. Het grote nadeel van formele talen is dat ze vaak nogal moeilijk te begrijpen zijn voor domein specialisten en stakeholders. Omdat een CNL afgeleid is van een natuurlijke taal is het voor personen die de natuurlijke taal beheersen makkelijker om deze te leren, gebruiken, lezen en schrijven in vergelijking met andere technische talen zoals programmeertaal of een formele taal. In tabel 4.3 staan de eigenschappen en indicatoren om vast te stellen hoe eenvoudig er met een CNL gewerkt kan worden. De genoemde indicatoren komen uit de lijst van Wyner (2010).

Eigenschap		**Antwoord mogelijkheid**
Is de taal makkelijk te leren?	-	De regels van de CNL zijn formeel opgesteld en ontoegankelijk voor gebruikers zonder kennis van formele talen.
	+/-	De restrictie op de grammatica en semantiek zijn in de CNL taal toegankelijk beschreven. Het aantal regels is beperkt en de regels zijn niet formeel opgesteld.
	+	De regels van de CNL taal zijn volledig in de tool van de taal geïmplementeerd. Voor een gebruiker is het niet nodig om de regels van de taal te leren.
Is de taal makkelijk te lezen?	-	In de CNL taal wordt gebruik gemaakt van moeilijke taalconstructie zoals "not at least". In de taal zijn ingewikkelde constructie nodig om eenvoudige dingen duidelijk te maken.
	+	In de CNL taal kunnen geen ingewikkelde taalconstructie worden gemaakt.
Is de taal makkelijk te schrijven?	-	Is er geen "syntactic sugar" in de taal aanwezig die het schrijven vergemakkelijkt.
	+	In de CNL taal wordt gebruik gemaakt van syntactic sugar om het schrijven van zinnen te vergemakkelijken. Met syntactic sugar zijn er verschillende manieren om een tekst te formuleren terwijl de tekst op dezelfde manier wordt geïnterpreteerd.
Is de taal makkelijk te begrijpen?	-	De CNL taal voelt niet natuurlijk aan. Zinnen die worden gemaakt met de CNL taal hebben een andere betekenis als lezers in een natuurlijke taal aan de zin zouden geven.

	+	De CNL taal voelt natuurlijk aan. Zinnen die worden gemaakt met de CNL taal hebben een zelfde betekenis als lezers in een natuurlijke taal aan de zin zouden geven.

Tabel 4.5 Hoe eenvoudig is de CNL in gebruik.

Eenvoud (Simplicity)
De simplicity dimensie in het PENS schema beschrijft hoe eenvoudig of complex het is om de taal om te zetten naar een formele taal. Met deze dimensie meet Kuhn hoeveel inspanning het kost om de taal in een computer programma te implementeren. In tabel 4.6 staan de eigenschappen die een relatie hebben met deze dimensie. De eigenschappen en indicatoren komen uit de lijst van Wyner.

Eigenschap		**Antwoord mogelijkheid**
Worden zinnen omgezet naar een formele taal.	-	De zinnen in de CNL worden niet omgezet naar een formele taal.
	+/-	Zinnen worden omgezet naar een formele taal met een één op veel relatie (ambiguïteit). Wanneer er een aantal interpretaties mogelijk zijn moet de gebruiker de juiste interpretatie kiezen.
	+	De zinnen van de CNL taal worden één op één overgezet naar een formele taal. Bij het overzetten is er geen ambiguïteit of zijn er meerdere mogelijkheden.
Wordt de discours omgezet naar een formele taal.	-	De discours van de CNL zin wordt niet omgezet naar logica.
	+/-	In de CNL taal kunnen gebruikers zelf de discource definiëren. Door het aanvullen van de woordenschat of het definiëren van zinnen met daarin kennis van het domein.
	+	De discours van de CNL zin wordt omgezet naar logica.
Is er een mapping naar een grafische presentatie mogelijk.	-	Een zin in de CNL taal kan niet worden omgezet naar een grafische representatie.
	+	Een zin in de CNL taal kan worden omgezet naar een grafische representatie.
Ondersteunt de CNL taal meerder natuurlijke talen.	-	De CNL taal is gebaseerd op één natuurlijke taal.
	+	De CNL taal ondersteunt meer dan één natuurlijke taal.
Beschikt de CNL over een style richtlijn	-	De CNL taal beschikt niet over richtlijnen.
	+	Er is een richtlijn voor de CNL style beschikbaar die de gebruiker instructies geeft zoals “schrijf korte en enkelvoudige zinnen”
Wordt de CNL taal ondersteunt door een tool?	-	Er is geen tool beschikbaar voor de CNL taal.
	+/-	Er is tool beschikbaar voor de CNL taal die het opstellen van zinnen in de taal ondersteunt.
	+	Is er een tool beschikbaar die het opstellen van zinnen in de taal ondersteunt. De tool beschikt over automatische consistentie controle en over redundantie controle

Tabel 4.6 Welke algemene eigenschappen heeft de CNL taal.

Analyse
Het doel van het onderzoek is om de CNL taal te selecteren die het meest geschikt is om requirements op een kwalitatieve en gebruiksvriendelijke manier mee te definiëren. In de vorige paragraaf is beschreven hoe de talen worden vergeleken. In deze paragraaf wordt beschreven hoe de taal is geselecteerd die het meest geschikt is.

In het requirementsdocumenten onderzoek en ook in de interviews en enquête is onderzocht welke typen ambiguïteiten het meest voorkomen in de requirements. Op basis van deze onderzoeken hebben de eigenschappen van nauwkeurigheid een waarde gekregen; de eigenschap is zeer onbelangrijk, onbelangrijk, neutraal, belangrijk of zeer belangrijk om de kwaliteit van de requirements te verbeteren. In de interviews en enquête zijn de respondenten bevraagd op eigenschappen die zij belangrijk vinden voor een gecontroleerde taal om daarmee op een gebruiksvriendelijke wijze requirements mee op te stellen. Op basis van deze onderzoeken hebben de eigenschappen van expressiviteit, natuurlijk karakter en eenvoud een waarde gekregen; de eigenschap is zeer onbelangrijk, onbelangrijk, neutraal, belangrijk of zeer belangrijk voor de gebruiksvriendelijkheid van de gecontroleerde taal.

In de eigenschappenlijst is er bij een taal per eigenschap aangeven of deze wel (+), ten dele (+/-) of niet (-) voorkomt in de taal. De drie antwoorden hebben een gewicht gekregen. Plus heeft een gewicht van 1, plus / min heeft een gewicht van 0,5 en min heeft een gewicht van 0. De waarde van het belang van een eigenschap hebben ook een gewicht gekregen: zeer onbelangrijk (1), onbelangrijk (2), neutraal (3), belangrijk (4) of zeer belangrijk (5). Door het gewicht van de waarde van de eigenschap te vermenigvuldigen met het gewicht van het antwoord per taal op de eigenschap; heeft een taal per eigenschap een score gekregen. Om de methode te verduidelijken is het volgende voorbeeld opgenomen: De taal ACE beschikt over de eigenschap "*voorkomen van syntactische ambiguïteit*", de taal heeft voor deze eigenschap een plus (+) gekregen. De eigenschap "*voorkomen van syntactische ambiguïteit*" is zeer belangrijk om de kwaliteit van de requirements te verbeteren, het gewicht van deze eigenschap is 5. Het plus teken heeft een gewicht van 1, de score van deze taal op de eigenschap "voorkomen van syntactische ambiguïteit" is 1 keer 5.

Per onderwerp zijn de score van de eigenschappen per taal bij elkaar opgeteld. Hierdoor is er voor elke eigenschap een taal die het best scoort op kwaliteit of gebruiksvriendelijkheid. In het eindresultaat zijn de score van al de eigenschappen per taal bij elkaar opgeteld. De taal met de hoogste score is het meest geschikt om op een gebruiksvriendelijke manier kwalitatieve requirements mee op te stellen.

4.2 Onderzoeksstructuur

Het onderzoek bestaat uit een onderzoeksvraag met vier deelvragen. Om de onderzoeksvraag te beantwoorden zijn er vijf deelvragen opgesteld. In deze paragraaf wordt beschreven hoe deze vragen zijn beantwoord in dit onderzoek. In de eerste plaatst zullen de vijf deelvragen worden opgesomd met daarbij de beantwoordingswijze, daarna zal beschreven worden hoe de hoofdvraag wordt beantwoord.

1. Hoe kan ambiguïteit op een betrouwbare manier worden opgespoord in de requirements?
In het requirementsdocumenten onderzoek is een methode onderzocht waarmee ambiguïteiten in de requirements kunnen worden opgespoord. Uit de resultaten van het requirementsdocumenten onderzoek blijkt hoe effectief deze methode is die hiervoor gebruikt is.

2. Welke factoren beïnvloeden in de praktijk de kwaliteit van de requirements?
In het requirementsdocumenten onderzoek is de kwaliteit van de requirements onderzocht. In dit onderzoek is voornamelijk de eigenschap eenduidigheid onderzocht. De eenduidigheid van de requirements wordt vastgesteld door het percentage ambiguïteiten dat gevonden wordt in de requirementsdocumenten. Naast de eigenschap eenduidig is in de requirementsdocumenten ook gekeken of er nog andere eigenschappen zijn die invloed op de kwaliteit van het document hebben. Op basis van de resultaten van het requirementsdocumenten onderzoek is het interview en de enquête opgesteld. In deze twee onderzoeksactiveiten is dieper ingegaan op de kwaliteitsfactoren die in het requirementsdocumenten onderzoek gevonden zijn.

3. Wat zijn de wensen vanuit de praktijk voor een gebruiksvriendelijke gecontroleerde natuurlijke taal?
In het interview en de enquête zijn respondenten uit de praktijk vragen gesteld over het opstellen en gebruiken van requirements. De vragen zijn gebaseerd op eigenschappen van gecontroleerde talen zoals deze in de CNL-taal checklist staan. De respondenten hebben aangegeven hoe belangrijk zij deze eigenschappen vinden voor de requirements. Op basis van de resultaten van het interview en de enquête is in de CNL-taal checklist aangegeven welke eigenschappen belangrijk zijn voor de gebruiksvriendelijkheid van een gecontroleerde taal.

4. Welke van de vier geselecteerde gecontroleerde natuurlijke talen heeft de juiste eigenschappen (kwaliteit en gebruiksvriendelijk) om daarmee requirements te definiëren?
De gecontroleerde talen zijn met elkaar vergeleken met de CNL-taal checklist. In de checklist is aangegeven welke eigenschappen belangrijk zijn voor kwaliteit en / of gebruiksvriendelijkheid. De taal die het beste scoort op beide eigenschappen is geselecteerd als beste gecontroleerde taal om requirements mee op te stellen.

Hoofdvraag: *"Waar moet een gecontroleerde natuurlijke taal aan voldoen om de kwaliteit van software requirements in de praktijk op een gebruiksvriendelijke manier te verbeteren?"*
Uit verschillende onderzoeksactiviteiten zijn factoren gevonden die invloed hebben op de kwaliteit van de requirements; op basis van deze factoren zijn eigenschappen gedefinieerd die de kwaliteit van de requirements verbeteren. In het interview en de enquête zijn respondenten uit de praktijk vragen gesteld over het opstellen en gebruiken van requirement, op basis van de resultaten hier van zijn eigenschappen gedefinieerd waaraan een gebruiksvriendelijke gecontroleerde taal moet voldoen.
In het onderzoek is een CNL taal geselecteerd die het best aan de eigenschappen voor kwaliteit en gebruiksvriendelijkheid voldoet.

5. Onderzoeksresultaten

In dit hoofdstuk worden de resultaten van het onderzoek weergegeven. Het hoofdstuk bestaat uit twee paragrafen. In de eerste paragraaf worden de resultaten per onderzoeksactiviteit genoemd. In de tweede paragraaf worden de resultaten van de onderzoeksactiviteiten samengenomen en op onderwerp geanalyseerd.

5.1 Resultaten

In deze paragraaf worden de resultaten van de verschillende onderzoeksactiviteiten weergegeven.

5.1.1 Ambiguïteiten opsporen in requirements

Bij het onderzoek naar de kwaliteit van de requirements is onderzocht hoeveel requirements ambigu zijn in een document. Voorafgaande hieraan is nagegaan hoe ambiguïteiten op een betrouwbare manier gevonden kunnen worden. In deze paragraaf staan de resultaten hiervan beschreven.
Door vier respondenten zijn dertig requirements (requirementslijst staat in bijlage A.1) onderzocht op ambiguïteit. Hierbij hebben de eerste drie respondenten (r1, r2 en r3) gebruik gemaakt van de vertaalde checklist van Kamsties, Berry en Paech. Eén respondent (r4) heeft de checklist gebruikt die is aangepast op basis van de feedback van de eerste respondenten (zie paragraaf 4.1.1).

De respondenten hebben gemiddeld 25 van de 30 requirements aangemerkt als ambigu. Respondent r2 heeft de meeste requirements aangemerkt als ambigu; 29 van de 30. De respondent die gebruik gemaakt heeft van de nieuwe checklist heeft 27 requirements aangemerkt als ambigu. De andere twee respondenten (r1 en r3) hebben beide 21 requirements aangemerkt als ambigu. In sommige requirements hebben de respondenten meer dan één ambiguïteit gevonden. In tabel 5.1 is te zien hoeveel ambiguïteiten de respondenten gevonden hebben, en hoe zij deze geclassificeerd hebben. In bijlage A.2 staat het uitgebreide resultaat; de ambiguïteiten die de respondenten gevonden en geclassificeerd hebben.

Type ambiguïteit	**R1**	**R2**	**R3**	**R4**
Polysemie	*9*	*9*	*5*	*18*
Systematische polysemie	*3*	*4*	*3*	*4*
Referentiële ambiguïteit	*15*	*18*	*6*	*7*
Discourse ambiguïteit	*2*	*2*	*3*	*6*
Domein ambiguïteit	*2*	*2*	*6*	*7*
Totaal aantal gevonden ambiguïteiten	***31***	***35***	***23***	***42***

Tabel 5.1 Aantal gevonden ambiguïteiten in de requirements per respondent.

De referentiële ambiguïteit is het meest door de respondenten gevonden. De respondent r4 (met vernieuwde checklist) heeft de meeste ambiguïteiten gevonden. In tabel 5.1 is te zien dat deze respondent veel ambiguïteiten van het subtype polysemie heeft gevonden. Dit komt omdat de respondent veel afkortingen aangemerkt heeft als ambiguïteit. Zoals blijkt uit de resultaten van het requirementsdocumenten onderzoek is dit niet verkeerd. Bij de requirementsdocumenten worden namelijk geen verklarende woordenlijsten opgenomen waarin de betekenis van woorden en afkortingen wordt uitgelegd. Door het resultaat van respondent r4 wordt wel uitgesloten dat de aanpassingen aan de checklist er de oorzaak van is dat de respondent meer ambiguïteiten vindt.

De vier respondenten hebben in totaal 91 verschillende ambiguïteiten gevonden in de requirements. In tabel 5.1 is te zien dat de meeste respondenten maar een derde van het totaal aantal gevonden

ambiguïteiten vinden. Eén persoon is dus niet in staat om alle ambiguïteiten in de requirements te vinden. In tabel 5.2 is te zien hoeveel van de gevonden ambiguïteiten van één respondent ook door de andere respondenten gevonden zijn. In de tabel is een onderverdeling gemaakt van het aantal gevonden ambiguïteiten door één, twee of drie andere respondenten. Op de laatste rij van de tabel staat het percentage van het aantal gevonden ambiguïteiten die ook door een andere respondenten gevonden zijn.

	R1	**R2**	**R3**	**R4**	**Gemiddeld**
Gevonden door één andere respondent.	*15*	*10*	*3*	*4*	*8*
Gevonden door twee andere respondenten.	*9*	*7*	*8*	*6*	*7,5*
Gevonden door drie andere respondenten.	*1*	*1*	*1*	*1*	*1*
Totaal gevonden door andere respondent	***25***	***18***	***12***	***11***	***16,5***
Percentage van dezelfde ambiguïteiten	*81%*	*51%*	*52%*	*26%*	*53%*

Tabel 5.2 Aantal dezelfde gevonden ambiguïteiten per respondent in de requirements.

Uit de resultaten blijkt dat gemiddeld 47% van de gevonden ambiguïteiten door één respondent niet door een andere respondent gevonden worden. Door de aanpassing van de checklist is het vinden van dezelfde ambiguïteiten niet verbeterd. Respondent (r4) heeft het minst aantal van dezelfde ambiguïteiten gevonden. In tabel 5.3 is te zien hoeveel ambiguïteiten de respondenten gemiddeld per requirement vinden.

	R1	**R2**	**R3**	**R4**
Totaal gevonden in 30 requirements	*31*	*35*	*23*	*42*
Gemiddeld gevonden in 1 requirement	*1,0*	*1,2*	*0,8*	*1,4*

Tabel 5.3 Gemiddeld aantal gevonden ambiguïteiten per respondent.

Totaal zijn er door vier respondenten 91 verschillende ambiguïteiten gevonden. Hiermee vinden vier respondenten gemiddeld 2,6 ambiguïteiten in één requirement, terwijl één respondent gemiddeld 1,1 ambiguïteit vindt in één requirement. In twee requirements worden dus door vier personen ongeveer vijf ambiguïteiten gevonden en door één persoon twee.

Met de respondenten zijn de ambiguïteiten besproken die zij niet hebben gevonden. Overheen lezen is de reden die zij opgeven als oorzaak voor het niet vinden van een ambiguïteit. Dit komt overeen met wat Popescu, Rugaber, Medvidovic en Berry (2008) schrijven. Een menselijke reviewer leest snel over een ambiguïteit heen in het requirementsdocument, dit komt omdat hij aanneemt dat de eerste interpretatie van de requirement die hem te binnen schiet de juiste is. De reviewer is zich meestal onbewust van een andere mogelijke betekenis van de requirement.

Bij het onderzoek naar de requirementsdocumenten is gebruik gemaakt van een checklist om ambiguïteiten te vinden en te classificeren. In tabel 5.4 staat de checklist die gebruikt is. De checklist is gemaakt op basis van de ambiguïteiten checklist van Kamsties, Berry en Paech (2001). De checklist van Kamsties, Berry en Paech (2001) is vertaald vanuit het Engels naar het Nederlands. Op basis van feedback van respondenten zijn er extra voorbeelden toegevoegd aan de checklist. De ambiguïteitstype “syntactische ambiguïteit” en “taalfout ambiguïteit” zijn toegevoegd aan de checklist.

Type	Beschrijving
Polysemie *(Lexicale ambiguïteit)*	*Een woord kan meer dan één betekenis hebben.* Bij polysemie heeft een woord meer dan één betekenis in de requirement. Let hierbij op dat er lexicale ambiguïteit kan ontstaan in het bijzonder bij de betekenis van een woord in de context van het RE domein. Voorbeeld: Het woord *groen;* dit kan een kleur betekenen maar ook in sommige contexten jeugdig en onervaren.
Systematische polysemie *(Lexicale ambiguïteit)*	Systematische polysemie komt voor op de volgende manieren: De betekenis van een woord is verwarrend tussen klassen en instantie. Voorbeeld: "*Ik vind deze jas leuk"*. Dit kan verwijzen naar een individuele kas of naar het type jas. De betekenis van een woord is verwarrend tussen proces en product. Voorbeeld: "*building*" of *"writing"*. Het zelfstandig naamwoord verwijst naar een proces of naar een product. De betekenis van een woord is verwarrend tussen gedrag en dispositie. Voorbeeld: *"Dit is een snelle auto",* kan verwijzen naar het gedrag of naar de klasse van de auto zoals een Ferrari. De betekenis van een woord is verwarrend tussen een blijvende of een tijdelijke eigenschap van een object. De betekenis is verwarrend doordat enkelvoud en meervoud door elkaar worden gehaald.
Analytische ambiguïteit *(Syntactische ambiguïteit)*	Komt voor wanneer een rol in een zin meerdere betekenissen kan hebben. Een voorbeeld hiervan is de zin: "De Engelse geschiedenis leraar". De betekenissen van deze zin kunnen zijn: "De geschiedenis leraar uit Engeland" of "De geschiedenis leraar die gespecialiseerd is in Engeland".
Attachment ambiguïteit *(syntactische ambiguïteit)*	Treedt op wanneer een bepaalde syntactische bestanddeel van een zin, zoals een voorzetsel, kan worden bevestigd aan twee delen van een zin. Een voorbeeld hiervan is: "De politie schoot op de relschoppers met geweren". Het deel met geweren kan bij het zelfstandig naamwoord relschoppers horen of bij het werkwoord schoot. De betekenissen kunnen dan zijn "De politie schoot op relschoppers die geweren hebben" of "De politie schoot met geweren op relschoppers".
Coördinatie ambiguïteit *(Syntactische ambiguïteit)*	Komt voor wanneer meer dan één conjunctie, "en" of " of", wordt gebruikt in een zin of wanneer een conjunctie wordt gebruikt met een beperkend bijwoord. Voorbeeld van de eerste soort is: "Ik zag Peter en Paul en Mary zag mij". Deze zin kan zijn Ik zag (Peter en Paul) en Mary zag mij. Of "Ik zag Peter en (Paul en Mary) zagen mij". Een voorbeeld van de tweede soort is "Jonge man en vrouw". Deze zin kan de betekenis hebben van "Jonge man en jonge vrouw" of "jonge man en een vrouw".
Elliptische ambiguïteit *(Syntactische ambiguïteit)*	Is een gat in een zin die wordt veroorzaakt door weglating van een lexicaal of syntactisch noodzakelijk bestanddeel. Een voorbeeld hiervan in een zin is: "Peter kent een rijkere man dan Paul". Deze zin heeft twee betekenissen; Peter kent een man die rijker is dan Paul" of "Peter kent een man die rijker is dan al de mannen die Paul kent".
Referentiële ambiguïteit (zin) *(Semantische ambiguïteit)*	De verwijzing naar een woord of naar een deel van de zin is verwarrend. Een woord of een deel van de zin in een requirement verwijst naar meer dan één object in de requirement. Controleer op voornaamwoorden (persoonlijke, bezittelijke, aanwijzende), zelfstandige naamwoorden en ellips. Voorbeeld: "*Anke heeft maar één zus en die woont al jaren in Noorwegen, ze zou het wel wat gezelliger vinden om wat dichter bij haar in de buurt te wonen, maar ze kan het prachtige Scandinavische landschap niet missen."* In het tweede gedeelte van de zin is niet duidelijk naar wie er verwezen wordt, Anke of haar zus.
Referentiële ambiguïteit (context) *(Pragmatische ambiguïteit)*	Meer dan één interpretatie op basis van de verwijzing naar voorgaande requirements. Deze ambiguïteit komt voor wanneer; - woorden zoals, eerste, vorige, tussen, na en laatste worden gebruikt die kunnen verwijzen naar verschillende elementen. - bijvoeglijke naamwoorden, werkwoorden of zelfstandige naamwoord zinnen verwijzen naar meer dan één conditie die beschreven is in een voorgaande requirement.
Taalfout ambiguïteit	Een taalfout ambiguïteit treedt op wanneer een grammaticale, interpunctie, woord keuze of een andere fout in het gebruik van de taal leidt tot een tekst die op meerdere manieren geïnterpreteerd kan worden dan wat de intentie van de auteur was.
Domein ambiguïteit *(Software engineerings ambiguïteit)*	Is de requirement ambigu in relatie tot de kennis van de applicatie, systeem of het ontwikkeld domein. Voorbeeld: "*De deuren van een lift openen niet bij een verdieping, tenzij de lift stilstaat op die verdieping*". De actie openen kan worden uitgevoerd door de hardware of software. Wanneer dit door de hardware gedaan wordt is implementatie in de software niet meer nodig.

Tabel 5.4 Checklist voor het opsporen en classificeren van ambiguïteiten.

5.1.2 *Onderzoek requirementsdocumenten*

In het requirementsdocumenten onderzoek is de kwaliteit van de requirements van VitalHealth onderzocht. In paragraaf 3.1 zijn een aantal kwaliteitscriteria beschreven voor requirementsdocumenten. De focus in dit onderzoek lag vooral op het criterium eenduidig. Naast eenduidig zijn er ook nog een aantal andere opvallende zaken in de documenten gevonden. Deze zaken hebben te maken met twee andere kwaliteitscriteria.
De drie onderzochte documenten zijn opgesteld in een natuurlijke taal. Twee (Project 1 en Project 3) zijn opgesteld in het Engels en één document (project 2) is opgesteld in het Nederlands.

Eenduidig

In het onderzoek zijn er 266 requirements gecontroleerd op ambiguïteit. In 161 requirements is minstens één ambiguïteit gevonden. Dit houdt in dat 60% van de requirements ambigu is. In het eerste document is 51% van de requirements ambigu. In het tweede document is 85% van de requirements ambigu en in het derde document is 61% van de requirements ambigu. Gemiddeld is 66% van de requirements ambigu in de requirementsdocumenten van VitalHealth. In tabel 5.5 staat hoeveel ambiguïteiten er per project gevonden zijn.

	Project 1	**Project 2**	**Project 3**	**Totaal**
Aantal requirements	*119*	*39*	*108*	*266*
Aantal ambigu requirements	*61*	*33*	*66*	*161*
Percentage ambigu	*51%*	*85%*	*61%*	*60%*

Tabel 5.5 Aantal requirements die ambigu zijn per requirementsdocument.

In diagram 5.1 is het totaal aantal requirements te zien die per project onderzocht zijn. Het requirementsdocument van project 1 bestond uit de meeste requirements. In diagram 5.1 is het gedeelte waarin ambiguïteiten voorkwamen aangegeven.

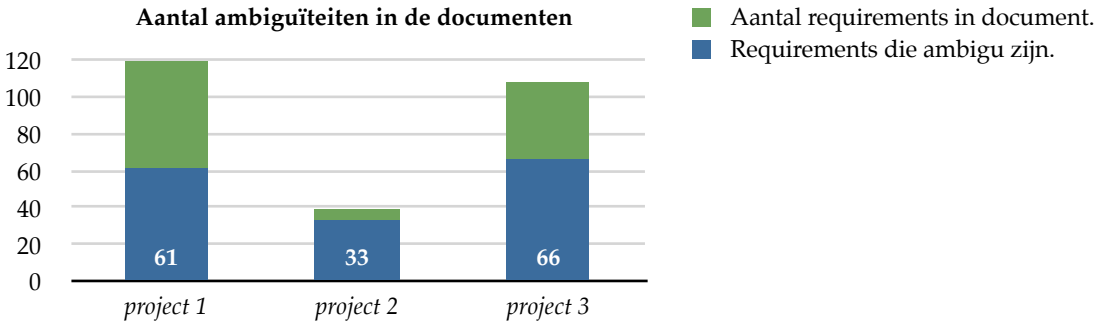


Diagram 5.1 Totaal aantal requirements die onderzocht zijn met daarbij aangegeven het gedeelte welke ambigu is.

Linguïstische ambiguïteiten

In de 266 requirements zijn er 227 linguïstische ambiguïteiten gevonden. In een aantal requirements is er meer dan één linguïstische ambiguïteit gevonden. In diagram 5.2 is de onderverdeling te zien tussen de typen linguïstische ambiguïteiten die gevonden zijn in de requirements.

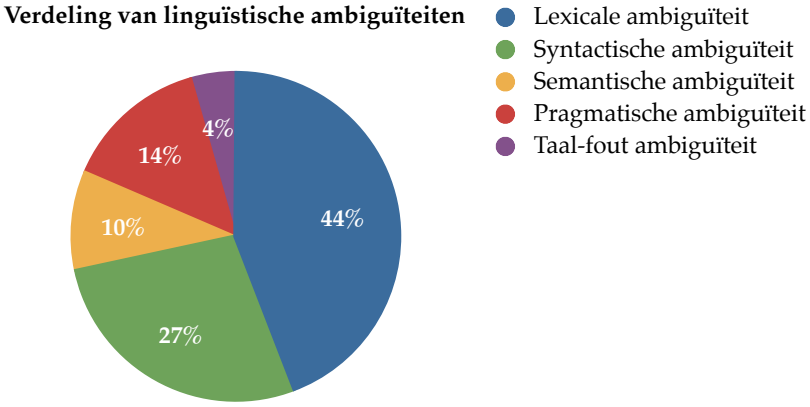


Diagram 5.2 Verdeling van de gevonden linguïstische ambiguïteiten.

In diagram 5.3 is te zien hoeveel linguïstische ambiguïteiten er gevonden zijn in de requirements-documenten. In het document van project 3 zijn de meeste linguïstische ambiguïteiten gevonden. In de diagram is de onderverdeling te zien tussen de typen linguïstische ambiguïteiten die gevonden zijn.

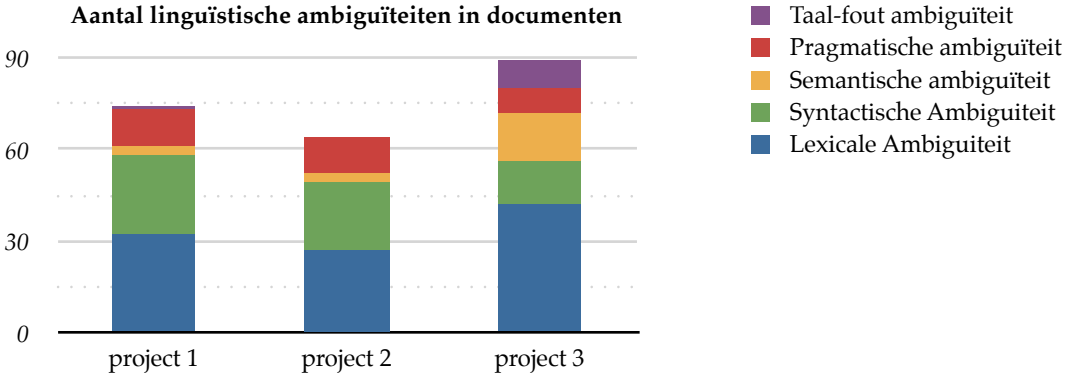


Diagram 5.3 Aantal gevonden linguïstische ambiguïteiten in requirementsdocumenten.

In tabel 5.6 zijn de absolute aantallen van de linguïstische ambiguïteiten die gevonden zijn te zien.

Ambiguïteit	**Project 1**	**Project 2**	**Project 3**	**Totaal**
Lexicale ambiguïteit	32	27	42	101
Syntactische ambiguïteit	26	22	14	62
Semantische ambiguïteit	3	3	16	22
Pragmatische ambiguïteit	12	12	8	32
Taalfout ambiguïteit	1	0	9	10
Totaal	74	64	89	227

Tabel 5.6 Aantal gevonden linguïstische ambiguïteiten per requirementsdocument.

In diagram 5.4 is de onderverdeling in linguïstische ambiguïteiten per project te zien. In deze diagram is te zien dat de verdeling in linguïstische ambiguïteiten ongeveer gelijk is in de documenten. Het document van project 1 en 2 hebben een bijna gelijke verdeling in linguïstische ambiguïteiten. Bij het document van project 3 zijn er meer semantische en taalfout ambiguïteiten gevonden in plaats van syntactische en pragmatische ambiguïteiten.

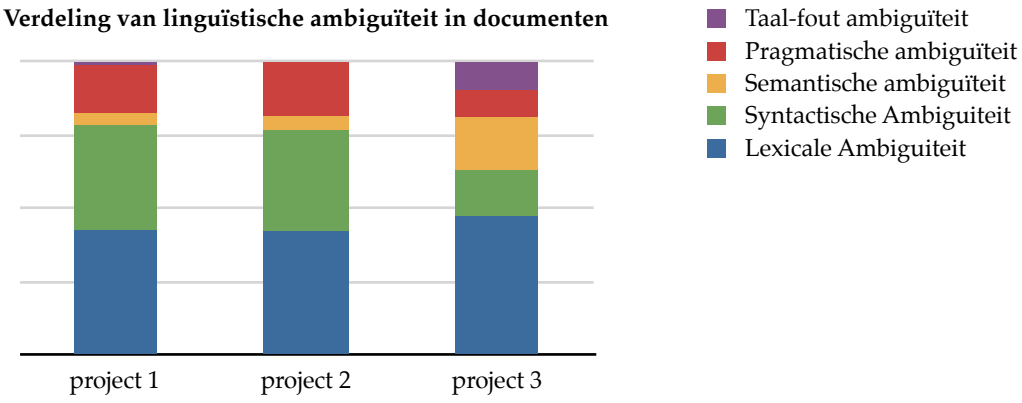


Diagram 5.4 Verdeling van linguïstische ambiguïteiten per requirementsdocument.

Lexicale ambiguïteit
Van de gevonden linguïstische ambiguïteiten is 44% (101 van 227) een lexicale ambiguïteit. In diagram 5.4 is te zien dat er per document ongeveer een gelijk aantal lexicale ambiguïteiten zijn gevonden.

In de IEEE Standaard 830 (1984) wordt geadviseerd om woorden of kenmerken die meerdere betekenissen hebben in het requirementsdocument op te nemen in een woordenlijst. Bij geen van de drie onderzochte documenten is een verklarende woordenlijst opgenomen. In één van de documenten

(project 3) is wel een lijst opgenomen waarin de betekenis van de gebruikte afkortingen is beschreven. Woorden of afkortingen die één betekenis hebben in het domein maar daarbuiten meerdere zijn aangemerkt als polysemie. Dit is gedaan omdat in het document niet wordt uitgelegd wat het woord of afkorting betekent.

Polysemie is het meest gevonden in de requirements. Van de gevonden lexicale ambiguïteiten is 88% (89 van 101) van de vorm polysemie. Van al de verschillende subtypen van linguïstische ambiguïteit is polysemie het meest gevonden. Van de linguïstische ambiguïteiten is 39% (89 van 226) van het subtype polysemie. In geen één van de drie documenten is een verklarende woordenlijst opgenomen. In de documenten zijn risicovolle en minder risicovolle vormen van polysemie gevonden. Bij een risicovolle polysemie kan er niet vanuit het domein worden achterhaald wat de betekenis van het woord is. Er worden enkele voorbeelden gegeven van vormen van polysemie die gevonden zijn.

Het volgende voorbeeld requirement bevat een risicovolle en een minder risicovolle vorm van polysemie.

> *"The EHR presents one form for all data entered during an exam." (Project 1)*

In het requirement kan het woord "*form*" de betekenis van "*vorm*" of van "*formulier*" hebben. Het woord "*exam*" kan de betekenis van *"examen | tentamen"* of van "*onderzoek*" hebben. Met een klein beetje domeinkennis van het project kan worden gezegd dat het eerste woord, *form*, een risicovolle ambiguïteit is en het tweede woord "*exam*" een minder risicovolle ambiguïteit. Het woord *form* kan in het domein meerdere betekenissen hebben. De betekenis van de requirement kan zijn *"het EHR toont één formulier voor al de data die ingevoerd is tijdens het onderzoek"* of *"het EHR toont de data die ingevoerd is tijdens het onderzoek in één vorm"*. Het woord "*exam*" kan de betekenis hebben van "*tentamen*" of van "*onderzoek*", omdat de softwareapplicatie voor oogartsen is, is het waarschijnlijk dat in deze requirement het woord "*exam*" de betekenis van "*onderzoek*" heeft. Met enkele domeinkennis kan worden gezegd dat een oogarts wel onderzoeken uitvoert, maar geen tentamens afneemt. Antropomorfisme is het toekennen van menselijke eigenschappen aan dingen zoals computers of softwareapplicaties. Antropomorfisme is een aantal keren gevonden in de requirements. In de requirements wordt antropomorfisme gebruikt in combinatie met spreektaal. In het volgende requirement is deze vorm gevonden.

> *"Op de peildatum dient het KIS dan te kijken naar het aantal DFZ-patiënten en de DIT opslag te berekenen door het aantal geïncludeerde DFZ-patiënten op peildatum te vermenigvuldigen met het opslag bedrag" (Project 2)*

Bij de requirement wordt gebruik gemaakt van het werkwoord kijken. Een computer of software-applicatie heeft geen menselijke zintuigen en kan dus niet kijken. Het gebruik van het werkwoord kijken maakt deze requirement dubbelzinnig en niet eenduidig. Er zijn in de requirements ook een aantal vormen van polysemie gevonden die aangemerkt kunnen worden als niet risicovol. Een voorbeeld hiervan is te zien in het volgende requirement:

> *"In order to make sure that the <project 3> system is maintainable and transparent, we propose to agree on a fixed set of concrete statuses that requests of all types can go through. This is also to avoid inconsistencies in user experience between for example a referral and an e-Consult." (Project 3)*

Het woord "*concrete*" kan de betekenis hebben van "*concreet*" of van "*beton*". Met een klein beetje kennis van het domein kan wel worden uitgesloten dat er in een medische softwareapplicatie statussen van beton voorkomen.

In de requirements is systematische polysemie minder vaak gevonden dan polysemie. Binnen de lexicale ambiguïteit is 12% (12 van 101) van de ambiguïteiten van de vorm systematische polysemie. Over het totaal is 5% (12 van 227) van de linguïstische ambiguïteiten van de vorm systematische polysemie.

In de documenten zijn een aantal vormen van systematische polysemie gevonden. In het volgende requirement is een vorm van systematische polysemie aanwezig.

> *"The frontdesk users will not have access to the patient medical data" (Project 1)*

Het is niet duidelijk of met *"patient medical data"* een persoonlijk record van de bewuste patiënt wordt bedoeld of dat het hier gaat over al de records van al de patiënten die in het systeem staan. De betekenis van dit zinsdeel is daarom verwarrend tussen de klasse en instantie van het woord. In veel gevallen waarbij het in de requirements over *"data"* gaat komt deze vorm van ambiguïteit voor. In de requirements is het dan niet duidelijk of persoonlijke of al de data records bedoeld worden. In het volgende voorbeeld zit ook een vorm van systematische polysemie.

> *"It would be nice if (default) filtering can be stored for a user to see only a subset of queues" (Project 3)*

In deze requirement is het woord *filtering* verwarrend. Het is niet duidelijk of met dit woord een product of proces wordt bedoeld. De betekenis van dit woord kan zijn dat het proces van filteren wordt opgeslagen. Dit kan gedaan worden door bijvoorbeeld de paramaters van de filter op te slaan. Het opslaan van het resultaat van de filtering kan ook een mogelijk betekenis van deze requirement zijn. Deze vorm van ambiguïteit komt ook geregeld voor bij requirements waarbij het verwarrend is of er met een woord een naam of een proces wordt bedoeld, voorbeelden hiervan zijn "de status afgekeurd", is de status afgekeurd of heeft iets de status van afgekeurd. "De knop verzenden", moet de knop worden verzonden of is de naam van de knop verzenden.

Syntactische ambiguïteit

In diagram 5.5 is de verdeling te zien van het aantal gevonden syntactische ambiguïteiten. Van de gevonden linguïstische ambiguïteiten is 27% (62 van 227) van het type syntactische ambiguïteit.

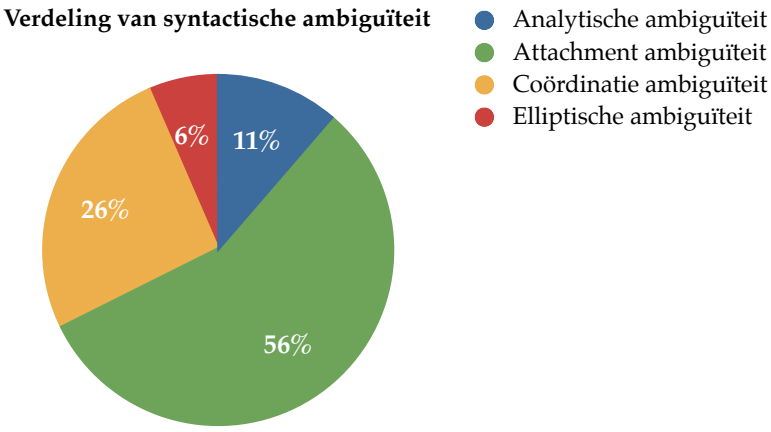


Diagram 5.5. Verdeling van gevonden syntactische ambiguïteiten.

In de requirements is analytische ambiguïteit niet zo vaak gevonden. Van de syntactische ambiguïteiten was 11% (7 van 62) van het subtype analytische ambiguïteit. Dit is 3% (7 van 227) van het totaal aantal linguïstische ambiguïteiten in de requirements.

In het volgende requirement staat een vorm van een analytische ambiguïteit.

> *"Indien status afgekeurd, in een tooltip de afkeur tekst tonen (tooltip verschijnt wanneer de muis over de tekst wordt bewogen)." (Project 2)*

Bij deze requirement is het verwarrend welke rol het woord *afgekeurd* heeft. Het woord *afgekeurd* kan een voltooid deelwoord zijn die iets zegt over de status, de status is niet meer geldig omdat deze is afgekeurd. Een andere betekenis is dat de status de waarde *afgekeurd* heeft, de requirement betekent dan: wanneer de waarde van de status is afgekeurd dan moet in de tooltip de afkeur tekst worden getoond.

In de documenten zijn een aantal requirements gevonden met het subtype attachment ambiguïteit. Een groot deel 56% (35 van 62) van de syntactische ambiguïteiten was van dit type ambiguïteit. Dit is 15% (35 van 227) van het totaal aantal linguïstische ambiguïteiten.

In het volgende voorbeeld komt een vorm van attachment ambiguïteit voor.

> *"The EHR presents one form for all data entered during an exam." (Project 1)*

Bij het requirement kan het woord *entered* bevestigd worden aan de twee zinsdelen: *"for all data entered"* of *"entered during an exam"*. De betekenis van de requirement is afhankelijk van het zinsdeel waar het woord *entered* aan wordt bevestigd. In het eerste voorbeeld is de betekenis van de requirement: al de data die in het EHR is ingevoerd wordt in één view getoond tijdens het onderzoek. In het tweede voorbeeld is de betekenis van de requirement: al de data die ingevoerd is tijdens het onderzoek wordt getoond in één view. Een ander voorbeeld van een attachment ambiguïteit is in de volgende requirement gevonden:

> *Deze export wordt dan door de software aangemaakt en aangeboden, welke daarna lokaal is op te slaan dan wel te printen. (Project 2)*

Het zinsdeel *welke daarna* kan in deze requirement bevestigd worden aan twee zinsdelen, dit zijn de zinsdelen *"deze export"* en *"de software"*. Met deze requirement kan worden bedoeld *"de export is lokaal op te slaan en te printen"* of *"de software is lokaal op te slaan en te printen"*.

In de documenten zijn een aantal ambiguïteiten gevonden van het subtype coördinatie ambiguïteit. Een deel 26% (16 van 62) van de syntactische ambiguïteiten was van dit type ambiguïteit. Dit is 7% (16 van 227) van het totaal aantal linguïstische ambiguïteiten. Yang et al. (2010) heeft onderzoek gedaan naar coördinatie ambiguïteit in requirements. In dit onderzoek heeft hij in 26829 zinnen uit 11 requirementsdocumenten naar coördinatie ambiguïteit gezocht. De verhouding van het aantal gevonden coördinatie ambiguïteiten van Yang komt ongeveer overeen met de resultaten van dit onderzoek. In 3404 (12,68%) van de zinnen die Yang onderzocht heeft is een coördinatie ambiguïteit gevonden. In dit onderzoek zijn in 16 (6,01%, in het onderzoek zijn geen requirements gevonden waarin meer dan één keer een coördinatie ambiguïteit voorkwam) van de 266 requirements coördinatie ambiguïteiten gevonden.

In het volgende voorbeeld komt een vorm van coördinatie ambiguïteit voor.

> *"The different tests and measurements done by the technician can be different in multiple practices." (Project 1)*

In deze requirement komt twee keer een coördinatie ambiguïteit voor. In de requirements is het is niet duidelijk of de beperkende bijwoorden "*different*" en "*done*" van toepassing zijn op beide woorden "*test*" en "*measurements*" of dat hier gelezen moet worden als "*different tests*" en "*measurements done by ...*". De betekenis van de requirement kan zijn *"The [different test] and [measurements done by the technician] can be different in multiple practices"* of *"The [different test done by the technician] and [different measurements done by the technician] can be different in multiple practices"*. In het volgende voorbeeld zit ook een vorm van attachment ambiguïteit.

> *"Members are external users of type Referring Physician or Support Staff." (Project 3)*

In deze requirement is het niet duidelijk of het beperkende bijwoord "type" van toepassing is op beide rollen "*Referring Physician*" en "*Support Staff*" of alleen op de rol "*Referring Physician*".

In de documenten zijn een aantal ambiguïteiten gevonden van het subtype elliptische ambiguïteit. Een klein deel 6% (4 van 62) van de syntactische ambiguïteiten was van dit type ambiguïteit. Dit is 2% (4 van 227) van het totaal aantal linguïstische ambiguïteiten.

In het volgende voorbeeld komt een vorm van elliptische ambiguïteit voor.

> *"The Mayo Temporary Access Admin will enter Date of Birth, Last name and MCN Number, and the MCN Number will be validated using the algorithm described in 5.1 Validate MCN number." (Project 3)*

Uit dit requirement valt niet af te leiden van wie "*Date of Birth, Last name and MCN numbe*r" zijn. Waarschijnlijk zijn deze eigenschappen van een patiënt, maar dit is niet in deze requirement opgenomen. Misschien is dit weggelaten omdat de auteur van de requirement ervan uitgaat dat de lezer van de requirement genoeg domeinkennis heeft om dit zelf in te vullen.

Semantische ambiguïteit
Van de gevonden linguïstische ambiguïteiten is 10% (22 van 227) van het type semantische ambiguïteit. Een subtype van semantische ambiguïteit is scope ambiguïteit. Deze vorm van ambiguïteit is niet gevonden in de requirements die onderzocht zijn.

Een *referentiële ambiguïteit* komt voor wanneer een verwijzing in de zin refereert naar meer dan één antecedent. Deze vorm van ambiguïteit komt voor binnen een zin of tussen de zin en een andere zin waarnaar verwezen wordt. In dit deel, bij semantische ambiguïteit, worden voorbeelden gegeven van requirements waarbij de referentie binnen de requirement niet duidelijk is.

In het volgende voorbeeld komt een vorm van referentiële ambiguïteit voor.

> *"By clicking on a problem a popup will show the possible treatments, after selecting a treatment or a free text description this treatment will be connected to the problem." (Project 1)*

In deze requirement is het niet duidelijk naar welke antecedent de anafoor *"this treatment"* verwijst. De verwijzing kan naar "*possible treatments*" zijn of naar "*a treatment*". In de volgende requirement is dezelfde ambiguïteit gevonden.

> *"If the MCN Number exists but has a different DOB and/or Last name, the service will return the information found, and this will be displayed to the user." (Project 3)*

Het woord *this* in deze requirement kan naar meerdere elementen verwijzen, dit kan zijn naar "*the information found"* of naar "*DOB and/or Last name"*.

Pragmatische ambiguïteit
Pragmatische ambiguïteit kan op meerder manieren voorkomen, een vorm hiervan is referentiële ambiguïteit in de context van de tekst. Van de gevonden linguïstische ambiguïteiten is 14% (32 van 227) het type pragmatische ambiguïteit.

Een *referentiële ambiguïteit* komt voor wanneer een verwijzing in de zin refereert naar meer dan één antecedent in de tekst. In het volgende voorbeeld staan twee requirements uit één requirementsdocument. Bij de laatste requirement is niet duidelijk waarnaar verwezen wordt.

> *"- The different tests and measurements done by the technician can be different in multiple practices.*
> *- The EHR will accommodate this in the exam form layout." (Project 1)*

Het woord *this* in de laatste requirement kan naar meerdere elementen verwijzen in de voorgaande requirement, dit kan zijn naar "*tests"* of naar "*measurements*".

Taalfout ambiguïteit
Van de gevonden linguïstische ambiguïteiten is 5% (10 van 227) van het type taalfout ambiguïteit. In het volgende voorbeeld komt een vorm van een taalfout ambiguïteit voor.

> *"The main users that have access to the EHR are the technician and the optometrist." (Project 1)*

In deze requirement wordt enkelvoud en meervoud door elkaar gebruikt. In de requirement staat dat de *main users* (meervoud) de *technician* (enkelvoud) en de *optometrist* (enkelvoud) zijn. Dit houdt in dat er in het totale systeem maar twee main users kunnen voorkomen een technician en een optometrist.

Oorzaken linguïstische ambiguïteit
Ambiguïteit in requirements heeft diverse oorzaken, per type ambiguïteit zijn er verschillende oorzaken aan te wijze.
Een lexicale ambiguïteit ontstaat wanneer een woord meer dan één betekenis heeft. Om lexicale ambiguïteit te voorkomen moeten de betekenissen van een woord worden ingeperkt. Dit kan gedaan worden door gebruik te maken van een verklarende woordenlijst. In de requirementsdocumenten zijn geen verklarende woordenlijsten opgenomen. In een verklarende woordenlijst wordt uitgelegd wat de betekenis van een woord of afkorting is. In de requirements worden er veel afkortingen gebruikt. In één van de requirementsdocumenten is hiervoor een verklarende lijst opgenomen, in de andere twee documenten niet. Voorbeelden van afkortingen die gebruikt worden zijn: CSV, HPI, RUN, HAP, enz.

De Bruijn en Dekkers (2010) hebben onderzoek gedaan naar ambiguïteit in requirements. Hierbij hebben zij verschillende factoren onderzocht. Eén van hun bevindingen is dat het aantal woorden

effect heeft op de ambiguïteit van een requirement. Hoe meer woorden een requirement heeft, hoe groter de kans is dat er een syntactische ambiguïteit in voorkomt. De Bruijn en Dekkers geven wel aan dat lange requirements zo lang zijn omdat deze vaak ook gecompliceerdere zaken moeten uitdrukken. In de requirementsdocumenten is te zien dat het aantal woorden in een requirement niet de oorzaak voor syntactische ambiguïteit is. In het document van project 2 wordt veel meer gebruik gemaakt van lange zinnen ten opzichte van de andere twee documenten. Verhoudingsgewijs komen er in het project 2 document niet meer syntactische ambiguïteiten voor dan in het project 1 document. Hierdoor kunnen lange zinnen niet worden aangemerkt als het enige risico voor syntactische ambiguïteiten.
In de requirements van project 2 is te zien dat lange zinnen de requirements wel extra vatbaar maken voor syntactische ambiguïteiten. Vooral de kans op attachment en coördinatie ambiguïteit is groot. In langere zinnen komen er meer zinsdelen voor. Bij het gebruik van voorzetsels of koppelwoorden is de kans groter dat deze bevestigd worden aan een verkeerd zinsdeel.
In de requirements komen syntactische ambiguïteiten ook meer voor bij requirements waarin meer dan één feit of beperking wordt beschreven. Bij meerdere feiten of beperkingen is het in één requirement niet altijd duidelijk waarover dit gaat. Als voorbeeld de volgende requirement.

> *"The summary overview will display the latest medical and most relevant information in a single view." (Project 1)*

In deze requirement staat meer dan één feit, dit zijn: *"The summary overview will display the latest medical"* en *"The summary overview will display the relevant information"*. Omdat er in deze zin twee feiten gebruikt worden is het door coördinatie ambiguïteit niet duidelijk of het laatste zinsdeel ("i*n a single view*") op beide feiten van toepassing is.

Bij requirements met veel woorden is de kans op referentiële ambiguïteit in een zin ook groter; vooral wanneer in een zin veel gebruik wordt gemaakt van tussenzinnen. Hoe meer zinsdelen of tussenzinnen hoe lastiger het is om te zien naar welk deel gerefereerd wordt. Als voorbeeld de volgende requirement.

> *"Een knop in de menubalk van waaruit de inhoud van deze view kan worden geëxporteerd naar Excel, met inbouw van filtermogelijkheid, waardoor gefilterde exports kunnen worden gemaakt (filters nog bepalen), export wordt dan door de software aangemaakt en aangeboden, welke daarna lokaal is op te slaan dan wel te printen." (Project 2)*

Het laatste woord "*welke"* kan in deze requirement verwijzen naar verschillen delen: een knop, een view, Excel, export of software. Door het opsplitsen van deze requirement in kortere zinnen kan dit voorkomen worden. De requirements worden in de documenten opgesteld in stukken tekst waarin meerdere requirements zijn verwerkt. Binnen in de stukken tekst wordt veel verwezen naar voorgaande zinnen. De kans op referentiële ambiguïteit is hierdoor groter.

Het aantal taalfout ambiguïteiten in de requirements is afhankelijk van de opsteller. Een opsteller maakt vaak dezelfde fouten in de requirements. In project 3 is het aantal taalfout ambiguïteiten groter dan in de andere documenten. De oorzaak hiervoor is dat de opsteller van dit document een aantal keer dezelfde fout heeft gemaakt. De opsteller heeft vaak enkelvoud en meervoud door elkaar gehaald.

Software engineerings ambiguïteit
In de 266 requirements zijn er 11 software engineerings ambiguïteiten gevonden. In het document van project 1 twee, in het document van project 2 twee en in het document van project 3 zeven. In de requirements zijn veel minder software engineerings ambiguïteiten gevonden dan linguïstische ambiguïteiten; dit heeft twee oorzaken. De eerste oorzaak is dat de ambiguïteit checklist meer geschikt is voor het vinden van linguïstische ambiguïteiten dan voor software engineerings ambiguïteiten. In de checklist zijn vijf typen linguïstische ambiguïteiten opgenomen en één type software engineerings ambiguïteit. De tweede oorzaak is gebrek aan domeinkennis. De personen die de requirements hebben gecontroleerd, hadden maar een beperkte kennis van het domein waarvoor de requirements zijn gemaakt.

In het volgende voorbeeld komt een vorm van een software engineerings ambiguïteit voor:

> *"... deze export wordt dan door de software aangemaakt en aangeboden, welke daarna lokaal is op te slaan dan wel te printen." (project 2)*

Het woord *software* kan binnen VitalHealth drie verschillende betekenissen hebben: het applicatiemodel, het platform van VitalHealth of een extern software component die in het platform is opgenomen. In deze requirement in het niet duidelijk welke betekenis het woord *software* heeft.

Implementatie onafhankelijk

In de drie onderzochte documenten komen requirements voor waarin ontwerp- en implementatiedetails zijn opgenomen. Het volgende requirement is hier een voorbeeld van.

> *"This portlet shows the appointment type and date for upcoming scheduled appointments" (Project 1)*

Bij dit requirement wordt verwezen naar een schermontwerp van een portlet. In dit schermontwerp is aangegeven waar knoppen en lijsten moeten worden geplaatst op het scherm. In het document wordt dit meerdere keren op dezelfde manier gedaan. Requirements worden ondersteunt door schermontwerpen waarin ontwerp- en implementatiebeslissingen zijn opgenomen.

In het document van project 2 zijn er ook geregeld ontwerp- en implementatiedetails opgenomen, een voorbeeld requirement is de volgende:

> *"Een knop in de menubalk van waaruit de inhoud van deze view kan worden geëxporteerd naar Excel, met inbouw van filtermogelijkheid, waardoor gefilterde exports kunnen worden gemaakt (filters nog bepalen), export wordt dan door de software aangemaakt en aangeboden, welke daarna lokaal is op te slaan dan wel te printen." (Project 2)*

In dit requirement zijn implementatiedetails verwerkt. Dit wordt gedaan door gebruik te maken van woorden als "knop", "menubalk", "view" en "Excel". Deze woorden duiden erop dat er al een beslissing is gemaakt in de requirement van hoe deze geïmplementeerd gaat worden.

Traceerbaar

De requirements die onderzocht zijn voldoen niet aan de richtlijnen die de IEEEstandaard 830 stelt voor traceerbare requirements. Een requirement die goed traceerbaar is moet zowel voorwaarts als achterwaarts traceerbaar zijn. De requirements zijn niet achterwaarts traceerbaar omdat per requirement niet inzichtelijk is wat de herkomst van de requirement is; de bron en de geschiedenis van de requirement worden niet beschreven.
In de onderzochte documenten is, bij requirements die relaties met andere requirements hebben, niet genoteerd of inzichtelijk gemaakt welke requirements gerelateerd zijn. Hierdoor zijn de requirements ook niet goed voorwaarts traceerbaar.

In de documenten worden de requirements opgesteld in stukken tekst. In één stuk tekst zijn meerdere requirements verwerkt. Een voorbeeld hiervan is het volgende stuk tekst:

> *"Each site also has an Appointment Office that processes the requests. Employees within this office are Appointment Coordinators. MCR has an International Appointment Coordinator that manages the international e-Consults. MCF and MCA have Transplant and International Appointment Coordinators who manage those requests separate from the general requests." (Project 3)*

In deze tekst staan verschillende zinnen waarin requirements zijn verwerkt. De requirements zijn niet apart aangegeven met namen of nummers, hierdoor zijn deze requirements niet goed traceerbaar.

5.1.3 *Interviews*

Bij vijf medewerkers van VitalHealth is een interview afgenomen. De interviews zijn volledig uitgeschreven en gecodeerd. De gecodeerde interviews zijn te vinden in bijlage B interviews. De respondenten zijn werkzaam bij één of meer productlijnen van VitalHealth. In tabel 5.7 is te zien in welke rollen en productlijnen de respondenten werkzaam zijn.

Respondent	**Productlijn**	**Rol**
Respondent 1 (r1)	*EHR*	*Requirements engineer*
Respondent 2 (r2)	*Patiëntenportalen en maatwerkproject*	*Req. engineer en applicatieontwikkelaar*
Respondent 3 (r3)	*KIS*	*Applicatieontwikkelaar*
Respondent 4 (r4)	*EHR en patiëntenportalen*	*Applicatieontwikkelaar*
Respondent 5 (r5)	*QuestManager, EHR en maatwerkproject*	*Applicatietester*

Tabel 5.5 Overzicht van de respondenten die geïnterviewd zijn.

Twee respondenten hebben de rol van requirements engineer. Eén van de requirements engineers (r2) heeft ook de rol van applicatieontwikkelaar, de respondent werkt ook mee aan de ontwikkeling van de softwareapplicatie. Twee respondenten zijn werkzaam als applicatieontwikkelaar, de laatste respondent is werkzaam als applicatietester.

Requirements engineerings proces

Bij VitalHealth worden de wensen en eisen van de stakeholder op verschillende manieren verzameld. In de meeste projecten wordt dit gedaan door gesprekken met de stakeholders. In sommige projecten heeft de stakeholder zelf al zijn wensen en eisen verzamelt in een document.

> *"...hebben we wel een soort high level requirementsdocument gekregen van <stakeholder> zelf van we willen een nieuw systeem waarmee we patiënten kunnen doorverwijzen naar <stakeholder> vanuit huisartsen. En dat moet voldoen aan een heleboel aspecten, dat was een enorme lijst...." (r2)*

In een aantal projecten zijn de stakeholders al werkzaam met de softwareapplicatie. Voor een nieuwe versie van de applicatie hebben de stakeholders dan ook weer nieuwe wensen. De nieuwe wensen kunnen de stakeholders invoeren in het supportsysteem van VitalHealth. Het supportsysteem wordt gebruikt om wensen en fouten van een softwareapplicatie te registreren en als communicatiemiddel tussen de requirements engineer en de applicatieontwikkelaar. De nieuwe wensen in het supportsysteem worden door de productmanager omgezet naar requirements.

> *"...klanten loggen zelf ook wensen en dan wordt het eerst een ticket, klanten komen eerst bij ons een ticket loggen, die wordt automatische omgezet naar een issue en <productmanager> die beoordeelt al de wensen van de klanten." (r3)*

De requirements worden op verschillende manieren aangeleverd bij de applicatieontwikkelaar. Het requirementsdocument is de meest gebruikte vorm hiervoor. Email en het VitalHealth supportsysteem worden ook gebruikt om de requirements aan te leveren.

> *"...soms is dat gewoon een korte mail of zo of een Word document. Het ligt dus wel vast, negen van de tien keer staat het vast. En we werken ook veel met issues, dan hebben we een issue en dan staat daar het hele stuk in en dan heb ik een vraag en dan reageert <productmanager> er weer op." (r3)*

Kwaliteit van de requirements

Er zijn vijf kwaliteitscriteria (eenduidig, consistent, traceerbaar, verifieerbaar en implementatie onafhankelijk) beschreven waaraan een requirementsdocument moet voldoen. In deze paragraaf is per criteria beschreven of de respondenten vinden dat de requirements aan het criteria voldoen. In tabel 5.8 is een overzicht hiervan te zien; er staat per criteria of de requirements hier wel of niet aan voldoen volgens de respondenten. De criteria traceerbaar is niet door elke respondent genoemd.

Criteria	**Wel**	**Niet**
Eenduidig	*0*	*5*
Consistent	*2*	*3*
Implementatie onafhankelijk	*0*	*5*
Traceerbaar	*2*	*0*
Verifieerbaar	*4*	*1*

Tabel 5.8 Kwaliteitscriteria waaraan de requirements volgens de respondenten wel of niet voldoen.

Eenduidig
De vijf respondenten vinden allemaal dat de requirementsdocumenten van VitalHealth niet eenduidig zijn. De meningen van de respondenten komen overeen met wat geconstateerd is in het requirementsdocumenten onderzoek.
Woorden die dubbelzinnig zijn kunnen op een verkeerde manier geïnterpreteerd worden. In tabel 5.9 is te zien hoe vaak de respondenten in de praktijk meemaken dat een woord in een requirement op een verkeerde manier wordt geïnterpreteerd. In de tabel is het onderscheid te zien tussen interpretatieproblemen bij stakeholders en bij applicatieontwikkelaars. Alleen de requirements engineers hebben de interpretatieproblemen bij de stakeholders genoemd.

	Wel	**Soms**	**Niet**
Interpretatieproblemen stakeholders	*1*		*1*
Interpretatieproblemen ontwikkelaars	*2*	*2*	*1*

Tabel 5.9 Interpretatieproblemen bij woorden in de requirements door stakeholders en applicatieontwikkelaars.

De requirements engineers hebben een verschillende mening over interpretatieproblemen bij de stakeholders. Eén van de requirements engineer (r2) komt dit wel tegen en de andere requirements engineer (r1) komt dit niet tegen in de praktijk. Aan de respondent (r1) worden nooit vragen over woorden in de requirements gesteld door de stakeholders. Hij geeft als reden hiervoor dat de stakeholders de requirements slecht lezen en het door hun kennisgebrek komt wanneer ze een woord niet begrijpen.

> *"....maar ik denk dat dat komt doordat ze slecht lezen en alles wat ze niet snappen dan denken ze dat het aan hun ligt wanneer ze het niet snappen." (r1)*

De andere respondent (r2) komt het wel tegen dat woorden verkeerd begrepen worden. Om dit te voorkomen maakt hij in de requirements gebruik van woorden uit het domein van de stakeholders.

> *"...je neemt heel gauw de begrippen van die klant aan. Wij praten bijvoorbeeld in ons patiënten portaal over zorgverleners en bij de klant hebben ze het over healthcare professionals dat zijn ook wel zorgverleners maar die begrippen neem je dan over in je documenten omdat dat de kreten zijn die zij zo goed kennen." (r2)*

Uit de reacties van de respondenten blijkt dat het in de praktijk voorkomt dat stakeholders woorden niet begrijpen of verkeerd interpreteren.

Ook over het interpretatieprobleem bij applicatieontwikkelaars zijn de meningen van de requirements engineers verschillend. De requirements engineer (r1) die geen vragen van stakeholders krijgt geeft aan deze ook niet van de applicatieontwikkelaars te krijgen. De andere requirements engineer (r2) krijgt wel vragen van applicatieontwikkelaars over woorden.

> *"Ja, niet alleen de begrippen maar ook requirements en ook de oplossingsrichting." (r2)*

De mening van de applicatieontwikkelaars is ook verschillend over dit onderwerp. Eén applicatieontwikkelaar (r3) komt heel zelden woorden tegen die hij niet begrijpt terwijl dit bij de andere applicatieontwikkelaar (r4) wel vaak voorkomt. Een citaat van de applicatieontwikkelaar:

> *"Nee, heel soms iets en heel zelden als ik een medische term niet weet. Meer van een bepaalde ziekte..." (r3)*

En een citaat van de andere applicatieontwikkelaar die wel vaker woorden tegenkomt die hij niet begrijpt. De respondent geeft aan dat de oorzaak vaak de summiere beschrijving van de requirement is.

> *"Ja, dat is vaak wel zo en dan moet je gokken aan de hand van het voorgaande of het nakomende wat het dan is. Maar vaak is dat nog niet eens het probleem, vaak is het de summiere beschrijving van het probleem." (r4)*

De applicatietester (r5) komt *"niet echt"* woorden in de requirements tegen die hij niet begrijpt. Uit de reactie van de respondenten die wel woorden tegenkomen die niet begrepen worden blijkt de oorzaak vaak de beschrijving van de requirement te zijn. De beschrijving is niet afdoende om woorden te begrijpen en goed te interpreteren.

In de requirementsdocumenten die onderzocht zijn komen geen verklarende woordenlijst voor. In tabel 5.10 is te zien hoe vaak respondenten in de praktijk verklarende woordenlijsten tegenkomen in requirementsdocumenten.

	Wel	**Soms**	**Niet**
Verklarende woordenlijsten in documenten	*2*	*1*	*2*

Tabel 5.10 Aantal respondenten die wel, soms of niet verklarende woordenlijsten tegenkomen in requirementsdocumenten.

De requirements engineers geven aan dat verklarende woordenlijsten alleen worden opgenomen in de hoofd requirementsdocumenten van een project. In de documenten waarin requirements voor deelcomponenten worden opgenomen staan geen woordenlijsten.

> *"...die hebben natuurlijk wel relaties met elkaar. En als je dan één zo'n los ontwerp eruit trekt dan zie je veel begrippen en die kom je dan tegen in andere documenten daar worden die dan uitgelegd." (r2)*

De applicatieontwikkelaars (r3 en r4) zien soms of nooit verklarende woordenlijsten in de documenten. Ook de applicatietester (r5) is nog nooit een verklarende woordenlijst tegengekomen in een requirementsdocument.
De conclusie is dat de verklarende woordenlijsten worden opgenomen in de hoofd-requirementsdocumenten. De respondenten die geen verklarende woordenlijsten tegenkomen maken waarschijnlijk geen gebruik van de hoofddocumenten.

In de requirementsdocumenten zijn veel requirements gevonden die ambigu zijn. De respondenten geven aan dat zij requirements tegenkomen die op meerdere manieren geïnterpreteerd kunnen worden. De requirements engineers bevestigen dit sterker dan de applicatieontwikkelaars. De reactie van de requirements engineers is *"Oh ja zat" (r1)* en *"Ja, dat zeker" (r2)*. De reactie van de ontwikkelaars is *"Ja, dat is wel eens ja" (r3)* en *"Ja soms heb je dat wel ja" (r4)*.

Wanneer de applicatieontwikkelaars requirements tegenkomen die dubbelzinnig zijn dan gaan ze terug naar de opsteller van de requirement en vragen om uitleg.

> *"Als ik even een bepaalde requirement niet begrijp dan vraag ik dat direct" (r3)*

Een requirement wordt niet altijd aangepast wanneer een applicatieontwikkelaar of een applicatietester aangeeft dat deze dubbelzinnig is. Eén respondent (r3) geeft aan dat het weleens voorkomt dat de requirements zijn aangepast. Ook de applicatietester (r5) komt requirements tegen die ambigu zijn, maar de requirements worden niet aangepast wanneer hij aangeeft dat deze ambigu zijn:

> *"Het requirement is zoals het is. Het requirement blijft staan." (r5).*

De applicatietester komt requirements tegen waarvan de applicatieontwikkelaar al bij de opsteller heeft aangegeven dat deze dubbelzinnig zijn.

> *"Maar ja het komt ook weleens voor dat de ontwikkelaar met dezelfde punten zat maar daarna is niet het requirements aangepast. Dat blijft zoals het afgesproken is zeker vanuit de klant." (r5)*

De respondenten noemen diverse oorzaken voor het verkeerd interpreteren of begrijpen van requirements. In tabel 5.11 is per rol te zien welke oorzaken de respondenten genoemd hebben. Gebrek aan domeinkennis wordt het meest genoemd.

Oorzaak	**RE**	**Ontw.**	**Tester**	**Totaal**
Gebrek aan domeinkennis	*1*	*1*	*1*	*3*
Verkeerde zinsstructuur	*1*	*0*	*1*	*2*
Incomplete requirements	*0*	*1*	*1*	*2*
Geen inzicht in de werking van de applicatie	*1*	*0*	*0*	*1*
Slecht lezen van requirements	*1*	*0*	*0*	*1*

Tabel 5.11 Oorzaken van interpretatieproblemen bij requirements.

Een requirements engineer (r2) noemt het gebrek aan domeinkennis als oorzaak waardoor requirements niet goed begrepen worden.

"De ontwikkelaar moet dan kennis van het domein hebben? Ja, en daarom ontstaan er best wel eens misverstanden." (r2)

Eén requirements engineer (r2) geeft aan dat hij bij het opstellen van de requirements snel gebruik maakt van domeinwoorden.

"...je neemt heel gauw de begrippen van die klant aan. Wij praten bijvoorbeeld in ons patiënten portaal over zorgverleners en bij de klant hebben ze het over healthcare professionals dat zijn ook wel zorgverleners maar die begrippen neem je dan over in je documenten omdat dat de kreten zijn die zij zo goed kennen." (r2)

Om domeinkennis over te brengen op de applicatieontwikkelaar maken beide requirements engineers gebruik van mondelinge overdracht:

"....met de woordenlijst probeer je een beetje de domeinkennis die jij hebt over te brengen naar de ontwikkelaars toe? Ja maar heel veel gaat ook mondeling." (r1)

De applicatieontwikkelaars (r3 en r4) geven aan dat zij ook vaak contact met de requirements engineers hebben over domeinkennis: *"Dat vraag ik de consultant zelf" (r3).* Het overdragen van domeinkennis gaat mondeling, via documenten of per mail. Eén respondent geeft aan dat hij soms ook contact met een stakeholder heeft. Hij doet dit omdat de requirements engineer dan niet voldoende domeinkennis heeft.

"...als ik iets tegenkom dan vraag ik het eerst aan een consultant en soms weet een consultant het ook niet dan bel ik zelf even de klant op." (r3)

De applicatietester heeft voordat hij gaat testen contact met de applicatieontwikkelaar die het te testen deel heeft ontwikkeld. De requirements die hij test bespreekt hij met de applicatieontwikkelaar en de domeinkennis die hij mist vraagt hij aan de applicatieontwikkelaar.

"...de nieuwe zijn dan de requirements, en dan ga je echt één voor één, want het zijn echt verschillende ontwikkelaars en dan weet je wie hem opgepakt heeft gelukkig en dan ga je gewoon even zitten." (r5)

De applicatietester (r5) noemt linguïstische ambiguïteit als één van de oorzaken voor dubbelzinnige requirements. Hij noemt dat de requirements vaak taalkundig op meerdere manieren te interpreteren zijn.

"Hoe komt het dan dat je het niet begrijpt, is het omdat het een dubbele betekenis heeft in het domein of dat het taalkundig dubbelzinnig is? Taalkundig. Dat het niet netjes is." (r5)

De requirements engineers (r2) vindt dit een lastig punt. Hij geeft aan dat het uitmaakt hoe een requirement wordt opgeschreven. Hij is er zich van bewust dat een requirement altijd anders kan worden geïnterpreteerd.

"....want hoe je het ook opschrijft, dat maakt wel uit hoe je het opschrijft, er zijn altijd wel het kan altijd anders geïnterpreteerd worden..." (r2)

Twee respondenten vinden incomplete requirements één van de oorzaken voor dubbelzinnige requirements. De applicatieontwikkelaar benoemt dit op de volgende manier:

"Maar vaak is dat nog niet eens het probleem, vaak is de summiere beschrijving het probleem." (r4)

Ook de applicatietester komt requirements tegen die summier beschreven zijn. Om deze requirements te begrijpen heeft hij domeinkennis van de opsteller of ontwikkelaar nodig.

Ja klopt, omdat je key-tester bent ken je het domein behoorlijk alleen dan net dat stukje, in één twee zinnen kun je niet de lading overbrengen wat er overgedragen is aan de developer. (r5)

Uit de bovenstaande citaten blijkt dat er een relatie is tussen incomplete requirements en het gebrek aan domeinkennis. Incomplete requirements kunnen er de oorzaak van zijn dat de lezer van een requirement onvoldoende domeinkennis krijgt.

Eén van de requirements engineers (r1) noemt nog twee andere oorzaken waardoor requirements niet goed begrepen worden. Het eerste wat hij noemt is dat stakeholders en applicatieontwikkelaars de requirements slecht lezen.

"Ze krijgen ze wel maar mijn ervaring is dat ze het slecht lezen." (r1)
"Tekst gaan ontwikkelaars niet zitten lezen, die lezen zo slecht..." (r1)

Het tweede wat hij noemt is gebrek aan inzicht van een ontwikkelaar om te begrijpen wat er met een requirement bedoeld wordt.

"Kijk dat kan iedere ontwikkelaar snappen maar als je dan hier bijvoorbeeld een lijst met achthonderd waardes gaat zien, dan moet je als ontwikkelaar gaan snappen van dit gaat niet werken." (r1)

De respondenten zien dat de kwaliteit van de requirements invloed heeft op het software-ontwikkelproces. De requirements engineer (r2) geeft hier een voorbeeld van, hij noemt requirements die ambigu en summier beschreven zijn die gebruikt worden voor contracten met klanten.

"Dan sturen we een offerte naar de klant en dan staat dan één zinnentje die is gewoon voor heel veel uitleg vatbaar. Dat is en daar ontstaan de discussiepunten en dat is niet nodig als je drie zinnen extra eraan wijdt. Het is zo belangrijk dat je in jou communicatie naar klanten gewoon duidelijk zegt wat je doet." (r2)

Het "over-de-muur-gooi" principe van Rawas en Easterbrook (1996) (dit principe is beschreven in hoofdstuk 3) komt de requirements engineer ook tegen.

"...maar dat komt met name voor omdat die requirements zijn verzameld door een functioneel consultant vaak met de opdrachtgever en die worden op een vrij hoog niveau bedacht. En wat er dan gebeurde dat ze die doorstuurde naar de ontwikkelaar op de afdeling in India. Als je dat op een hoog niveau laat staan en je kan zelf al vijfentwintig vragen stellen, dan moet je niet denken dat je iets heel consistents terug krijgt." (r2)

De applicatietester ziet ook requirements voorbij komen die ambigu zijn en al in de softwareapplicatie zijn geïmplementeerd. De applicatieontwikkelaar heeft de ambiguïteit dan niet opgemerkt en in de softwareapplicatie ingebouwd.

"Ja, dat is.... eigelijk zou je dat al aan de voorkant moeten zien wanneer een ontwikkelaar daar mee aan de gang gaat en die ziet van ik kan dit op die manier interpreteren of op die manier of weer op een andere manier. Wanneer hij dat niet ziet dan loop ik er wel tegenaan dus eigenlijk een dubbel check. Maar eigenlijk zou ik daar dan niet meer tegenaan moeten lopen." (r5)

Er worden geen activiteiten ondernomen om ambiguïteit in de requirements te voorkomen. Door een respondent (r1) wordt hiervoor de volgende reden gegeven.

"Nee, ik steek het liefst zo veel mogelijk energie in een mondelinge uitleg en het tijdens het proces in de gaten houden en achter af weer, maar dat is meer omdat het efficiënter is in ons team." (r1)

De applicatieontwikkelaar (r3) communiceert veel met de requirements engineers om te voorkomen dat hij requirements op een verkeerde manier interpreteert. In een overleg neemt hij samen met de requirements engineer de requirements stap voor stap door. Daarna bouwt hij een stukje van de applicatie, deze wordt dan getest door de requirements engineer.

Het gebruik van een notatietechniek voor de requirements wordt door de respondenten niet genoemd als oorzaak van dubbelzinnige requirements. Er wordt ook weinig gebruik gemaakt van notatietechnieken, maar één respondent geeft aan gebruik te maken van use cases bij het opstellen van de requirements.

De meeste respondenten vinden eenduidigheid een belangrijk criterium voor de requirements. Eén van de respondenten geeft aan eenduidigheid onbelangrijk te vinden voor de requirements.

"En deze denk ik als minste onderaan omdat, dat doel haal je toch nooit, dus doe het gewoon zoals je..... Want dan krijg je weer tegenstrijdige belangen want dan krijg je een heel uitgebreid document maar dan leest weer niemand het. Dus dan kan het wel heel eenduidig zijn maar wordt niet gebruikt." (R1)

De respondent is van mening dat het doel van eenduidige requirements niet haalbaar is, tenzij de requirements heel uitgebreid worden opgesteld in het document. De mening van deze respondent komt overeen met wat Mullery schrijft; het maken van een SRS waarin geen ambiguïteiten voorkomen kost veel tijd. De tijd die dit kost moet wel in overeenkomst zijn met de totale tijd die besteed wordt aan de ontwikkeling van de softwareapplicatie.

Implementatie onafhankelijk

De respondenten geven aan dat de requirementsdocumenten van VitalHealth niet implementatie onafhankelijk worden opgesteld. In het requirementsdocumenten onderzoek kwam naar voren dat er vaak geen onderscheid wordt gemaakt tussen het requirementsdocument en het functioneel ontwerp. In een project wordt meestal een combinatie van requirements en ontwerp in één document opgesteld. Een requirements engineer beschrijft op de volgende manier hoe hij zijn document indeelt:

> *"Kijk in mijn ontwerp-document heb ik die twee hoofdstukken nu dus zitten. In het ene hoofdstuk is echt requirements daarin probeer ik echt te focussen op requirements. En in het andere hoofdstuk de implementatie. Ik zou zeggen de requirements moeten eigenlijk zo ontwerp onafhankelijk mogelijk zijn en in het tweede hoofdstuk niet" (r1)*

De applicatietester (r5) geeft aan dat de ontwerp-documenten vaak een combinatie van ontwerp en requirements zijn.

> *"Het is binnen VitalHealth vaak een combi tussen functioneel ontwerp en requirements. Het is niet echt dat er een scheiding gemaakt wordt tussen beide." (r5)*

De requirements van VitalHealth worden veelal procedureel opgesteld. In tabel 5.12 is te zien welke vorm de voor- of afkeur heeft van de respondenten. De meeste respondenten hebben een voorkeur voor het procedureel opstellen van de requirements.

Procedureel	
Voorkeur	*3*
Afkeur	*1*
Declaratief	
Voorkeur	*2*
Afkeur	*0*

Tabel 5.12 De voorkeur van het aantal respondenten voor procedureel of declaratief.

Een requirements engineer geeft de volgende reden waarom procedureel zijn voorkeur heeft.

> *"Nou omdat je bij requirements moet je weten van welke requirement wordt door wie op welk moment gebruikt en is door wie op welk moment nodig. Als je dat niet weet dan krijg je per definitie geen goeie user interface design." (r1)*

De requirements engineer wil weten hoe de requirements werken zodat hij in het user interface design de interactie tussen de requirements kan beschrijven. In deze mening komt naar voren dat er geen duidelijk verschil wordt gemaakt tussen requirements en het functioneel ontwerp. De requirements engineer wil in de requirements het "hoe" en het "wat" van de softwareapplicatie beschrijven. Dit blijkt ook uit de moeilijkheden waar hij tegenaan loopt bij het procedureel beschrijven van de requirements.

> *"Het nadeel ervan vind ik wel is dat je, kijk als je bepaalde functie hebt, ik heb hier ook best wel lang mee zitten strubbelen met de structuur van mijn design document, mijn initiële design document. Want als je een functie beschrijft die heeft op heel veel plekken heeft die raakvlakken.." (r1)*

Door het "hoe" en het "wat" in de requirements te beschrijven krijgt de requirements problemen met het overzichtelijk opstellen van zijn document.

Een applicatieontwikkelaar (r4) geeft op de volgende manier aan waarom hij de voorkeur heeft voor het declaratief opstellen van requirements:

> *"Ja niet die tweede, die eerste* (declaratief) *voornamelijk. Zo wil ik het graag, ik wil weten hoe het systeem zou moeten werken en de oplossing daar mogen we zelf over nadenken daarvoor zijn wij technische ontwikkelaars in principe. Dat wij iets technische gaan maken." (r4)*

Het scheiden van de rollen binnen het softwareproject vindt deze respondent belangrijk. Dit komt overeen met wat Kulak en Guiney (2004) vinden; in het software-ontwikkelproces moeten personen activiteiten uitvoeren die past bij de rol en vaardigheden die zij hebben. De personen die de requirements verzamelen kunnen vaardigheden hebben die niet geschikt zijn om ontwerp en implementatie beslissingen te maken. Dit komt ook goed naar voren in wat een applicatieontwikkelaar (r3) aangeeft over het opstellen van de requirements:

"Als de <productmanager> op een gegeven moment met het technische er zelf niet uitkomt dan beschrijft hij wel meer het proces, hoe het dan moet gebeuren en dan lichten wij het technisch toe. Van jongens we gaan het zo bouwen. Soms weet een consultant niet hoe technisch iets gemaakt moet worden. Dan hebben ze wel ongeveer een idee komt hij bij ons en dan gaan wij overleggen en dan gaan we een beetje brainstormen van welke gedachte welke kant gaan we op zeg maar." (r3)

Een requirements engineer zonder een technische achtergrond is betrokken bij het ontwerp proces van de softwareapplicatie. In de requirements wil deze requirements engineer implementatie beslissingen maken. De andere requirements engineer (r2) die geïnterviewd is geeft de voorkeur aan het declaratief opstellen van de requirements. Deze respondent geeft hier de volgende reden voor:

"Ik denk dat je altijd moet beginnen met wat wil die klant, wat is eigenlijk het probleem. Wat gaan we oplossen wat is eigenlijk het issue wat is de business case die er ligt." (r2)

Deze requirements engineer vindt het nodig om te weten wat het probleem is van de stakeholder voordat hij de softwareapplicatie kan gaan ontwikkelen. Deze mening komt overeen met wat Kulak en Guiney (2004) aangeven. De oplossing van het probleem moet komen nadat het probleem is geïdentificeerd, begrepen, gedocumenteerd en overeengekomen met de stakeholder.

Traceerbaar

Twee respondenten geven aan dat de requirements van VitalHealth goed traceerbaar zijn. De andere drie respondenten hebben niets gezegd over de traceerbaarheid van de requirements. De respondenten geven aan dat er wel relaties zijn tussen de requirements, maar dat deze niet altijd expliciet zijn gedocumenteerd.

"Ja, dat zou ik zeggen van wel, inderdaad maar dat staat niet in de requirements beschreven. Er is wel een onderlinge relatie tussen de requirements, ik denk dat deze wel in de requirementsdocumenten zit." (r4)

In de requirementsdocumenten die onderzocht zijn worden er geen unieke namen of nummers aan de requirements gegeven. In de requirementsdocumenten zijn de requirements opgesteld in een beschrijvend stukje tekst. In tabel 5.13 is te zien aan welk formaat de respondenten de voorkeur geven.

	RE	**Ontw.**	**Tester**	**Totaal**
Tekst	*0*	*0*	*0*	*0*
Lijst	*1*	*2*	*1*	*4*

Tabel 5.13 Verdeling van het aantal respondenten met voorkeur voor het opstellen van requirements in tekst of in een lijst.

Eén respondent (r2) had geen duidelijke voorkeur. De andere vier respondenten hebben de voorkeur voor het opstellen van requirements in een lijst. De meeste respondenten geven aan dat een lijst met daarbij een proiritering van de requirement hun voorkeur heeft. De requirements engineer verwoord zijn voorkeur op de volgende manier:

"Tekst gaan ontwikkelaars niet zitten lezen, die lezen zo slecht dus wat dat betreft is een lijstje waar ze een soort checklist aan het einde hebben van requirements heeft wel een voordeel." (r1)

De applicatietester heeft als voorkeur van een combinatie van beide. Een lijst waarin de requirements zijn opgesomd met daarbij in tekst een uitleg van de requirements.

"Daarom die combinatie van dat je ze eerst opsomt en vervolgens de uitleg erbij ja dan zie je van dit is daar verwerkt en dit is daar verwerkt zo in je stukje uitleg zie je dat meteen terug. Dit heeft zeker mijn voorkeur want dan kan je het ook minder snel op een andere manier interpreteren." (r5)

Verifieerbaar

Vier van de vijf respondenten vinden de requirements goed verifieerbaar. De applicatietester kan het best beoordelen of de requirements verifieerbaar zijn. Deze persoon test de applicaties op basis van de requirements. Hij vindt de requirements goed verifieerbaar. De applicatietester geeft wel aan dat het test proces kan versnellen wanneer de requirements kwaliteit van de requirements beter wordt.

"En gelukkig hebben we korte lijntjes. Verifieerbaar is het wel degelijk. Het scheelt tijd wanneer het vooraf al helder is." (r5)

Eén requirements engineer vindt het lastig om requirements te maken die goed verifieerbaar zijn. Vooral bij complexe applicaties vindt hij het moeilijk om de requirements verifieerbaar te definiëren.

> *"Want de applicaties zijn dermate complex aan de applicaties zelf kan je die requirements moeilijk terug vinden. Dit is best een moeilijke vraag. Bijvoorbeeld als je een heel ingewikkeld autorisatie model hebt en je hebt dat geïmplementeerd en je hebt dat laten ontwikkelen hoe kan je dan heel snel in zo'n applicatie zien dat dat erin zit. Dat die requirements zijn opgelost." (r2)*

Nadat de softwareapplicatie gebouwd is wordt deze getest op basis van de requirements. Dit vormt de acceptatie voor de stakeholder. In de softwareapplicatie wordt getest of de wensen en eisen van de stakeholder in de applicatie zijn verwerkt. De requirements engineer geeft aan dat er geen duidelijk proces is voor de acceptatie testen:

> *"Maar hoe controleer je het, je moet gewoon de klant laten tekenen. De klant moet dan bij de acceptatie weer tekenen of het klopt wat hij in zijn requirements heeft getekend." (r2)*

De applicatieontwikkelaars geven aan dat de applicaties die zij bouwen getest worden door de requirements engineers. De requirements engineer controleert of al de requirements zijn verwerkt in de softwareapplicatie.

> *"Het wordt eigenlijk twee keer gecheckt. Als ontwikkelaar vergeet je ook wel eens wat, dan komt de consultant van dat moest er nog in. Of soms heb je het niet helemaal begrepen en komt er nog een requirement bij." (r3)*

Consistent

Twee respondenten vinden de requirements consistent en drie respondenten niet. De beide requirements engineers (r1 en r2) vinden dat inconsistentie in de requirements vooral wordt veroorzaakt door conflicterende wensen van de stakeholders.

> *"...in die grote lijst waar ik het net over had daar stonden echt dingen in die met elkaar conflicteerde. En dat ziet de klant ook niet altijd, dan ga je in gesprek en dan komt hij ook totdat inzicht." (r2)*

Requirements engineer r1 kiest ervoor om conflicterende wensen van de stakeholders wel in de requirements op te nemen, in het functioneel ontwerp maakt hij pas de keuze voor één van de wensen. De andere requirements engineer (r2) probeert conflicterende requirements al in de analyse fase te voorkomen, maar hij denkt dat er tijdens de ontwikkelfase nog wel conflicterende requirements in de documenten staan.

De applicatieontwikkelaars denken verschillende over de consistentie van de requirements. De applicatietester loopt niet zo vaak tegen conflicterende requirements aan. Wel ziet hij dat er af en toe een requirement conflicteert met een wijziging die een stakeholder al veel eerder heeft ingediend.

<u>Eigenschappen gecontroleerde taal</u>

Om een gecontroleerde taal te gebruiken moet er door de betrokkenen van de requirements tijd worden geïnvesteerd om deze te leren. De requirements engineers zijn bereid om tijd te investeren in het leren van een gecontroleerde taal.

> *"Als dat deze problemen oplost, dan is dat goed. Dan denk ik dat ik daar wel voor opensta. Want wij lopen hier vaak tegenaan in de requirements." (r2)*

De applicatieontwikkelaars zijn ook bereid om tijd te investeren om een gecontroleerde taal te leren. De ontwikkelaars stellen zelf niet veel requirements op maar zij geven aan dat wanneer een requirements engineer hier gebruik van zal maken dit ook voordelen heeft voor hun.

> *"Ja daar zal ik zeker wel tijd in willen investeren, dit vind ik wel belangrijk. Niet dat ik altijd die requirements opstel, dat doen vaak consultants, maar waneer ze dat volgens een goed proces doen met deze eigenschappen daarin dan zal het zeker het proces verbeteren." (r3)*

De applicatietester geeft aan dat hij geen tijd wil investeren. Hij vindt dat het leren van een gecontroleerde taal meer bij de personen hoort die de requirements opstellen.

> *"Nou nee, dat vind ik meer dat dat aan de voorkant thuis hoort. Ik stel nooit zelf de requirements op. Dat is het verschil en als je dat wel doet en ook als je ontwerper bent en dat hoort denk ik echt thuis bij consultantie. Het opstellen van de requirements." (r5)*

De respondenten vinden het moeilijk om aan te geven hoeveel tijd zij willen investeren. De tijd die zij willen investeren moet in positieve relatie staan met wat het oplevert. De respondenten geven aan dat zij geen ervaring met gestructureerde of formelen talen hebben.

In tabel 5.14 staan vijf taaleigenschappen. Per eigenschappen staat er hoeveel van de respondenten vinden dat deze in wel of niet in de requirements voorkomt.

Eigenschap	**Wel**	**Niet**
Afwisseling in structuur van zinnen	*4*	*1*
Tussenzinnen	*5*	*0*
Beeldspraak	*3*	*2*
Gezegde	*0*	*5*
Spreekwoord	*0*	*5*
Verwijzingen	*5*	*0*

Tabel 5.14 Aantal respondenten die aangeven dat een taaleigenschap wel of niet gebruikt wordt in de requirements.

De respondenten zien dat er in de requirements gebruik wordt gemaakt van verwijzingen. Afwisseling in structuur van zinnen en tussenzinnen komen bijna al de respondenten tegen in de requirements. Gezegde en spreekwoorden komen de respondenten niet tegen in de requirements. In tabel 5.15 is te zien welke taaleigenschappen de respondenten willen blijven gebruiken in de requirements.

Eigenschap	**Wel**	**Niet**
Afwisseling in structuur van zinnen	*4*	*1*
Tussenzinnen	*2*	*3*
Beeldspraak	*2*	*3*
Gezegde	*1*	*4*
Spreekwoord	*0*	*5*
Verwijzingen	*3*	*0*

Tabel 5.15 Aantal respondenten die aangeven dat een taaleigenschap wel of niet belangrijk is voor de requirements.

Afwisseling in structuur van zinnen vinden vier respondenten belangrijk voor de requirements. Het gebruik van tussenzinnen vindt de meerderheid van de respondenten niet belangrijk. De beide applicatieontwikkelaars geven aan dat zij tussenzinnen niet in de requirements willen. Zij merken dat requirements met tussenzinnen moeilijker te begrijpen zijn. Bij tussenzinnen is het risico op interpretatieproblemen groot. Een aantal respondenten geeft aan dat het gebruik van beeldspraak kan helpen om requirements duidelijker te maken. Een respondent (r4) vindt dat het gebruik van beeldspraak problemen kan opleveren in de communicatie met Indische collega's. De respondenten vinden dat spreekwoorden niet in de requirements horen. Drie respondenten hebben aangeven dat zij verwijzingen belangrijk vinden voor de requirements.

Afwisseling in structuur van zinnen willen vier respondenten graag gebruiken in de requirements. Gezegden en spreekwoorden wil bijna geen één van de respondenten gebruiken. Drie van de vijf respondenten heeft aangegeven dat zij het gebruik van verwijzingen essentieel vinden bij het opstellen van de requirements. In tabel 5.16 staan een aantal eigenschappen voor CNL talen. De respondenten hebben deze eigenschappen geordend op belang; van belangrijk naar minder belangrijk.

Eigenschap	**1**	**2**	**3**	**4**	**Gemiddeld**
Tool ondersteuning	*1*	*1*	*2*	*1*	3,25
Meertalig	*1*	*1*	*0*	*3*	3,75
Eenvoudig te leren	*3*	*1*	*1*	*0*	1,00
Handleiding beschikbaar	*0*	*2*	*2*	*1*	3,50

Tabel 5.16 Ordening van CNL eigenschappen door de respondenten.

Gemiddeld vinden de respondenten de eigenschap eenvoudig te leren de belangrijkste. Het minst belangrijk vinden de respondenten dat een CNL meer dan één taal ondersteunt.

5.1.4 *Enquête*

De enquête is ingevuld door achttien personen. Dit is de helft van de personen die binnen VitalHealth met requirements te maken hebben. Bij VitalHealth zijn er ongeveer vijfendertig personen die direct of indirect betrokken zijn bij de requirements. Respondenten de werkzaam zijn bij verschillende productlijnen hebben de enquête ingevuld. De meeste respondenten zijn werkzaam bij één productlijn, maar er zijn ook respondenten die bij meerdere productlijnen werkzaam zijn. De respondenten hebben verschillende rollen in de projecten. In tabel 5.17 is de verdeling in rollen van de respondenten te zien. Projectmanager is een nieuwe rol die een aantal respondenten hebben ingevuld.

Rol	
Applicatieontwikkelaar	33%
Requirements engineer	17%
Requirements engineer en ontwikkelaar	22%
Applicatietester	17%
Projectmanager	11%

Tabel 5.17 Verdeling van rollen van de respondenten in de enquête

De helft van de respondenten heeft ingevuld dat zij requirements opstellen. De respondenten die requirements opstellen hebben de rol van requirements engineer of requirements engineer en applicatieontwikkelaar. De beantwoording van deze vraag komt overeen met de veronderstelling dat requirements engineers de requirements opstellen. In tabel 5.18 is te zien hoe vaak de respondenten in de praktijk te maken hebben met requirements. De respondenten die requirements opstellen hebben vaker ingevuld dat zij "heel vaak" met requirements te maken hebben dan de respondenten die geen requirements opstellen.

	Opstellers	**Niet-opstellers**	**Alle**
Heel vaak	*67%*	*11%*	*39%*
Vaak	*22%*	*33%*	*28%*
Soms	*11%*	*44%*	*28%*
Nooit	*0%*	*11%*	*6%*

Tabel 5.18 Verdeling in hoeverre respondenten in de praktijk te maken hebben met requirements.

Requirements opstellen

De requirements kunnen gebruikt worden door verschillende rollen. De mate waarin een rol gebruik maakt van de requirements is verschillend. In diagram 5.6 is te zien in welke mate een rol gebruik maakt van de requirements die worden opgesteld door de requirements engineer.

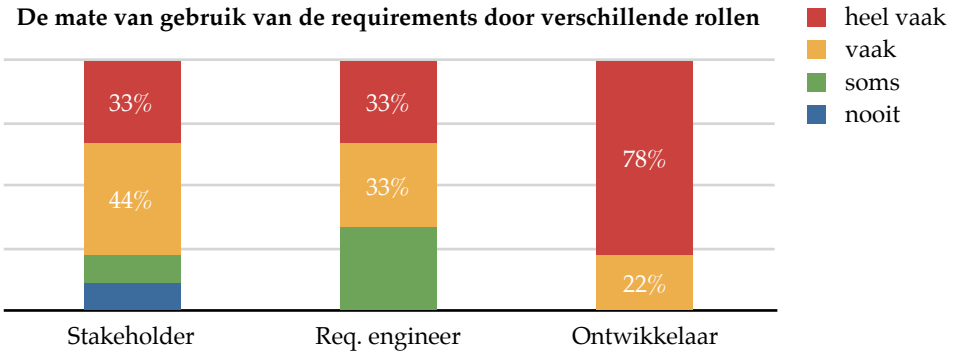


Diagram 5.6 In welke mate maken verschillende rollen gebruik van de requirements.

De respondenten geven aan dat de applicatieontwikkelaars het meest gebruik maken van de requirements. Een klein aantal respondenten (11%) heeft aangegeven dat de stakeholders nooit gebruik maken van de requirements die zij opstellen. Uit andere antwoorden van deze respondenten is af te leiden dat de respondenten werkzaam zijn in interne projecten en daar zelf de requirements voor opstellen. De requirements engineers in diagram 5 zijn de collega's van degenen die de requirements opstellen.

De requirements worden gebruikt door drie verschillende rollen. Elke rol heeft een andere achtergrond. Voor de opsteller van de requirements kan het lastig zijn om de software requirements op te stellen zodat deze goed te begrijpen zijn voor al de drie rollen. Aan de respondenten die requirements opstellen is gevraagd om aan te geven hoe bij hun de verhouding ligt bij het opstellen van de requirements. Worden de requirements alleen voor de stakeholders begrijpelijk opgesteld of alleen voor de applicatieontwikkelaar. De respondenten hebben dit aangeven met een nummer op een schaal die liep van één tot zeven. Bij nummer één stellen de respondenten de requirements alleen op voor de stakeholders, bij nummer zeven stellen de respondenten de requirements alleen op voor de applicatieontwikkelaars. Bij nummer vier houden de respondenten met beide rollen evenveel rekening. In diagram 5.7 is de verdeling te zien in hoeverre requirements engineers met een rol rekening houden bij het opstellen van de requirements.

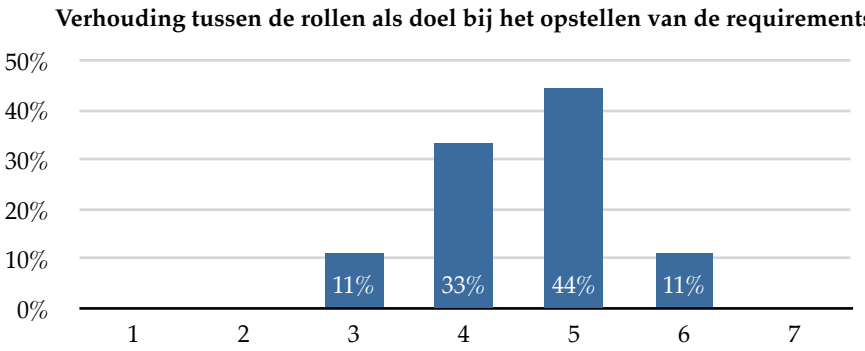


Diagram 5.7 Verhouding bij het opstellen van de software requirements.

Het gemiddelde van de ingevulde getallen is 4,56 met een standaarddeviatie van 0,83. De mediaan is 5, het laagst ingevulde getal is 3 en het hoogst ingevulde getal is 6. Uit de resultaten blijkt dat bij het opstellen van de requirements meer rekening wordt gehouden met de applicatieontwikkelaars dan met de stakeholders van de softwareapplicatie.

Notatietechniek

De meeste respondenten (61%) geven aan dat er binnen hun project geen gebruik wordt gemaakt van een notatietechniek. UML wordt het meest genoemd als notatietechniek die gebruikt wordt. Daarnaast worden er nog een aantal minder officiële notatietechnieken genoemd. Voorbeelden hiervan zijn: het gebruik van plaatjes of spreadsheet. In de meeste gevallen wordt er voor een notatietechniek gekozen omdat deze standaard gebruikt wordt in het project waarin de respondent werkzaam is.

Declaratief of Procedureel

Uit het requirementsdocumenten onderzoek blijkt dat requirements vaak implementatie afhankelijk worden opgesteld. De meeste requirements zijn procedureel beschreven. Bij de meeste respondenten (61%) heeft dit de voorkeur ten opzichte van declaratief opstellen van de requirements. Een respondent geeft de volgende toelichting waarom hij voor procedureel heeft gekozen:

> *"De eerste optie* (declaratief) *resulteert vaak weer in vervolg wensen. Het moet dan toch weer anders werken(procedureel) als beschreven."*

Een respondent geeft de volgende toelichting waarom hij voor declaratief heeft gekozen:

> *"Belangrijk is dat niet slechts de oplossing wordt beschreven, maar vooral ook het probleem en het probleem achter het probleem."*

Tekstueel of in een lijst

In de requirementsdocumenten viel het op dat de requirements worden opgesteld in een beschrijvend stukje tekst. In één stuk tekst werden diverse requirements opgenomen. Bij de meeste respondenten (83%) heeft deze vorm niet de voorkeur. De respondenten hebben de voorkeur voor het beschrijven van de requirements in een lijst. Als toelichting geven de meeste respondenten aan dat zij de voorkeur voor een lijst hebben omdat zij dit overzichtelijker vinden. Een respondent die voor een lijst heeft gekozen geeft hier de volgende reden voor:

> *"Dit is overzichtelijker dan dat er in een heel verhaal allemaal requirements staan."*

Twee andere respondent geven een andere reden waarom zij voor een lijst gekozen hebben:

"Bij lijstjes is het makkelijk om individuele requirements aan te wijzen en te testen."
"Lap tekst is vaak wollig, redundant, etc. Bij een lijst moet de opsteller goed nadenken, de juiste volgorde bepalen en prioriteren."

Een aantal (17%) respondenten geeft de voorkeur aan het tekstueel beschrijven van de requirements. Een respondent die hiervoor gekozen heeft geeft hier de volgende reden voor:

"Voor kleine projecten is dit voldoende."

Requirements engineerings proces

In de requirements wordt gebruik gemaakt van domeinwoorden en technische termen. De domeinwoorden zijn voor een stakeholder goed te begrijpen. De technische termen zijn voor de applicatieontwikkelaar goed te begrijpen. Andersom heeft de stakeholder technische kennis nodig om technische termen te begrijpen. De applicatieontwikkelaar heeft domeinkennis nodig om domein woorden te begrijpen. In diagram 5.8 en 5.9 is te zien hoe vaak het voorkomt dat een woord niet begrepen wordt in de requirements. In diagram 5.8 is te zien hoe vaak een requirements engineer dit meemaakt bij een stakeholder of ontwikkelaar. In diagram 5.9 is te zien hoe vaak een ontwikkelaar een woord niet begrijpt in de requirements.

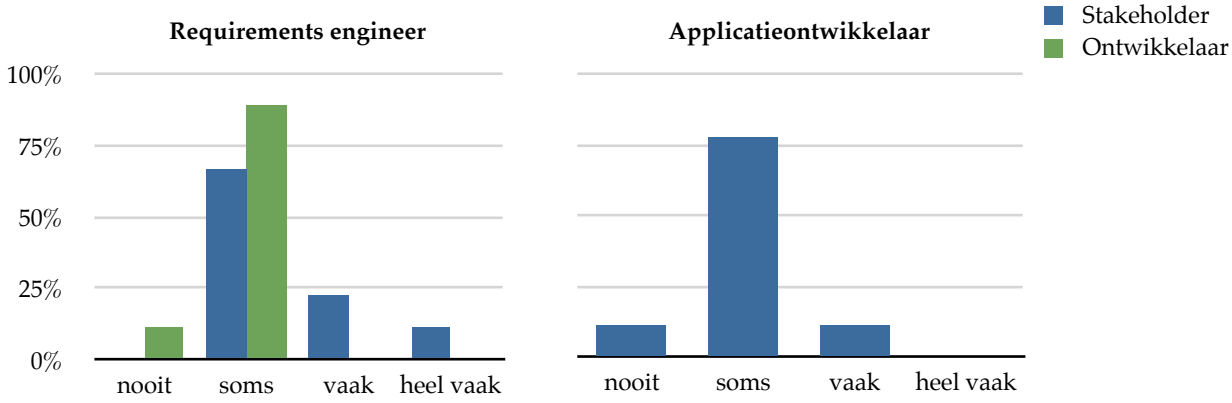


Diagram 5.8 Mate van het niet begrijpen van woorden in requirements naar de ervaring van requirements engineers.

Diagram 5.9 Mate van het niet begrijpen van woorden in requirements naar de ervaring van ontwikkelaars.

Uit de ervaring van de requirements engineers blijkt dat stakeholders vaker woorden in de requirements niet begrijpen dan ontwikkelaars. De ervaring van een klein aantal requirements engineers is dat stakeholders vaak (22%) of zelfs heel vaak (11%) woorden niet begrijpen. De ervaring met het niet begrijpen van woorden is bij ontwikkelaars ongeveer gelijk aan de ervaring van de requirements engineers hiermee. De ervaring van een klein deel (11%) van de ontwikkelaars is dat dit vaak voorkomt.

Domeinkennis kan op verschillende manieren worden overgedragen aan de applicatieontwikkelaar. In diagram 5.10 en 5.11 is te zien in welke mate de requirements engineer met de applicatieontwikkelaar communiceert om domeinkennis over te brengen. In diagram 5.10 wordt de ervaring van de requirements engineer weergegeven. In diagram 5.11 wordt de ervaring van de applicatieontwikkelaar weergegeven.

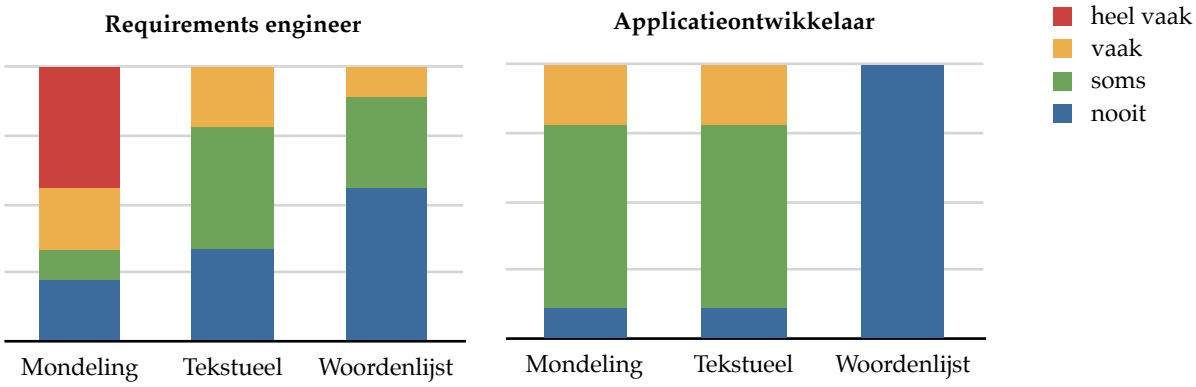


Diagram 5.10 Mate van domeinkennis overdracht naar de ervaring van requirements engineers.

Diagram 5.11 Mate van domeinkennis overdracht naar ervaring van applicatieontwikkelaars.

Mondelinge toelichting geven bij de requirements wordt het meest toegepast om domeinkennis over te dragen. Een toelichting schrijven bij de requirements met extra domein informatie wordt ook toegepast. Meer dan de helft (56%) van de requirements engineers voegt nooit een verklarende woordenlijst toe aan een requirementsdocument. De applicatieontwikkelaars zijn zelfs nog nooit een verklarende woordenlijst tegengekomen. Uit de interviews bleek dat alleen in de hoofd requirementsdocumenten verklarende woordenlijsten worden opgenomen. De respondenten die geen verklarende woordenlijsten tegenkomen maken waarschijnlijk geen gebruik van de hoofddocumenten. Hiermee is het verschil tussen de ervaring van applicatieontwikkelaars en requirements engineers te verklaren.

Communicatie

Door de opstellers van de requirements wordt gecommuniceerd met de stakeholders en applicatieontwikkelaars. De communicatie gaat op verschillende manieren. In diagram 5.12 is te zien in welke mate de requirements engineer met de stakeholder communiceert over de requirements.

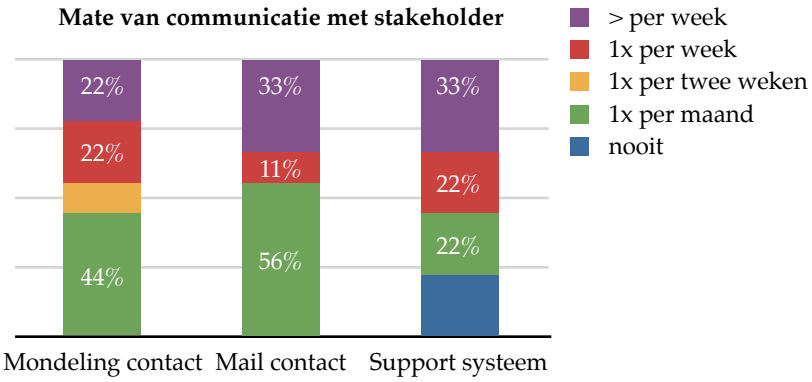


Diagram 5.12 Mate van communicatie van requirements engineer met de stakeholders.

Met het VitalHealth support systeem wordt het meest frequent gecommuniceerd. Meer dan de helft (55%) van de respondenten geeft aan dat zij één of meerdere keren per week contact hebben met de stakeholder via het supportsysteem. De respondenten hebben ook nog andere manieren van communiceren opgegeven. Dit zijn: tijdens het testen en via een document, beide zijn één keer genoemd.

In diagram 5.13 is te zien in welke mate de requirements engineer met de applicatieontwikkelaars communiceert.

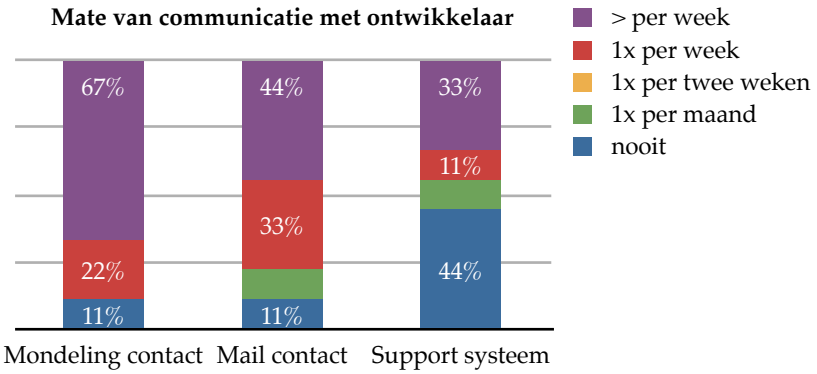


Diagram 5.13 Mate van communicatie van requirements engineer met de applicatieontwikkelaars.

De requirements engineer communiceert het meest frequent met de applicatieontwikkelaar door mondeling contact. Dit komt overeen met wat de requirements engineers in de interviews hebben aangegeven. De requirements engineers vinden mondeling contact de meest efficiënte manier om met de applicatieontwikkelaars te communiceren.

Requirements interpretatie

Requirements kunnen op meerdere manieren geïnterpreteerd worden. In diagram 5.14 is te zien hoe vaak requirements engineers meemaken dat een requirement verkeerd wordt geïnterpreteerd door een stakeholder of ontwikkelaar. In diagram 5.15 is te zien hoe vaak applicatieontwikkelaars requirements anders interpreteren dan de requirements engineer.

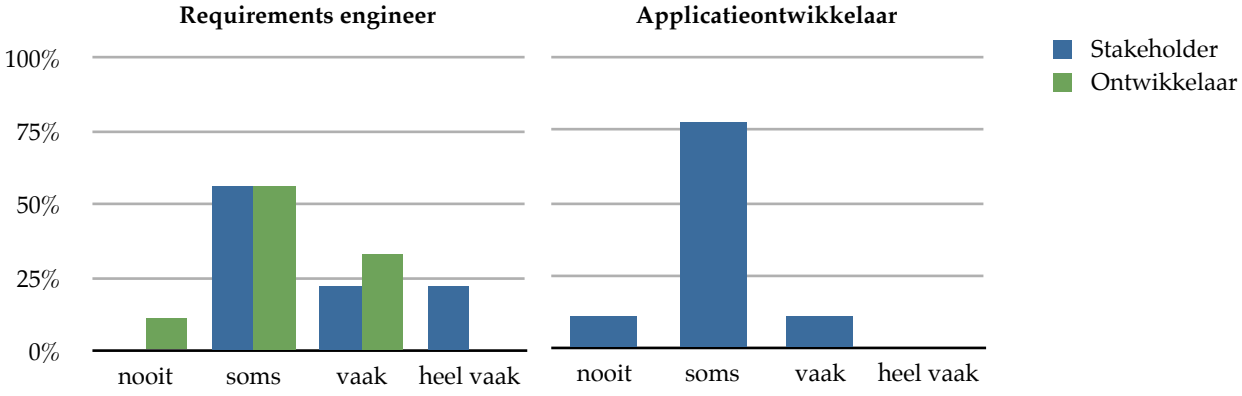


Diagram 5.14. Mate van verkeerd interpreteren van requirements op basis van ervaring van requirements engineers.

Diagram 5.15 Mate van verkeerd interpreteren van requirements op basis van ervaring van ontwikkelaars

De requirements engineers maken iets vaker mee dat een stakeholder de requirements verkeerd interpreteert dan dat een ontwikkelaar dit doet. Het gedeelte nooit bij de requirements engineers is ingevuld door respondenten die de rol van requirements engineer en applicatieontwikkelaar hebben. Deze groep verzameld zelf de requirements voor de applicatie die zij gaan bouwen. Zoals uit de resultaten blijkt komt het niet voor dat zij hun eigen requirements op een andere manier interpreteren.

Eén van de respondenten die geïnterviewd is geeft aan dat het slecht lezen van de requirements een oorzaak is van het verkeerd interpreteren van de requirements. De respondenten hebben op een schaal aangegeven hoe nauwkeurig de requirements door de stakeholders gelezen worden. De schaal liep van zeer onnauwkeurig (1) tot zeer nauwkeurig (7). In diagram 5.16 is te zien hoe nauwkeurig de requirements gelezen worden door de stakeholder.

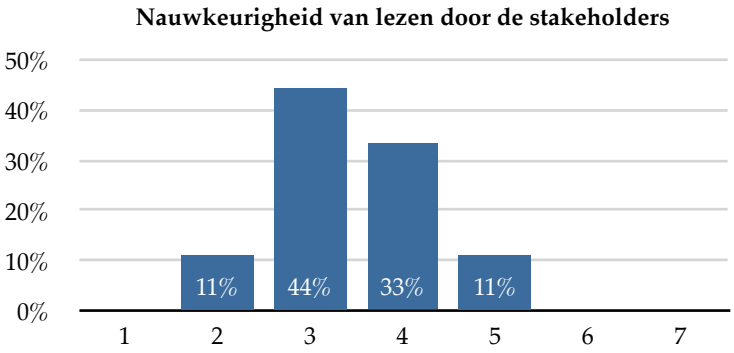


Diagram 5.16 Nauwkeurigheid van lezen van de requirements door de stakeholders.

Het gemiddelde van de ingevulde getallen is 3,44 met een standaarddeviatie van 0,83. De mediaan is 3, het laagst ingevulde getal is 2 en het hoogst ingevulde getal is 5. Hieruit blijkt dat de requirements engineers vinden dat de stakeholder de requirements eerder onnauwkeurig dan nauwkeurig lezen. De respondenten hebben toegelicht waarom zij vinden dat een stakeholder de requirements onnauwkeurig of nauwkeurig leest. De respondenten die aan de positieve kan zitten (44% respondenten die 4 of 5 hebben ingevuld) geven aan dat het verschilt per stakeholder. Een respondent die nummer vijf heeft ingevuld heeft dit op de volgende manier toegelicht:

> *"Het is verschillende per requirement, en ook per klant de ene is gemakkelijker in staat om het te lezen dan een ander"*

De respondenten die aan de positieve kan zitten (55% respondenten die 2 of 3 hebben ingevuld) geven aan dat uit de reacties van stakeholders vaak blijkt dat ze dingen niet gelezen hebben. Een respondent die nummer drie heeft ingevuld heeft dit op de volgende manier toegelicht:

> *"Het gebeurt regelmatig dat je vragen krijgt die niet nodig waren geweest als er goed gelezen was."*

De respondenten geven aan dat er verschil zit tussen lezen en begrijpen. Een respondent die nummer twee heeft ingevuld heeft dit op de volgende manier toegelicht:

> *"Lezen is 1, begrijpen is 2. Er wordt vaak niet goed doorzien wat het is"*

De requirements worden ook niet altijd doorgelezen door de stakeholders. De meeste (78%) van de respondenten geven aan dat het voorgekomen is dat de requirements niet door de stakeholder was doorgelezen.

De respondent die in het interview aangaf dat requirement slecht gelezen worden door stakeholders gaf dit ook aan voor de ontwikkelaars. De respondenten hebben op een schaal aangegeven hoe nauwkeurig de requirements door de applicatieontwikkelaars gelezen worden. De schaal liep van zeer onnauwkeurig (1) tot zeer nauwkeurig (7). In diagram 5.17 is te zien hoe nauwkeurig de requirements gelezen worden door de applicatieontwikkelaars. Hierbij is weergegeven wat de ervaring is van de requirements engineer en van de ontwikkelaars zelf.

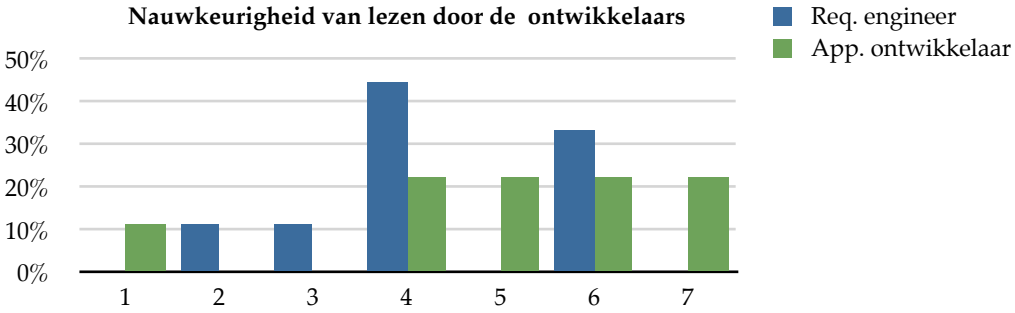


Diagram 5.17 Nauwkeurigheid van lezen van de requirements door de applicatieontwikkelaars.

Bij de requirements engineers is het gemiddelde van de ingevulde getallen 4,33 met een standaarddeviatie van 1,33. De mediaan is 4, het laagst ingevulde getal is 2 en het hoogst ingevulde getal is 6. Hieruit blijkt dat de requirements engineers vinden dat de ontwikkelaars niet heel onnauwkeurig, maar ook niet heel nauwkeurig de requirements lezen. In vergelijking met de stakeholders vinden de requirements engineers dat de applicatieontwikkelaars de requirements nauwkeuriger lezen. De spreiding van de getallen is in dit antwoord wel groter, de meningen van de

requirements engineers zijn meer verdeeld. Bij de applicatieontwikkelaars is het gemiddelde van de ingevulde getallen 5 met een standaarddeviatie van 1,73. De mediaan is 5, het laagst ingevulde getal is 1 en het hoogst ingevulde getal is 5. De applicatieontwikkelaars vinden gemiddeld dat ze de requirements nauwkeurig lezen. Hieruit blijkt dat de applicatieontwikkelaars positiever zijn dan de requirements engineers. De spreiding van de antwoorden is wel groot. De applicatieontwikkelaars hebben niet dezelfde mening over hoe zij de requirements lezen.

De requirements engineers hebben toegelicht waarom zij vinden dat een applicatieontwikkelaar de requirements onnauwkeurig of nauwkeurig leest. Een aantal respondenten geeft aan dat dit verschilt per applicatieontwikkelaar. Een respondent die nummer vier heeft ingevuld, heeft dit op de volgende manier toegelicht:

> *"Ook dit is weer afhankelijk van de persoon. Het wordt wel gelezen maar vaak oppervlakkig, details worden onvoldoende doorgrond"*

Sommige respondenten geven aan dat de kwaliteit van de requirements het lees gedrag beïnvloed. Bij requirements die gedetailleerde beschreven zijn lezen de applicatieontwikkelaars nauwkeuriger. Een respondent die nummer vier heeft ingevuld heeft dit op de volgende manier toegelicht:

> *"Als het gedetailleerd beschreven is wordt er vaak heel nauwkeurig gelezen. Als het globaal beschreven wordt blijft er ruimte voor interpretatie, ontstaat soms een voorstelling die niet bedoeld is."*

Met dit citaat geeft deze respondent een verband aan tussen de kwaliteit van de requirements en het nauwkeurig lezen van de requirements.
In de interviews kwam naar voren dat requirements engineers mondeling contact de meest efficiënte manier vinden om met de applicatieontwikkelaar te communiceren. Een respondent die nummer twee heeft ingevuld heeft dit op de volgende manier toegelicht:

> *"Het blijkt lastig om via documenten te communiceren. Er lijkt een bepaalde drempel voor ontwikkelaars om een document te gaan lezen. Mondelinge overdracht blijkt veel effectiever."*

Ambiguïteit in requirements
Aan de respondenten zijn vier oorzaken voorgelegd waardoor een requirement dubbelzinnig wordt. Dit zijn de volgende oorzaken: lexicale ambiguïteit, syntactische of semantische ambiguïteit, requirementsdocument ambiguïteit of een domein ambiguïteit. Aan de respondenten is gevraagd hoe vaak de genoemde oorzaken er de reden voor zijn dat requirements verkeerd worden geïnterpreteerd. In diagram 5.18 is het resultaat te zien.

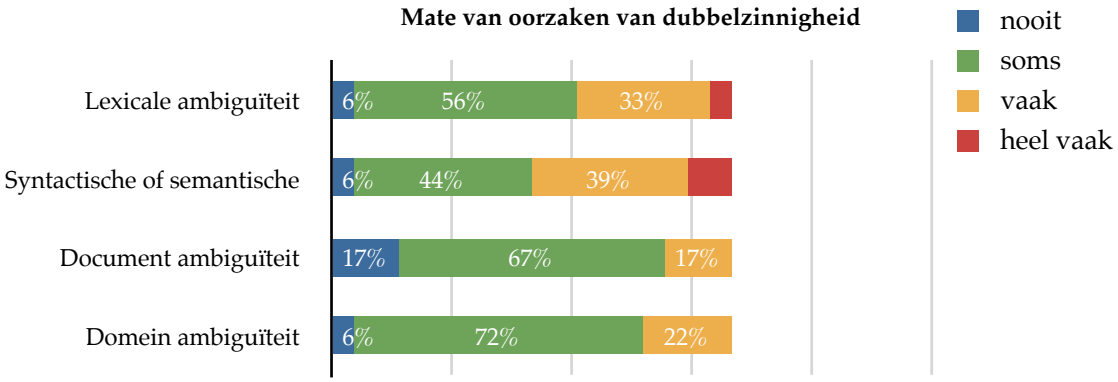


Diagram 5.18 Mate van het verkeerd interpreteren van een requirement per interpretatieprobleem-oorzaak.

De helft (50%) van de respondenten vindt syntactische of semantische ambiguïteit vaak of heel vaak de oorzaak van het verkeerd interpreteren van requirements. Minder dan de helft (38%) van de respondenten vindt lexicale ambiguïteit vaak of heel vaak de oorzaak voor het verkeerd interpreteren van requirements. Requirementsdocument en domein ambiguïteit zien de respondenten vooral soms als oorzaak van dubbelzinnigheid. Uit deze resultaten blijkt dat de respondenten de twee linguïstische ambiguïteiten veel vaker zien als oorzaak voor verkeerd interpreteren van requirements dan de requirements ambiguïteiten.

Kwaliteitscriteria voor requirements

Aan de respondenten is in de enquête gevraagd welke kwaliteitscriteria zij belangrijk vinden voor de requirements.

Eenduidig

In diagram 5.19 is te zien hoe belangrijk respondenten eenduidige requirements vinden.

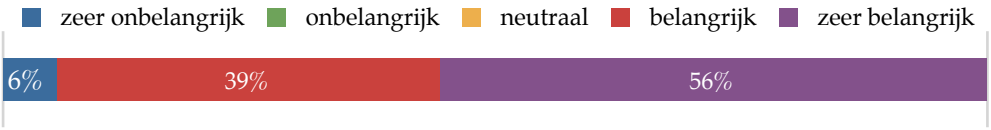


Diagram 5.19 Het belang van eenduidige requirements.

Het gemiddelde van de ingevulde getallen is 4,4 met een standaarddeviatie van 0,95. De mediaan is 5, de laagst waarde is 'zeer onbelangrijk' en de hoogste waarde is 'zeer belangrijk'. De hoge spreiding in de antwoorden wordt veroorzaakt door een klein (6%) gedeelte van respondenten die 'zeer onbelangrijk' heeft ingevuld. Uit deze resultaten blijkt dat de respondenten eenduidige requirements belangrijk vinden. Een klein aantal respondenten vinden eenduidige requirements onbelangrijk. In de interviews heeft ook een respondent aangegeven dit criterium onbelangrijk te vinden. De reden die hij hiervoor opgaf is: *"dat doel haal je toch nooit"*.

Consistent

In diagram 5.20 is te zien hoe belangrijk respondenten consistente requirements vinden.

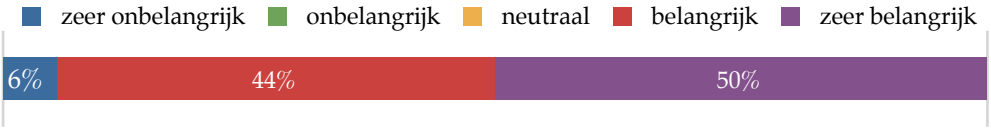


Diagram 5.20 Het belang van consistente requirements.

Het gemiddelde van de ingevulde getallen is 4,3 met een standaarddeviatie van 0,94. De mediaan is 4,5, de laagst waarde is 'zeer onbelangrijk' en de hoogste waarde is 'zeer belangrijk'. De hoge spreiding in de antwoorden wordt veroorzaakt door een klein (6%) gedeelte van respondenten die 'zeer onbelangrijk' heeft ingevuld.

Verifieerbaar

In diagram 5.21 is te zien hoe belangrijk respondenten verifieerbare requirements vinden.

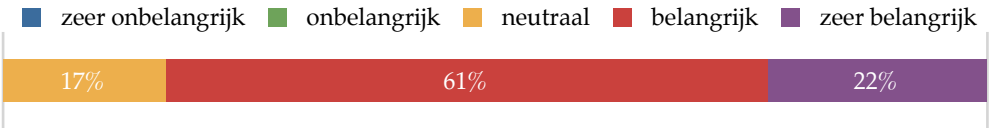


Diagram 5.21 Het belang van verifieerbare requirements.

Het gemiddelde van de ingevulde getallen is 4,1 met een standaarddeviatie van 0,62. De mediaan is 4, de laagst waarde is 'neutraal' en de hoogste waarde is 'zeer belangrijk'. De spreiding van de antwoorden is minder hoog dan bij de eigenschappen consistent en verifieerbaar.

Traceerbaar

In diagram 5.22 is te zien hoe belangrijk respondenten traceerbare requirements vinden.

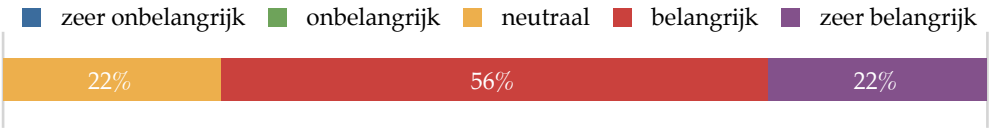


Diagram 5.22 Het belang van traceerbare requirements.

Het gemiddelde van de ingevulde getallen is 4 met een standaarddeviatie van 0,67. De mediaan is 4, de laagst waarde is 'neutraal' en de hoogste waarde is 'zeer belangrijk'. De spreiding van de antwoorden is minder hoog dan bij de eigenschappen consistent en verifieerbaar.

Implementatie onafhankelijk

In diagram 5.23 is te zien hoe belangrijk respondenten implementatie onafhankelijke requirements vinden.

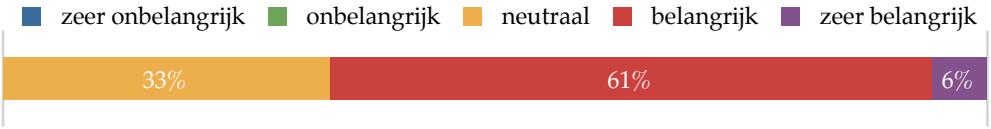


Diagram 5.23 Het belang van implementatie onafhankelijke requirements.

Het gemiddelde van de ingevulde getallen is 3,7 met een standaarddeviatie van 0,56. De mediaan is 4, de laagst waarde is 'neutraal' en de hoogste waarde is 'zeer belangrijk'. De spreiding van de antwoorden is het laagst van al de kwaliteitseigenschappen.

De vijf waarden zijn voorzien van een getal van één tot vijf. Op basis van deze waarde is een gemiddelde uitgerekend. In tabel 5.19 staan de requirements kwaliteitseigenschappen met daarbij het gemiddelde. De eigenschappen zijn gerangschikt op belang voor de requirements op basis van de mening van de respondenten.

Criteria	***Mediaan***	***Gemiddelde***	***SD***	
Eenduidig	*5,0*	*4,40*	*0,95*	*Zeer belangrijk*
Consistent	*4,5*	*4,30*	*0,94*	*Belangrijk*
Verifieerbaar	*4,0*	*4,10*	*0,62*	*Belangrijk*
Traceerbaar	*4,0*	*4,00*	*0,67*	*Belangrijk*
Implementatie onafhankelijk	*4,0*	*3,70*	*0,56*	*Belangrijk*

Tabel 5.19 Gemiddelde van het belang voor de requirements per kwaliteitscriteria.

De respondenten vinden de eigenschap eenduidig gemiddeld de meest belangrijke eigenschap voor de requirements.

<u>Gecontroleerde taal leren</u>

Met een gecontroleerde taal kan de kwaliteit van de requirements verbeterd worden. Een gecontroleerde taal is gebaseerd op een natuurlijke taal, maar bij een gecontroleerde taal zijn er restricties op de grammatica, woordenschat en semantiek gelegd. De meeste respondenten (89%) hebben nog nooit met een gecontroleerde taal gewerkt. Bij formele talen liggen er ook restrictie op de grammatica en semantiek. Ook hier hebben de meeste respondenten (83%) geen ervaring mee.

Om een gecontroleerde taal te gebruiken moet er tijd worden geïnvesteerd om de taal te leren. In diagram 5.24 is te zien hoeveel tijd de respondenten willen investeren om een gecontroleerde taal te leren.

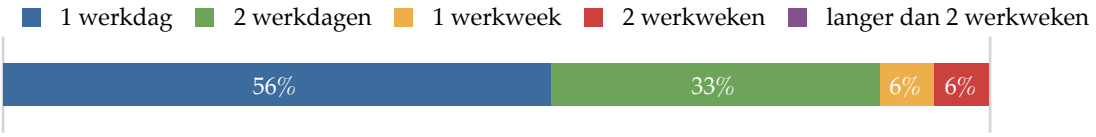


Diagram 5.24 Tijd die respondenten willen investeren om een CNL te leren.

Meer dan de helft (56%) van de respondenten heeft er maximaal één werkdag voor over om een gecontroleerde taal te leren. In diagram 5.25 is de verdeling te zien tussen de requirements engineers en de applicatieontwikkelaars in tijd die zij willen investeren om een gecontroleerde taal te leren.

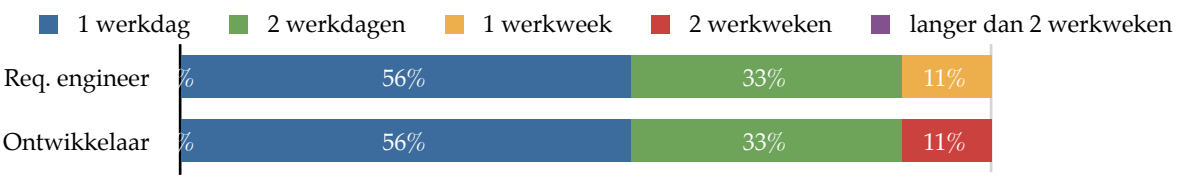


Diagram 5.25 Verdeling tussen requirements engineers en gebruikers van requirements in tijd die zij willen investeren om een CNL te leren.

In de resultaten is te zien dat er niet veel verschil zit tussen de requirements engineers en de applicatieontwikkelaars in de tijd die zij willen investeren om een gecontroleerde taal te leren.

Eigenschappen voor een gecontroleerde taal

De respondenten hebben de eigenschappen voor een gecontroleerde taal beoordeeld op hoe belangrijk zij deze vinden voor de requirements.

Verwijzing

In een requirement kan gebruik worden gemaakt van een verwijzing naar een andere requirement. In diagram 5.26 is te zien hoe belangrijk respondenten verwijzingen in de requirements vinden.

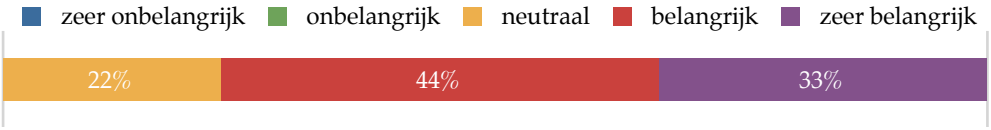


Diagram 5.26 Het belang van verwijzingen in de requirements.

Het gemiddelde van de ingevulde getallen is 4,11 met een standaarddeviatie van 0,74. De mediaan is 4, de laagst waarde is 'neutraal' en de hoogste waarde is 'zeer belangrijk'. Een klein aantal respondenten (22%) vindt deze eigenschap niet belangrijk of zeer belangrijk.

Opsomming

In een requirement kan gebruik worden gemaakt van opsommingen. In diagram 5.27 is te zien hoe belangrijk respondenten opsommingen in de requirements vinden.

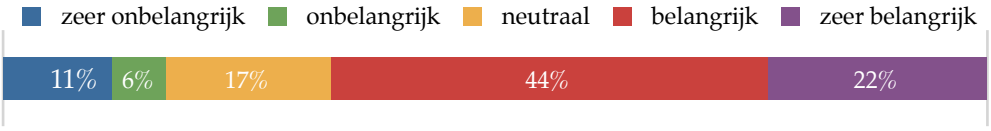


Diagram 5.27 Het belang van opsommingen in de requirements.

Het gemiddelde van de ingevulde getallen is 3,61 met een standaarddeviatie van 1,21. De mediaan is 4, de laagst waarde is 'zeer onbelangrijk' en de hoogste waarde is 'zeer belangrijk'. De spreiding van de antwoorden is hoog. Meer dan de helft (66%) van de respondenten vinden opsommingen belangrijk of zeer belangrijk.

Tussenzinnen

In een requirement kan gebruikt worden gemaakt van tussenzinnen. Dit zijn requirements zoals "De gebruiker, dit is een persoon in rol x, moet inloggen op het systeem". In diagram 5.28 is te zien hoe belangrijk respondenten opsommingen in de requirements vinden.

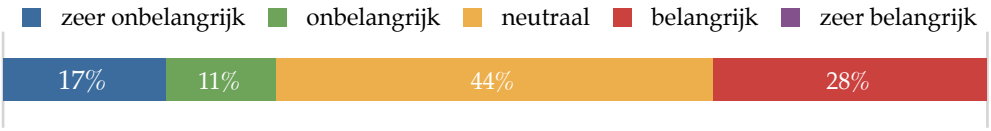


Diagram 5.28 Het belang van tussenzinnen in de requirements.

Het gemiddelde van de ingevulde getallen is 2,82 met een standaarddeviatie van 1,01. De mediaan is 3, de laagst waarde is 'zeer onbelangrijk' en de hoogste waarde is 'belangrijk'. De spreiding van de antwoorden is hoog. Minder dan de helft (28%) van de respondenten vinden het gebruik van tussenzinnen in de requirements belangrijk.

Afwisseling in structuur van zinnen

In een requirements kan gebruikt worden gemaakt van afwisseling in de structuur van zinnen. Dit zijn requirements zoals "Patiënt heeft ziekte x en is gebruiker" en later voor een requirements met dezelfde betekenis de volgende structuur "De patiënt is gebruiker en heeft ziekte x" hanteert. In diagram 5.29 is te zien hoe belangrijk respondenten opsommingen in de requirements vinden.

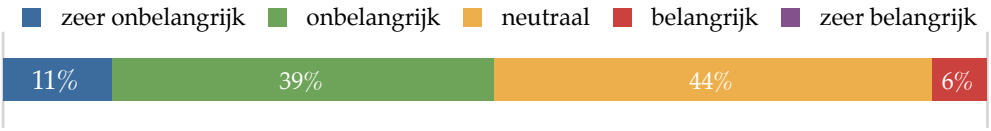


Diagram 5.29 Het belang van afwisseling van structuur van zinnen in de requirements.

Het gemiddelde van de ingevulde getallen is 2,44 met een standaarddeviatie van 0,76. De mediaan is 2,5, de laagst waarde is 'zeer onbelangrijk' en de hoogste waarde is 'belangrijk'. Weinig van de respondenten (6%) vinden het gebruik van afwisseling in structuur van zinnen in de requirements belangrijk.

Beeldspraak

In de requirements kan gebruik worden gemaakt van beeldspraak. In de requirements wordt gebruik gemaakt van zinsdelen zoals "als een kapstok gebruiken" of "in de database kijken". In diagram 5.30 is te zien hoe belangrijk respondenten beeldspraak in de requirements vinden.

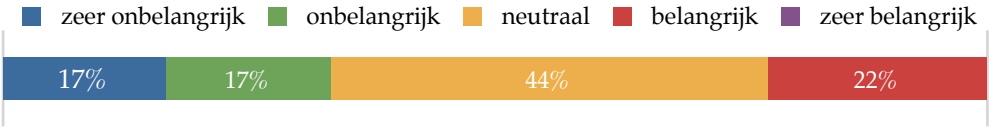


Diagram 5.30 Het belang van tussenzinnen in de requirements.

Het gemiddelde van de ingevulde getallen is 2,72 met een standaarddeviatie van 0,98. De mediaan is 3, de laagst waarde is 'zeer onbelangrijk' en de hoogste waarde is 'belangrijk'. Een klein aantal van de respondenten (28%) vinden het gebruik van beeldspraak in de requirements belangrijk.

Gezegden

In de requirements kan gebruik worden gemaakt van gezegden. In de requirements wordt gebruik gemaakt van zinsdelen zoals "aan het roer zitten". In diagram 5.31 is te zien hoe belangrijk respondenten gezegden in de requirements vinden.

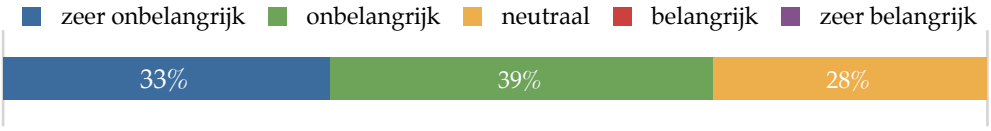


Diagram 5.31 Het belang van gezegden in de requirements.

Het gemiddelde van de ingevulde getallen is 1,94 met een standaarddeviatie van 0,78. De mediaan is 2. De laagst waarde is 'zeer onbelangrijk' en de hoogste waarde is 'neutraal'. Geen van de respondenten vinden het gebruik van gezegde in de requirements belangrijk.

Spreekwoorden

Bij natuurlijke taal kan gebruik gemaakt worden van spreekwoorden. In diagram 5.32 is te zien hoe belangrijk respondenten spreekwoorden in de requirements vinden.

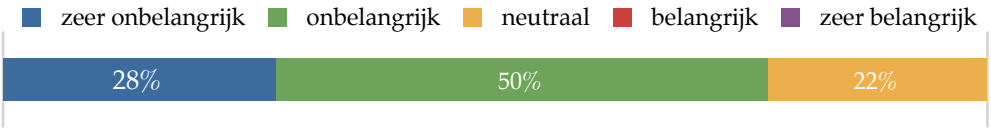


Diagram 5.32 Het belang van spreekwoorden in de requirements.

Het gemiddelde van de ingevulde getallen is 1,94 met een standaarddeviatie van 0,7. De mediaan is 3, de laagst waarde is 'zeer onbelangrijk' en de hoogste waarde is 'neutraal'. Geen van de respondenten vinden het gebruik van spreekwoorden in de requirements belangrijk.

De vijf waarden van de schaal zijn voorzien van een getal van één tot vijf. Op basis van deze waarde is een gemiddelde uitgerekend. In tabel 5.20 staan de eigenschappen met daarbij het gemiddelde. De eigenschappen zijn gerangschikt op belang voor de requirements op basis van de mening van de respondenten.

Eigenschap	*Mediaan*	*Gemiddelde*	*SD*	
Verwijzingen	*4,0*	*4,11*	*0,74*	*Belangrijk*
Opsommingen	*4,0*	*3,61*	*1,21*	*Belangrijk*
Tussenzinnen	*3,0*	*2,82*	*1,01*	*Neutraal*
Beeldspraak	*3,0*	*2,72*	*0,98*	*Neutraal*
Afwisseling structuur zinnen	*2,5*	*2,44*	*0,76*	*Onbelangrijk*
Gezegde	*2,0*	*1,94*	*0,78*	*Onbelangrijk*
Spreekwoord	*2,0*	*1,94*	*0,70*	*Onbelangrijk*

Tabel 5.20 Gemiddelde van het belang voor de requirements per taaleigenschap.

De respondenten vinden verwijzingen en opsommingen belangrijk om te gebruiken in de requirements. Het gebruik van een gezegde of spreekwoorden vinden de respondenten onbelangrijk. In tabel 5.21 staan nog twee andere eigenschappen van een gecontroleerde taal. Bij een aantal gecontroleerde talen is een tool ontwikkeld. Bij het opstellen van de requirements wordt dan ondersteunt door een tool. De meeste gecontroleerde talen zijn opgesteld voor het Engels. Er is wel een klein aantal talen die meerdere talen ondersteunen.

Eigenschap	*Mediaan*	*Gemiddelde*	*SD*	
Tool ondersteuning	*3,5*	*3,50*	*0,83*	*Belangrijk*
Meertalig	*3,0*	*3,11*	*0,87*	*Neutraal*

Tabel 5.21 Eigenschappen voor een CNL taal.

De respondenten vinden het belangrijk dat de gecontroleerde taal wordt ondersteunt door een tool. Het ondersteunen van meerdere talen door een CNL vinden de respondenten minder belangrijk.

5.1.5 *Vergelijking van de gecontroleerde talen*

Op basis van de eigenschappenlijst zijn de CNL talen met elkaar vergeleken. De uitgebreide beschrijvingen van de talen is terug te vinden in paragraaf 3.3. In de bijlage D *"Controled Natural Language"* staat er per taal een overzicht hoe tot de resultaten gekomen is.
Per eigenschap zijn er drie antwoorden mogelijk: plus, plus/min en min. Het plusteken houdt in dat de taal over de eigenschap beschikt. Bij het min teken beschikt de taal niet over de eigenschap en bij het plus / min teken beschikt de taal ten dele over de eigenschap. In hoofdstuk zes is de exacte betekenis van de antwoorden uitgewerkt.

Nauwkeurigheid

In tabel 5.22 staat in hoeverre ambiguïteit voorkomen wordt bij het gebruik van de CNL taal.

Eigenschap	ACE	PENG	CLCE	CPL
Lexicale ambiguïteit	+/-	+/-	+/-	+
Syntactische ambiguïteit	+	+	+	+/-
Semantische ambiguïteit	+	+	+	+/-
Pragmatische ambiguïteit	+	+	+	+

Tabel 5.22 CNL taal vergelijking op nauwkeurigheid per ambiguïteitstype.

Voor het voorkomen van ambiguïteit gebruiken de eerste drie talen dezelfde methoden. Lexicale ambiguïteit wordt voorkomen door voor-gedefinieerde woorden of domeinwoorden die de gebruiker zelf moet invoeren. Om syntactische en semantische ambiguïteit tegen te gaan hebben deze talen regels die de grammatica en semantiek van een natuurlijke taal beperken. Pragmatische ambiguïteit, vooral referentiële ambiguïteit, wordt voorkomen door anaforen en antecedenten door syntactische regels aan elkaar te verbinden. CPL maak gebruik van een corpus hieruit haalt CPL de betekenis van een woord. Ambiguïteit is in CPL in bepaalde mate toegestaan. CPL zinnen met ambiguïteit worden door een CPL interpretator gehaald die de mogelijke betekenissen van een zin weergeeft, het is aan de gebruiker om te kiezen welke betekenis de juiste is.
In de hierop volgende tekst wordt er per linguïstische ambiguïteit beschreven hoe de talen dit voorkomen.

Lexicale ambiguïteit
In ACE en PENG zijn twee verschillende soorten van woorden opgenomen, functiewoorden en domeinwoorden. De functiewoorden zijn vooraf gedefinieerd, maar de domeinwoorden kunnen door de gebruiker worden toegevoegd aan de taal. In ACE en PENG kunnen veel domeinwoorden worden toegevoegd. Deze woorden zijn puur syntactische en hebben geen betekenis in de taal. Het is aan de gebruiker om het verschil te bepalen tussen de woorden. Hiermee leggen ACE en PENG de verantwoordelijkheid voor het checken van polysemie bij de gebruiker neer, zij kiezen welke woorden ze willen gebruiken in requirements en moeten zelf ook de consequentie dragen (Fuchs & Schwitter, 1996). Gruzitis en Barzdins (2010) beschrijven dat de werkwijze van ACE en PENG problemen kan opleveren. Een gebruiker kan namelijk niet al de mogelijke betekenissen van een woord tot in de detail weten. Als voorbeeld geven Gruzitis en Barzdins het woord "take". In de context van een zin kan het woord "take" betekenen: een foto maken met een camera, iemand helpen door hem thuis te brengen of iets anders. Met ACE en PENG wordt polysemie in de requirements niet voorkomen, de gebruiker moet zich hiervan bewust zijn bij het aanmaken van de domeinwoorden die hij gaat gebruiken in de requirements. Hetzelfde is het geval bij het voorkomen van systematische ambiguïteit. Bij het gebruik van ACE en PENG moet een gebruiker zich ook hiervan bewust zijn wanneer er woorden worden toegevoegd aan de domeinwoorden.
CLCE gaat lexicale ambiguïteit tegen door de correcte betekenis van ieder gereserveerd woord te beschrijven. De betekenis van de gereserveerde woorden is uniek gedefinieerd in de syntax. De betekenis van de gebruiker gedefinieerde woorden is gelimiteerd tot één betekenis van het woord die door de gebruiker aan het woord is gegeven. Op dezelfde manier als bij gecontroleerde talen ACE en PENG wordt hiermee lexicale ambiguïteit niet volledig voorkomen. Gebruikers kunnen aan woorden een dubbelzinnige betekenis geven.
Om lexicale ambiguïteit tegen te gaan gebruikt CPL WordNet als ontologie. WordNet is een grote database waarin Engelse woorden zijn opgeslagen zoals zelfstandig naamwoorden, werkwoorden, voorzetsels en bijwoorden. De betekenis de woorden wordt gepresenteerd aan de gebruiker voor verificatie of voor correctie. In tegenstelling tot ACE en PENG hoeft de gebruiker niet zelf woorden en betekenissen in te voeren.

Syntactische ambiguïteit
Fuchs (1996) beschrijft dat het door elkaar gebruiken van tegenwoordige en verleden tijd kan leiden tot analytische ambiguïteit. Om dit te voorkomen is het gebruik van verleden tijd in ACE niet toegestaan.

In het volgende voorbeeld staat een requirement die door een attachment ambiguïteit meer dan één betekenis heeft.

"The user must enter a card with a code."

Het zinsdeel *"with a code"* kan bevestigd worden aan *"must enter"* en aan *"a card"*. In ACE is alleen de eerste mogelijkheid mogelijk. Dit komt doordat in ACE de regel minimal attachment wordt toegepast. De meest eenvoudige syntactische structuur geeft de betekenis van de zin aan. Wanneer deze betekenis niet is wat de auteur van de requirement bedoelt dan moet de auteur de requirement herformuleren. Om te beschrijven dat de code bij de kaart hoort moet de auteur de requirement op de volgende manier formuleren: *"The user must enter a card that carries a code"*. Om te beschrijven dat beide kaart en code moeten worden gebruikt kan de auteur beter gebruik maken van de volgende constructie: *"The user must enter a card and a code"* (Fuchs, Höfler, Kaljurand, Rinaldi, & Schneider, 2005).

In PENG wordt attachment ambiguïteit voorkomen door het modificatie principe. Een voorbeeld van een requirement die niet meer ambigu is in PENG is de volgende:

"Gebruikers hebben netwerk toegang bij praktijken met hetzelfde netwerk".

De requirement heeft de volgende betekenissen: *"gebruikers hebben toegang tot hetzelfde netwerk vanuit een andere praktijk"* of "*gebruikers hebben netwerk toegang bij praktijken met hetzelfde netwerk"*. Door het modificatie principe zal de betekenis van deze requirement in PENG zijn "*Gebruikers hebben {netwerk toegang bij praktijken met hetzelfde netwerk}*".

In het Engels mogen voorzetsels verbonden worden aan zelfstandig naamwoorden, werkwoorden, bijvoeglijke naamwoorden of bijwoorden. In CLCE mag alleen de voorzetsel "*of*" worden verbonden aan een zelfstandig naamwoord. Andere voorzetsels mogen alleen verbonden worden aan werkwoorden. Het volgende requirement met een attachment ambiguïteit is niet toegestaan in CLCE:

"The user must enter a card with a code."

Omdat CLCE alleen het voorzetsel "*of*" mag worden verbonden aan een zelfstandig naamwoord is deze zin niet toegestaan. In dit requirement is het voorzetsel "*with*" verbonden aan het zelfstandig naamwoord "*code*".

In het volgende voorbeeld is een stukje uit een requirement genomen waarin een coördinatie ambiguïteit voorkomt.

"Patient and Caregiver must enter a code."

In dit voorbeeld is het niet duidelijk of de patiënt en de caregiver samen of apart een code moeten invoeren. Om deze ambiguïteit te voorkomen zijn in ACE de woorden *"together"* en "*each*" geïntroduceerd (Schwitter & Fuchs, 1996). Wanneer de betekenis van de requirement moet zijn dat de code door beide personen moet worden ingevoerd dan moet de volgende zin gebruikt worden: *"Patient and Cargeiver must enter a code together"*. Bij de betekenis dat beide personen een code moeten invoeren maar dit niet noodzakelijk tegelijk gedaan moet worden, moet de volgende zin gebruikt worden: *"Patient and Cargeiver must each enter a code"*.

In PENG wordt coördinatie ambiguïteit voorkomen door het distributie principe en het coördinatie principe. Op het volgende voorbeeld is het distributie principe van toepassing:

"Gebruikers hebben netwerk toegang door vingerafdruk authenticatie en een speciale code".

Door het distributie principe heeft de zin in PENG de volgende betekenis "*Gebruikers hebben netwerk toegang door vingerafdruk authenticatie en {hebben netwerk toegang door} een speciale code*". Bij het distributie principe gelden werkwoorden voor al de delen van de zin. In het geval van de requirement geldt het werkwoord *hebben* voor *vingerafdruk authenticatie* en voor *speciale code*. In het volgende voorbeeld is het coördinatie principe van toepassing bij de betekenis van de requirement:

"De gebruiker heeft toegang tot het netwerk dat beschikbaar is en online is".

In de requirement is het verwarrend of de gebruiker of het netwerk online moet zijn. Bij het coördinatie principe hoort een samengevoegde werkwoord zin bij het hoofdzinsdeel en niet bij de relatieve zin. Hierdoor is de betekenis van deze requirement *"De gebruiker {heeft toegang tot het netwerk dat beschikbaar is} en {online is}"*.

In het volgende voorbeeld komt een coördinatie ambiguïteit voor:

"Both internal and external users are linked to exactly one Platform type Users."

De requirement kan de betekenis hebben van *"internal users and external users"* of *"internal and external users"*. In CLCE wordt deze ambiguïteit voorkomen door het bijvoeglijke naamwoord te verbinden aan het zelfstandig naamwoord. In CLCE wordt dit op de volgende manier gedaan: "*internal_users and external_users*". In de zin wordt gebruik gemaakt van een underscore.

CPL heeft een andere werkwijze om syntactische en semantische ambiguïteit in zinnen te voorkomen dan de talen ACE, PENG en CLCE. In CPL wordt dit voorkomen door gebruik van de kennis representatie taal Knowledge Machine (KM). Het volgende proces wordt doorlopen om ambiguïteit in CPL te voorkomen: Eerst moet de gebruiker zinnen die gemaakt zijn in een natuurlijke taal vertalen naar CPL. Bij het vertalen moet de gebruiker rekening houden met de grammaticale regels die gelden voor CPL. Bij het herformuleren van zinnen naar CPL is het de taak van de gebruiker om de natuurlijke taal zinnen nauwkeuriger te maken op een manier zodat deze ondubbelzinnig door de KM geïnterpreteerd kunnen worden. Bij deze manier van werken heeft de gebruiker veel kennis en training van de taal nodig. Ten tweede moet de zin door de CPL interpretator worden gehaald die op basis van KM de betekenis van de zin definieert. De input van de CPL interpretator is in veel gevallen nog ambigu. Het doel van de CPL interpretator is om de ambiguïteit op te lossen. De CPL interpretator laat aan de gebruiker zien welke betekenis de zin heeft gekregen. In sommige gevallen

maakt de CPL interpretator fouten of heeft een zin meer dan één betekenis. De gebruiker moet dan de zin herformuleren zodat de zin in CPL de gewenste betekenis heeft.

Semantische ambiguïteit
De referentiële ambiguïteit kan zowel in de context van de zin als in de context van de tekst voorkomen. De referentiële ambiguïteit wordt daarom beschreven bij de pragmatische ambiguïteit.

In het volgende voorbeeld is een stukje uit een requirement genomen waarin een scope ambiguïteit voorkomt.

"Every patient is not a user."

De requirement kan betekenen dat er geen patiënt is die ook gebruiker is of dat niet iedere patiënt ook gebruiker is. De regel in ACE is dat de scope van een kwantor opent op de plek waar de kwantor staat en sluit aan het einde van de zin (Kaljurand, 2007). Dit betekent dat de zin *"Every patient is not a user"* in ACE de betekenis heeft van *"Voor al de patiënten geldt er is geen patiënt die ook een gebruiker is"*.

In PENG wordt hetzelfde principe toegepast als in ACE om scope ambiguïteit te voorkomen. In het volgende voorbeeld is een stukje uit een requirement genomen waarin een scope ambiguïteit voorkomt.

"Every user is not a cargiver."

Deze requirement kan betekenen dat er geen gebruiker is die ook caregiver is of dat niet iedere gebruiker ook caregiver is. Het scope principe in PENG schrijft voor dat de scope van een kwantor opent op de plek waar de kwantor staat en sluit aan het einde van de zin. Dit betekent dat de zin *"Every user is not a cargiver."* in PENG de betekenis heeft van *"Voor al de gebruikers geldt er is geen gebruiker die ook een cargiver is."*

In het Engels is het toegestaan om existentiële en universele kwantoren gemengd door elkaar te gebruiken in verschillende contexten. In CLCE is de positie van een universele kwantor beperkt tot het onderwerp van de zin. Iedere kwantor die op een andere positie geplaatst wordt moet een existentiële kwantor zijn.

Pragmatische ambiguïteit
Om referentiële ambiguïteit in zinnen te voorkomen heeft ACE regels om een anafoor met een antecedent te verbinden. Bij een verwijzing in een zin zoekt ACE in de voorgaande tekst naar een verwijzingen die hetzelfde nummer en geslacht heeft als het verwijswoord. Een tekst bestaat meestal uit meerdere zinnen. Het volgende voorbeeld bestaat uit een stuk tekst waarin meerdere requirements voorkomen.

"A patient enters a red card and a code. If the code is valid then EPD accepts the blue card. If the code is not valid then EPD rejects the card."

In de verschillende zinnen in dit voorbeeld wordt verwezen naar objecten uit de voorgaande zin. Voorbeelden hiervan zijn de objecten "*code*" en "*card*". Voor het object "*card*" bestaan er twee verschillende objecten: "*red card*" en de "*blue card*". In de laatste zin komt een referentiële ambiguïteit voor. De verwijzingen "*the card"* kan verwijzen naar de *"red card"* en naar de *"blue card"*. In ACE verwijst een anafoor naar de meest recente antecedent met hetzelfde nummer en geslacht. In dit geval verwijst de anafoor naar de antecedent *"blue card"*.

Om referentiële ambiguïteit in zinnen te voorkomen heeft PENG het anaforaprincipe. In PENG refereert een verwijswoord altijd naar het meest recente toegankelijke zelfstandige naamwoord die geschikt is. In de volgende zinnen komt een referentiële ambiguïteit voor.

"A user has a card X and a card Y. The user enters the card and a code."

Voor het object "*the card*" bestaan er twee verschillende objecten: "*X*" en "*Y*". Omdat in PENG de anafoor verwijst naar de meest recente toegankelijke zelfstandige naamwoord is in dit geval "*Y*" de antecedent.

Om referentiële ambiguïteit te voorkomen worden in CLCE zelfstandig voornaamwoorden omgezet naar variabelen. Hierdoor kan een unieke referentie worden gemaakt naar een object in de voorgaande tekst.

Een anafoor kan een voornaamwoord zijn. In CPL zijn geen voornaamwoorden toegestaan. Het verwijzen in een zin naar een andere zin met een voornaamwoord is niet mogelijk in CPL.
De volgende requirement is niet toegestaan in CPL. In deze requirement wordt met het woord "*it*" verwezen naar de "*card*".

"The patient has a card. The patient must use it to login to the system."

In CPL moet deze requirement geherformuleerd worden naar de volgende requirement. In deze requirement is het woord "it" vervangen door "the card".

"The patient has a card. The patient must use the card to login to the system."

Bij de herformulatie is gebruik gemaakt van de richtlijn: "*Gebruik 'a' om een object te introduceren en 'the' om naar hetzelfde object te refereren*".

Expressiviteit

De expressiviteit van de gecontroleerde taal geeft aan hoeveel linguïstische expressie de gecontroleerde taal ondersteunt. Een taal X is expressiever dan een taal Y wanneer taal X alles kan beschrijven van taal Y maar niet andersom. In tabel 5.23 staat in hoeverre taal eigenschappen worden ondersteunt door de gecontroleerde taal.

Eigenschap	**ACE**	**PENG**	**CLCE**	**CPL**
Maakt de CNL gebruik van een eigen woordenlijst	+/-	+/-	+/-	+
Kunnen er woorden worden toegevoegd aan de woordenlijst	+	+	+	-
Meervoud / enkelvoud	+	+	+	+
Heeft de CNL taal regels voor nominalisatie	-	-	-	+
Is een andere schrijfwijze van een woord toegestaan	+	+	-	+
Kunnen er verwijzingen worden gemaakt.	+	+	+	+/-
Kunnen er opsommingen worden gemaakt	+/-	-	+	+/-
Zijn tussenzinnen toegestaan.	-	-	-	-
Is beeldspraak toegestaan.	-	-	-	-
Zijn gezegde toegestaan.	-	-	-	-
Zijn spreekwoorden toegestaan.	-	-	-	-

Tabel 5.23 CNL taal vergelijking op expressiviteit per eigenschap.

In de resultaten is te zien dat figuurlijke taal niet toegestaan is in de gecontroleerde talen. Taal eigenschappen zoals, beeldspraak, gezegde en spreekwoorden worden niet ondersteunt door de talen. Figuurlijke taal is in zichzelf ambigu. Figuurlijke taal heeft meerder betekenissen; een letterlijke en een figuurlijke. Om ambiguïteit in zinnen te voorkomen moet figuurlijke taal niet worden toegestaan in zinnen.

Natuurlijk karakter

Het natuurlijke karakter van een gecontroleerde taal is in hoeverre de taal op een natuurlijke taal lijkt in termen van leesbaarheid en begrijpelijkheid. Het grote nadeel van formele talen is dat ze vaak nogal moeilijk te begrijpen zijn voor domein specialisten en stakeholders. Omdat een CNL afgeleid is van een natuurlijke taal is het voor personen die de natuurlijke taal beheersen makkelijker om deze te leren, gebruiken, lezen en schrijven. In tabel 5.24 is te zien in hoeverre de gecontroleerde talen een natuurlijk karakter hebben.

Eigenschap	**ACE**	**PENG**	**CLCE**	**CPL**
Is de taal makkelijk te leren?	+/-	+	-	-
Is de taal makkelijk te lezen?	+	+	+	+

Is de taal makkelijk te schrijven?	+	+	-	+/-
Is de taal makkelijk te begrijpen?	+	+	+	+

Tabel 5.24 CNL taal vergelijking op natuurlijke karakter per eigenschap.

De gecontroleerde taal PENG is het makkelijkst te leren en te gebruiken. PENG beschikt over een tool die de gebruiker door de regels van PENG heen leidt. Bij het intypen van een zin in de PENG tool, geeft de tool aan welke mogelijkheden de gebruiker heeft om de zin verder te schrijven. De talen CLCE en CPL zijn moeilijk te leren. CLCE is gebaseerd op eerste orde logica talen, de taal gebruikt dezelfde regels voor de grammatica en semantiek als de formele talen. Om een zin met CLCE te schrijven moet de gebruiker kennis hebben van formele talen. Van CPL noemt de ontwikkelaar Clark (2005) de gebruiksvriendelijkheid één van de uitdagingen die nog verbeterd moeten worden aan CPL.

Eenvoud

Met de eenvoud eigenschap wordt bedoelt in hoeverre het eenvoudig of complex is om de taal om te zetten naar een formele taal. In tabel 5.25 staan de eigenschappen die een relatie hebben met deze eigenschap.

Eigenschap	**ACE**	**PENG**	**CLCE**	**CPL**
Worden zinnen omgezet naar een formele taal.	+	+	+	+/-
Wordt de discours omgezet naar een formele taal.	+/-	+/-	+/-	+/-
Is er een mapping naar een grafische presentatie mogelijk.	-	-	+	-
Ondersteunt de CNL taal meerder natuurlijke talen.	-	-	-	-
Beschikt de CNL over een style richtlijn	+	+	-	+
Wordt de CNL taal ondersteunt door een tool?	+/-	+	-	+

Tabel 5.25 CNL taal vergelijking op eenvoud per eigenschap.

De zinnen die gemaakt worden met de gecontroleerde talen zijn om te zetten naar zinnen in formele talen. Bij de eerste drie talen (ACE, PENG en CLCE) kan dit één op één; een zin in deze gecontroleerde taal heeft maar één zin in een formele taal die precies dezelfde betekenis heeft. De zinnen die gemaakt worden met CPL hebben een één op veel relatie met formele zinnen; een CPL zin heeft meerdere formele zinnen die de betekenis van de CPL zin uitdrukken; CPL zinnen kunnen dus ambigu zijn. Voor drie talen is er een tool beschikbaar, alleen CLCE heeft geen tool waarin de zinnen kunnen worden ingevoerd. De vier gecontroleerde talen zijn allen gebaseerd op het Engels en ondersteunen geen andere natuurlijke talen.

5.2 Analyse

In deze paragraaf wordt er een analyse gedaan op de onderzoeksresultaten die zijn weergegeven in de bovenstaande paragraaf.

5.2.1 Factoren die invloed hebben op de kwaliteit van de requirements

In het onderzoek zijn er drie onderzoeksactiviteiten uitgevoerd om te achterhalen welke factoren invloed hebben op de kwaliteit van de requirements. Op basis van de vijf criteria is geanalyseerd welke factoren invloed op de requirements hebben.

Eenduidig

Veel van de requirements die worden opgesteld bij VitalHealth zijn dubbelzinnig. De respondenten in het interview hebben allemaal genoemd dat zij in de praktijk requirements tegenkomen die dubbelzinnig zijn. De requirements engineers die de enquête hebben ingevuld zien dat het in de praktijk voorkomt dat zowel de stakeholders als de applicatieontwikkelaars de requirements verkeerd interpreteren. Als oorzaak van dubbelzinnig requirements hebben de respondenten verschillende oorzaken genoemd. In deze paragraaf zullen de oorzaken worden beschreven en op basis van de onderzoeksresultaten worden bevestigd of weerlegd.

Ambiguïteit

In het requirementsdocumenten onderzoek zijn 266 requirements gecontroleerd op ambiguïteit, bij 161 van deze requirements zijn één of meer ambiguïteiten gevonden.
In de requirementsdocumenten is lexicale ambiguïteit het meest (44%) gevonden. Om lexicale ambiguïteit te voorkomen adviseert de IEEE Standaard 830 om een verklarende woordenlijst bij de requirements op te nemen. In de onderzochte requirementsdocumenten was bij geen enkel document een verklarende woordenlijst opgenomen. In één van de documenten was wel een lijst opgenomen met verklaringen van de gebruikte afkortingen. In de interviews geven de respondenten aan dat er alleen verklarende woordenlijsten worden opgenomen in hoofd requirementsdocumenten van een project. In de enquête geven de applicatieontwikkelaar en de applicatietester aan nog nooit een verklarende woordenlijst te hebben gezien in een requirementsdocument. Uit deze reactie blijkt dat er weinig gebruik gemaakt wordt van verklarende woordenlijsten.
Bij semantische en syntactische ambiguïteiten heeft een zin meer dan één betekenis. Na de lexicale ambiguïteit komt deze vorm het meest (27% en 10%) voor in de requirements. In het interview noemen twee respondenten deze vorm van ambiguïteit als oorzaak voor het verkeerd interpreteren van de requirements. In de enquête hebben de respondenten deze typen ambiguïteiten opgegeven als meest voorkomende oorzaak voor dubbelzinnige requirements.
Bij een pragmatische ambiguïteit heeft een zin meer dan één betekenis in de context waarin de zin wordt geuit. Bij een aantal (14%) van de requirements is deze vorm van ambiguïteit gevonden. Een oorzaak van pragmatische ambiguïteit is de manier waarop de requirements worden opgesteld; de requirements worden opgesteld in één stuk beschrijvende tekst waarbij de kans op referentiële ambiguïteit groot is. Respondenten in de enquête geven aan dat requirements die verwarrend zijn in het document minder vaak de oorzaak van dubbelzinnigheid is dan de lexicale, syntactische of semantische ambiguïteit.
Bij een klein (5%) aantal van de requirements zijn taalfout ambiguïteiten gevonden. Het aantal taalfout ambiguïteiten in de requirements is afhankelijk van de opsteller. Een opsteller maakt vaak dezelfde fout in de requirements.

Door de requirements engineers worden er geen activiteiten uitgevoerd om ambiguïteit in de requirements te voorkomen. Applicatieontwikkelaars of -testers die bij een requirements engineer aangeven dat een requirements ambigu is zien zelden dat de requirements engineer het requirement aanpast, de ambiguïteit uit de requirement haalt. De applicatietester komt requirements tegen waarbij de applicatieontwikkelaar al bij de requirements engineer heeft aangegeven dat deze ambigu is.

Requirements incompleet
In de interviews worden incomplete requirements genoemd als oorzaak voor dubbelzinnigheid. Respondenten vinden dat de requirements vaak summier beschreven zijn waardoor deze moeilijk te interpreteren zijn.

Gebrek aan domeinkennis
Drie respondenten noemen in de interviews gebrek aan domeinkennis een oorzaak voor het verkeerd interpreteren van requirements. Domeinkennis wordt door de requirements engineer mondeling of tekstueel, via mail of document, overgedragen op de applicatieontwikkelaar.
Het overdragen van domeinkennis en domein woorden via een verklarende woordenlijst wordt zelden toegepast. Lexicale ambiguïteit is genoemd als oorzaak waardoor een woord in een requirement niet of verkeerd wordt begrepen, maar de oorzaak hiervoor kan ook gebrek aan domeinkennis zijn. In de interviews geven de respondenten aan dat het in de praktijk voorkomt dat woorden in de requirements niet begrepen worden. Ook in de interviews geven de meeste respondenten aan dat het soms voor komt dat een ontwikkelaar een woord niet begrijpt.

Slecht lezen van de requirements
Slecht lezen van de requirements wordt in de interviews genoemd als oorzaak voor het verkeerd interpreteren van een requirement. In de enquête is deze stelling gecontroleerd. De requirements engineers geven in de enquête aan dat zij vinden dat de stakeholders de requirement onnauwkeurig lezen. De respondenten krijgen regelmatig vragen van stakeholders die niet nodig zijn wanneer de requirements nauwkeurig gelezen worden. De mening van de requirements engineers in de enquête is dat applicatieontwikkelaars de requirements niet heel onnauwkeurig, maar ook niet heel nauwkeurig

lezen. Een respondent noemt als oorzaak voor de nauwkeurigheid van lezen de kwaliteit van de requirements. Hoe gedetailleerder de requirements beschreven zijn hoe nauwkeuriger er door de ontwikkelaars gelezen wordt.

Interpretatieproblemen stakeholder en applicatieontwikkelaar
Uit de resultaten blijkt dat stakeholders vaker de requirements verkeerd interpreteren dan applicatieontwikkelaars. Uit de resultaten van de enquête zijn meerdere oorzaken hiervoor af te leiden. De requirements engineers geven aan dat zij bij het opstellen van de requirements meer rekening met de applicatieontwikkelaars houden dan met de stakeholders. Hierdoor kunnen requirements beter te begrijpen zijn door de applicatieontwikkelaars dan door de stakeholders. Ook wordt erdoor de requirements engineer minder vaak met de stakeholder gecommuniceerd dan met de applicatieontwikkelaar. Tijdens de communicatie kunnen er interpretatieproblemen naar boven komen; bij de applicatieontwikkelaars wordt deze dan eerder weggenomen. De requirements engineers geven ook aan dat de stakeholders de requirements minder nauwkeurig lezen.
Samengevat zijn er drie redenen waardoor stakeholders de requirements vaker verkeerd interpreteren: requirements worden opgesteld voor de ontwikkelaars, met de ontwikkelaar wordt meer gecommuniceerd en de ontwikkelaars lezen de requirements nauwkeuriger.

Consistent
Drie van de vijf respondenten die geïnterviewd zijn vinden de requirementsdocumenten niet consistent. De requirements engineers geven als oorzaak voor inconsistente requirements de wensen van de stakeholders die met elkaar conflicteren. Eén requirements engineer kiest ervoor deze op te nemen in de requirements. De andere requirements engineer probeert conflicterende requirements te voorkomen; dit lukt echter niet altijd.

Verifieerbaar
Vier van de vijf respondenten in het interview vinden de requirements verifieerbaar. De applicatietester geeft aan dat door het verbeteren van de kwaliteit van de requirements de applicatie sneller kunnen worden getest.

Traceerbaar
De requirements die onderzocht zijn voldoen niet aan de richtlijnen van de IEEE Standaard 830 voor traceerbaarheid. De requirements zijn namelijk niet achterwaarts traceerbaar, omdat de herkomst per requirement niet te achterhalen is. Ook zijn de requirements niet voorwaarts traceerbaar; bij de requirements zijn relaties naar andere requirements niet inzichtelijk gemaakt. In de interviews bevestigd één van de respondenten dit; er is een onderlinge relatie tussen de requirements, maar dit is niet expliciet in de requirementsdocumenten beschreven.
Om de requirements traceerbaar te maken adviseert de IEEE Standaard 830 om de requirements unieke namen of referentie nummers te geven. In de huidige requirements wordt dit niet gedaan. De huidige requirements worden opgesteld in een stuk tekst waarin meerdere requirements zijn verwerkt. De respondenten geven aan dat deze werkwijze niet hun voorkeur heeft. De respondenten geven de voorkeur aan het opstellen van requirements in een lijst, omdat zij deze vorm overzichtelijker vinden.

Implementatie onafhankelijk
De requirements in de documenten zijn implementatie afhankelijk opgesteld. De respondenten in de interviews bevestigen dit, zij geven allemaal aan dat de requirements niet implementatie onafhankelijk zijn opgesteld.
In het software-ontwikkelproces moeten personen activiteiten uitvoeren die past bij de rol en vaardigheden die zij hebben (Kulak & Guiney, 2004). Uit de interviews blijkt dat de rollen in de praktijk vaak niet goed gescheiden zijn. Requirements engineers met een functionele achtergrond maken technische beslissingen voor een softwareapplicatie.

De requirements in de documenten zijn procedureel beschreven. In de requirements is beschreven hoe de softwareapplicatie moet werken in plaats van wat de softwareapplicatie moet doen. De meeste

respondenten geven de voorkeur aan het procedureel beschrijven van de requirements. In de interviews drie van de vijf en bij de enquête meer dan de helft (61%) van de respondenten.

5.2.2 *Kwaliteitseigenschappen voor een gecontroleerde taal*

In de voorgaande paragraaf zijn de kwaliteitscriteria genoemd. In de enquête hebben de respondenten aangegeven welke criteria zij belangrijk vinden voor de requirements. In tabel 5.26 staan de resultaten die zijn overgenomen uit de resultaten van de enquête. De eigenschappen worden hieronder weergegeven gerangschikt op hoe belangrijk de respondenten deze vinden. Bij de eigenschappen zijn punten opgenomen hoe de specifieke eigenschap verbetert kunnen worden. Deze punten zijn in de vorige paragraaf nader beschreven.

Criteria	**Gemiddelde**	
Eenduidig	*4,40*	*Zeer belangrijk*
Consistent	*4,30*	*Belangrijk*
Verifieerbaar	*4,10*	*Belangrijk*
Traceerbaar	*4,00*	*Belangrijk*
Implementatie onafhankelijk	*3,70*	*Belangrijk*

Tabel 5.26 Ordening van het belang van eigenschappen

In het vervolg van deze paragraaf is per criterium beschreven wat er veranderd moet worden aan de requirements om deze te verbeteren. Voor het criterium verifieerbaar zijn voor de requirements geen verbeter punten gevonden, de respondenten hebben ook aangegeven dat de requirements wel goed verifieerbaar zijn.

<u>Eenduidig</u>

De respondenten vinden eenduidigheid het belangrijkste kwaliteitscriterium voor de requirements. Uit de resultaten van het onderzoek blijkt dat de volgende punten moeten worden toegepast om de kwaliteit van de requirements te verbeteren.

1. Reduceren van linguïstische ambiguïteit in de requirements.
    i. Voorkomen van lexicale ambiguïteit. Hierbij gebruik maken van een verklarende woordenlijst.
    ii. Voorkomen van syntactische ambiguïteit.
    iii. Voorkomen van semantische ambiguïteit.
    iv. Voorkomen van pragmatische ambiguïteit.
    v. Voorkomen van taalfout ambiguïteit.
2. Uitvoeren van activiteiten om ambiguïteit in requirements te voorkomen en op te sporen. Requirements veranderen waarvan de ontwikkelaar of testers aangeeft dat deze ambigu is.
3. Requirements compleet beschrijven. Voorkomen dat door summier beschreven requirements de requirement anders wordt geïnterpreteerd. Gebruik maken van een verklarende woordenlijst om domeinwoorden in de requirements uit te leggen.

<u>Consistent</u>

De respondenten vinden consistentie een belangrijk kwaliteitscriterium voor de requirements. Om de requirements consistenter te maken moeten conflicterende wensen van stakeholders niet in de requirements worden opgenomen.

<u>Traceerbaar</u>

De respondenten vinden traceerbaarheid een belangrijk kwaliteitscriterium voor requirements. Om de kwaliteit van de requirements te verbeteren moeten de volgende punten worden toegepast.

1. Gebruik maken van een unieke naam of nummer voor een requirement waarmee de requirement geïdentificeerd kan worden.
2. Duidelijk onderscheid tussen verschillende requirements maken. In plaats van stukken tekst met daarin de requirements de requirements afzonderlijk opstellen.
3. Bij de requirements de herkomst vermelden.

4. Per requirement de relaties met andere requirements opnemen.

Implementatie onafhankelijk
De respondenten vinden implementatie onafhankelijk het minst belangrijke kwaliteitscriterium voor de requirements. Om beter aan dit criterium te voldoen moeten de volgende zaken worden toegepast.

1. Onderscheid maken tussen het probleem en de oplossing van een softwareapplicatie. Beschrijf in de requirements welk probleem er opgelost moet worden. Beschrijf in het ontwerp hoe het probleem opgelost moet worden.
2. Onderscheid maken in rollen en werkzaamheden

5.2.3 *Gebruiksvriendelijkheid eigenschappen voor een gecontroleerde taal*

Gecontroleerde talen zijn gebaseerd op een natuurlijke taal. Omdat gecontroleerde talen voor deel gebruik maken van dezelfde grammatica en semantiek als natuurlijke talen is het voor een persoon die deze taal beheerst makkelijk om een zin te lezen en te begrijpen.

Om een gecontroleerde taal te leren moet de gebruiker tijd investeren. De geïnterviewde requirements engineers en applicatieontwikkelaars zijn bereid om tijd te investeren in het leren van een gecontroleerde taal. De applicatietester geeft aan dat hij geen tijd wil investeren. Hij vindt dat het leren van een gecontroleerde taal meer bij de personen hoort die de requirements opstellen.
In de interviews wil meer dan de helft (56%) van de respondenten één werkdag investeren om een gecontroleerde taal te leren gebruiken. De meeste respondenten hebben geen ervaring met gecontroleerde of formele talen.
De requirements engineers stellen de requirements op. Voor deze groep is het belangrijk dat de gecontroleerde taal makkelijk te leren en te gebruiken is. De applicatieontwikkelaars, applicatietesters en stakeholders maken van de requirements gebruik, voor deze groep is het belangrijk dat de gecontroleerde taal makkelijk te lezen en te begrijpen is.

In tabel 5.27 is te zien welke eigenschappen de respondenten belangrijk vinden voor een gecontroleerde taal. De eigenschappen komen uit het resultaat van de enquête. De eigenschappen die de respondenten onbelangrijk vinden voor een CNL zijn weggelaten uit tabel 5.27.

Eigenschap	**Gemiddelde**
Verwijzingen	*Belangrijk (4,11)*
Opsommingen	*Belangrijk (3,61)*
Tussenzinnen	*Neutraal (2,82)*
Beeldspraak	*Neutraal (2,72)*
Tool ondersteuning	*Belangrijk (3,5)*
Meertalig	*Neutraal (3,11)*

Tabel 5.27 Belang van taaleigenschappen voor een gecontroleerde taal.

Eigenschappen die respondenten belangrijk vinden worden in de voorgaande paragraaf gekenmerkt als een risico voor ambiguïteit. Eigenschappen zoals verwijzingen, tussenzinnen en beeldspraak vormen een risico voor de kwaliteit van de requirements.

5.2.4 *Prioritering eigenschappenlijst*

In de vorige paragrafen in dit hoofdstuk is beschreven welke factoren invloed op de kwaliteit hebben. Om de kwaliteit van de requirements te verbeteren moeten factoren die een grote invloed op de kwaliteit hebben worden opgelost. Het oplossen kan deels worden gedaan door gebruik te maken van een gecontroleerde taal voor de requirements. In de vergelijking van de gecontroleerde talen is gebruik gemaakt van een eigenschappenlijst. In de eigenschappenlijst staan eigenschappen van gecontroleerde talen die de kwaliteit van de requirements verbeteren, of gebruiksvriendelijk zijn voor het opstellen van de requirements. Op basis van de resultaten van het requirementsdocumenten onderzoek, interviews en enquête wordt in deze paragraaf aangegeven hoe belangrijk eigenschappen zijn. Per eigenschap is aangegeven of deze: zeer belangrijk, belangrijk, neutraal, onbelangrijk of zeer onbelangrijk is om de kwaliteit of de gebruiksvriendelijkheid te verbeteren.

Nauwkeurigheid
In de interviews en enquête hebben de respondenten aangegeven eenduidige requirements zeer belangrijk te vinden. Uit de resultaten van het requirementsdocumenten onderzoek blijkt dat ambiguïteit een grote factor is die de kwaliteit van de requirements beïnvloed.De nauwkeurigheid eigenschappen zijn belangrijk om de kwaliteit van de requirements te verbeteren.

Eigenschap	Belang kwaliteit
Lexicale ambiguïteit	Zeer belangrijk: In de requirements komt deze vorm van ambiguïteit het meest voor. Eén van de oorzaken hiervoor is het ontbreken van een verklarende woordenlijst. Het is erg belangrijk dat bij een CNL taal zelf woorden en beschrijvingen kunnen worden toegevoegd.
Syntactische ambiguïteit	Zeer belangrijk: Na lexicale ambiguïteit komt deze vorm van ambiguïteit het meest voor in de requirements. In de interviews en enquête geven respondenten aan dat de structuur van een zin één van de oorzaken is dat requirements verkeerd worden geïnterpreteerd.
Semantische ambiguïteit	Zeer belangrijk: Deze vorm van ambiguïteit komt niet heel vaak voor. In de interviews en enquête geven respondenten aan dat de structuur van een zin één van de oorzaken is dat requirements verkeerd worden geïnterpreteerd. Dit is ook voor dit type van belang.
Pragmatische ambiguïteit	Belangrijk: Deze vorm van ambiguïteit komt minder voor in de requirements dan lexicale ambiguïteit.

Tabel 5.28 Belang van nauwkeurigheidseigenschappen voor de gecontroleerde taal.

Expressiviteit (Expressiveness)
In de interviews en enquête hebben de respondenten aangeven in welke mate zij linguïstische eigenschappen belangrijk vinden voor het definiëren van requirements. Op basis hiervan is aan de eigenschappen een gewicht gegeven.

Eigenschap	Indicatoren
Maakt de CNL gebruik van een eigen woordenlijst	Belangrijk: Bij het gebruik van een eigen woordenlijst moet de gebruiker niet zelf de betekenis van een woord definiëren, de gebruiker hoeft dan geen rekening met lexicale ambiguïteit te houden.
Kunnen er woorden worden toegevoegd aan de woordenlijst	Zeer belangrijk: Toevoegen van eigen domeinwoorden is zeer belangrijk omdat in de requirements vaak gebruik gemaakt wordt van het jargon van de stakeholder.
Meervoud / enkelvoud	Belangrijk: In de huidige requirements wordt gebruik gemaakt van meervoud, vaak in combinatie met telwoorden.
Heeft de CNL taal regels voor nominalisatie.	Neutraal: Vergroot de expressiviteit van de taal.
Is een andere schrijfwijze van het woord toegestaan.	Belangrijk: In de huidige requirements wordt vaak gebruik gemaakt van afkortingen.
Kunnen er verwijzingen worden gemaakt.	Belangrijk: De respondenten in de enquete hebben aangegeven dat deze eigenschap belangrijk is.
Kunnen er opsommingen worden gemaakt	Belangrijk: De respondenten in de enquete hebben aangegeven dat deze eigenschap belangrijk is.
Zijn tussenzinnen toegestaan.	Neutraal: De respondenten in de enquete hebben aangegeven dat deze eigenschap niet belangrijk maar ook niet onbelangrijk is.
Is beeldspraak toegestaan.	Neutraal: De respondenten in de enquete hebben aangegeven dat deze eigenschap niet belangrijk maar ook niet onbelangrijk is.
Zijn gezegde toegestaan.	Onbelangrijk: De respondenten in de enquete hebben aangegeven dat deze eigenschap onbelangrijk is.
Zijn spreekwoorden toegestaan.	Onbelangrijk: De respondenten in de enquete hebben aangegeven dat deze eigenschap onbelangrijk is.

Tabel 5.29 Belang van expressiviteitseigenschappen voor de gecontroleerde talen.

Natuurlijke karakter (naturalness)
De meeste gebruikers van de requirements hebben geen ervaring met gecontroleerde of formele talen. Gemiddeld willen de gebruikers één werkdag investeren om een CNL taal te leren. Door deze geringe tijd is het belangrijk dat de CNL een natuurlijk karakter heeft. De CNL taal moet snel te leren en te begrijpen zijn. In tabel 5.30 is het gewicht van de eigenschappen aangegeven. Er is onderscheid gemaakt tussen het belang voor de opstellers en de gebruikers van de requirements. De opstellers zijn de requirements engineers en de gebruikers zijn de stakeholders, applicatieontwikkelaars en de applicatietesters. Voor opstellers is het belangrijk dat de gecontroleerde taal eenvoudig te leren en op te stellen is. Voor de stakeholders, applicatieontwikkelaars en applicatietesters is het belangrijk dat de CNL taal eenvoudig te lezen en te begrijpen is.

Eigenschap	Opsteller van de requirements	Gebruiker van de requirements
Is de taal makkelijk te leren?	Zeer belangrijk	Belangrijk
Is de taal makkelijk te lezen?	Belangrijk	Zeer belangrijk
Is de taal makkelijk te schrijven?	Zeer belangrijk	Belangrijk
Is de taal makkelijk te begrijpen?	Belangrijk	Zeer belangrijk

Tabel 5.30 Belang van natuurlijke karakter voor de gecontroleerde taal.

Eenvoud (Simplicity)
De eigenschappen voor eenvoud zijn belangrijk voor de gebruiksvriendelijkheid en de kwaliteit. In tabel 5.31 staan de eigenschappen die bij eenvoud horen.

Eigenschap	Belang voor gebruiksvriendelijkheid
Kunnen de zinnen van de CNL taal eenvoudig worden omgezet naar een formele taal.	Neutraal: Deze eigenschap is belangrijk wanneer de requirements omgezet moeten worden naar computer code. *(Gebruiksvriendelijkheid)*
Wordt de discours omgezet naar een formele taal.	Belangrijk: Voor het voorkomen van pragmatische ambiguïteit en software engineerings ambiguïteit. *(Kwaliteit)*
Is er een mapping naar een grafische presentatie mogelijk.	Neutraal: In de interviews heeft één respondent aangeven dat hij dit belangrijk vindt. *(Gebruiksvriendelijkheid)*
Ondersteunt de CNL taal meerder natuurlijke talen.	Neutraal: De respondenten hebben aangegeven dat zij dit niet belangrijk vinden. *(Gebruiksvriendelijkheid)*
Beschikt de CNL over een style richtlijn	Neutraal: De respondenten in de interviews vonden het niet belangrijk maar ook niet onbelangrijk dat er een handleiding voor de taal beschikbaar is waar in beschreven is hoe het beste zinnen kunnen worden opgesteld. *(Gebruiksvriendelijkheid)*
Wordt de CNL taal ondersteunt door een tool?	Belangrijk: De ondersteuning van de CNL door een tool vinden de respondenten belangrijk. *(Gebruiksvriendelijkheid)*

Tabel 5.31 Belang vaan eenvoudseigenschappen voor de gecontroleerde taal.

5.2.5 *Selectie van de gecontroleerde taal*
Op basis van de eigenschappenlijst zijn de CNL talen met elkaar vergeleken. In de vorige paragraaf is beschreven hoe de gecontroleerde talen geordend kunnen worden op kwalitatief en gebruiksvriendelijk. In deze paragraaf worden de score van de talen per onderwerp (nauwkeurig, expressief, natuurlijk, eenvoudig) weergegeven.
Kuhn (2012) heeft 95 verschillende CNL talen onderzocht, waaronder de talen die in dit onderzoek onderzocht zijn. Kuhn heeft de talen ook per onderwerp een score gegeven. Bij de beschrijvingen van de resultaten zullen zijn resultaten per onderwerp worden beschreven.

In tabel 5.32 is te zien hoe de gecontroleerde talen gescoord hebben op nauwkeurigheid. In de tabel is de kwaliteitsscore te zien. Hoe hoger de score is hoe meer de kwaliteit van de requirements wordt verbeterd door de gecontroleerde taal.

	ACE	PENG	CLCE	CPL
Score kwaliteit	*16,5*	*16,5*	*16,5*	*14*

Tabel 5.32 Score op nauwkeurigheid per gecontroleerde taal.

Uit de resultaten blijkt dat de talen ACE, PENG en CLCE het meest nauwkeurig zijn. In deze talen wordt ambiguïteit het minst toegestaan. De resultaten komen overeen met wat Clark (2010) beschrijft over de twee *"naturalist"* en *"formalist"* filosofieën. In de resultaten is te zien dat de *"naturalist"* talen (CPL) meer ambiguïteit toelaten. Bij de vergelijking van Kuhn hebben de talen PENG en CLCE beide de maximale vijf gescoord op nauwkeurigheid, ACE heeft vier punten en CPL heeft drie punten.

In tabel 5.33 is te zien hoe de talen gescoord hebben op expressiviteit, de taalconstructie die mogelijk zijn in de taal.

	ACE	PENG	CLCE	CPL
Score gebruiksvriendelijkheid	*21*	*19*	*19*	*17*

Tabel 5.33 Score op expressiviteit per gecontroleerde taal.

ACE heeft het hoogst gescoord op expressiviteit. Dit komt niet omdat er in ACE meer taalconstructie mogelijk zijn dan in de vier andere talen, maar omdat ACE beschikt over eigenschappen die belangrijk zijn voor de praktijk, de eigenschappen die de respondenten als belangrijk hebben geclassificeerd. In het onderzoek van Kuhn hebben al de vier de talen drie punten gescoord op expressiviteit. Talen die minder formeel zijn scoren in het onderzoek van Kuhn hoger op expressiviteit.

In tabel 5.34 is te zien hoe de talen gescoord hebben op natuurlijk karakter. In de resultaten is een splitsing gemaakt tussen twee groepen; requirements opsteller en de requirements gebruikers. De opstellers zijn de requirements engineers en de gebruikers zijn de stakeholders, applicatieontwikkelaars en de applicatietesters. Voor de opstellers is het belangrijk dat de taal makkelijk te leren en te gebruiken is. Voor de gebruikers van de requirements is het belangrijk dat de taal makkelijk te lezen en te begrijpen is.

	ACE	PENG	CLCE	CPL
Score gebruiksvriendelijkheid opsteller	*15,5*	*18*	*8*	*8,5*
Score gebruiksvriendelijkheid gebruiker	*16*	*18*	*10*	*12*

Tabel 5.34 Score op natuurlijk karakter per gecontroleerde taal.

PENG heeft het meest gescoord op natuurlijk karakter. Dit komt mede door de tool die ontwikkeld is voor PENG. Met de tool van PENG wordt een gebruiker door de regels van PENG heen geleid terwijl de gebruiker de PENG zinnen aan het opstellen is. Hierdoor is het voor een gebruiker niet nodig om uitgebreide kennis van de regels van PENG te hebben. In het onderzoek van Kuhn hebben de vier talen een vier gescoord op natuurlijk karakter.

In tabel 5.35 is te zien hoe de talen gescoord hebben op eenvoud. In de eigenschappen van eenvoud was onderscheid tussen kwaliteit en gebruiksvriendelijkheid.

	ACE	PENG	CLCE	CPL
Score kwaliteit	*2*	*2*	*2*	*2*
Score gebruiksvriendelijkheid	*8,5*	*11*	*6*	*9,5*

Tabel 5.35 Score op eenvoud per gecontroleerde taal.

Bij eenvoud heeft PENG weer de hoogste score. Ook in dit geval heeft PENG zich onderscheiden door de tool die beschikbaar is voor PENG. In het onderzoek van Kuhn hebben de eerste drie talen (ACE, PENG en CLCE) drie punten gescoord op eenvoud, CPL heeft twee punten gescoord. Kuhn heeft bij eenvoud alleen onderzocht hoe eenvoudig het is om zinnen van gecontroleerde talen om te zetten naar formele talen. In dit onderzoek zijn aan eenvoud nog andere eigenschappen toegevoegd, hierdoor komen de resultaten van eenvoud van Kuhn niet overeen met de resultaten van dit onderzoek.

In tabel 5.36 is de totaal score te zien. De score van de vier onderwerpen is bij elkaar opgeteld gescheiden in kwaliteit en gebruiksvriendelijkheid. Omdat bij natuurlijk karakter een onderscheid is gemaakt tussen requirements opstellers en requirements gebruikers is dit ook terug te zien in het eindresultaat.

	ACE	**PENG**	**CLCE**	**CPL**
Score kwaliteit	*18,5*	*18,5*	*18,5*	*16*
Score gebruiksvriendelijkheid opsteller	*45*	*48*	*33*	*37,5*
Score gebruiksvriendelijkheid gebruiker	*44,5*	*48*	*35*	*38,5*

Tabel 5.36 Totaal score op kwaliteit en gebruiksvriendelijkheid per gecontroleerde taal.

Op kwaliteit score de eerste drie talen (ACE, PENG en CLCE) gelijk. Op gebruiksvriendelijkheid scoort de gecontroleerde taal PENG het best. De gecontroleerde talen ACE en PENG lijken in opbouw van de grammatica en semantische regels veel op elkaar. Voor PENG is een gebruiksvriendelijke tool ontwikkeld waardoor het niet nodig is dat een gebruiker kennis van de regels heeft. Hierdoor heeft de taal PENG beter gescoord op gebruiksvriendelijkheid dan ACE.

6. Conclusie en discussie

In dit hoofdstuk worden de onderzoeksvragen beantwoord. Iedere onderzoeksvraag wordt behandeld door eerst de conclusie te geven waarna een korte discussie volgt. De antwoorden van de deelvragen worden gecombineerd om als laatste de onderzoeksvraag te beantwoorden.

Hoe kan ambiguïteit op een betrouwbare manier worden opgespoord in de requirements?
In het onderzoek is een checklist ontwikkeld om ambiguïteiten in requirements op te sporen. Met de checklist kunnen op een betrouwbare manier linguïstische ambiguïteiten worden opgespoord, mits dit gedaan wordt door meerdere personen. Met de checklist vinden meer personen meer ambiguïteiten in de requirements; in twee requirements worden ongeveer vijf ambiguïteiten gevonden door vier personen en door één persoon worden er maar twee gevonden.
Met de checklist zijn in de requirements ambiguïteiten gevonden van al de vijf verschillende typen linguïstische ambiguïteit. De software engineerings ambiguïteiten zijn veel minder gevonden met de checklist. Uit het onderzoek blijkt dan ook dat voor het opsporen van software engineerings ambiguïteiten de checklist minder toereikend is.

Zoals uit de resultaten blijkt is het lastig om alle ambiguïteiten in de requirements op te sporen. Dit komt omdat een menselijke lezer snel over ambiguïteiten in een zin heen leest. Een lezer neemt de eerste interpretatie die hem te binnen schiet aan als de juiste betekenis van een zin (Popescu et al., 2008). Het opsporen van ambiguïteiten in de requirements is hierdoor moeilijk; de reviewer is zich meestal onbewust van een andere mogelijke betekenis van de requirement. Het laten reviewen van de requirements door meerdere stakeholders vinden Gause en Weinberg (1989) de meest effectieve manier om ambiguïteiten in requirements te vinden. Met deze methode wordt er per requirement aan de stakeholder gevraagd welke interpretatie hij aan het requirement zou geven. Wanneer de interpretaties van de stakeholders van elkaar verschillen dan is de requirement ambigu. Tijdens de softwareontwikkeling zal deze methode om economische en praktische redenen niet haalbaar zijn. In de resultaten van het onderzoek is terug te zien dat personen snel over ambiguïteiten in de requirements heen lezen. Gemiddeld wordt bijna de helft (47%) van de ambiguïteiten die een respondent vindt, niet gevonden door de andere drie respondenten. De reden die de respondenten geven waarom ze ambiguïteiten niet vinden in de requirements is omdat zij over de ambiguïteit heen lezen.

De checklist is gemaakt op basis van de checklist van Kamsties (2001). In de checklist van Kamsties ontbrak echter één van de linguïstische ambiguïteiten. Het type syntactische ambiguïteit kwam niet voor in de checklist. Kamsties heeft dit type ambiguïteit niet in de checklist opgenomen omdat deze volgens hem niet vaak voorkomt in de requirements. Chantree en Nuseibeh (2006) vinden dit één van de nadelen van de checklist van Kamsties; de checklist methode van Kamsties spoort niet expliciet de syntactische ambiguïteiten in de requirements op. Chantree en Nuseibeh zien dat syntactische ambiguïteit wel voorkomt in de requirements. Ook Sawyer (2005) noemt syntactische ambiguïteit als duidelijke oorzaak voor ambiguïteit in requirements.
In dit onderzoek is bijna een derde (27%) van het totaal aantal gevonden linguïstische ambiguïteiten van het type syntactische ambiguïteit. Hieruit blijkt dat voor het betrouwbaar opsporen van de ambiguïteiten de syntactische ambiguïteit niet gemist mag worden.

De meest toegepast techniek om ambiguïteiten in de requirements te detecteren is de checklist techniek. Kamsties heeft daarom ook voor deze techniek gekozen. De checklist techniek is goed bruikbaar voor het controleren van ambiguïteiten die binnen de requirement voorkomen; zoals

lexicale en syntactische ambiguïteit. De checklist is minder bruikbaar om discourse ambiguïteiten mee te vinden.Vooral bij requirements die refereren naar een requirement op een andere pagina is het niet makkelijk om met de checklist te controleren of de referentie ambigu is. Bijna driekwart (71%) van de gevonden linguïstische ambiguïteiten is van het type lexicale of syntactische ambiguïteit. Deze twee typen ambiguïteiten komen voor binnen de context van een zin. Een type syntactische ambiguïteit is de coördinatie ambiguïteit. In 6,01% van de requirements is deze vorm van ambiguïteit gevonden. Dit percentage komt ongeveer overeen met de resultaten van Yang et al. (2010). Yang heeft in 12,68% van de requirements die hij onderzocht heeft coördinatie ambiguïteiten gevonden.
Semantische en pragmatische ambiguïteiten zijn veel minder (24%) gevonden in de requirements. Deze typen ambiguïteiten zijn meer van toepassing op de context waarin de zin staat. De oorzaak voor dit verschil in verhouding tussen de ambiguïteitstypen kan veroorzaakt worden doordat met de checklist beter ambiguïteiten opgespoord kunnen worden binnen de context van een requirement.

Welke factoren beïnvloeden de kwaliteit van de requirements?
In dit onderzoek is gebleken dat de requirements van VitalHealth niet aan de kwaliteitscriteria eenduidig, traceerbaar, consistent en implementatie onafhankelijk voldoen.
Uit de resultaten van het onderzoek blijkt dat de requirements niet eenduidig zijn. Veel (61%) van de onderzochte requirements zijn namelijk ambigu. De respondenten in het onderzoek vinden de requirements vaak dubbelzinnig en incompleet, hierdoor ontstaan er interpretatieproblemen. In het onderzoek is gebleken dat interpretatieproblemen niet alleen door de slechte kwaliteit van de requirements ontstaan, maar ook door slecht lezen en gebrek aan domeinkennis.
De onderzochte requirements bleken ook niet goed traceerbaar te zijn. De herkomst van de requirements wordt niet gedocumenteerd en de relaties tussen de requirements worden niet vastgelegd in het requirementsdocument. De respondenten vinden de requirements wel goed traceerbaar, maar geven daarbij wel aan dat de onderlinge relaties van de requirements niet in de documenten worden beschreven. De requirements zijn ook niet identificeerbaar, doordat een unieke naam of nummer ontbreekt.
In dit onderzoek geven de requirements engineers aan dat de requirements niet consistent zijn. De oorzaak hiervoor is vaak het opnemen van conflicteerde wensen van stakeholders in de requirements. In de requirements worden vaak ontwerp- en implementatiedetails opgenomen. Meestal wordt er ook geen onderscheid gemaakt tussen het functioneel ontwerp en de requirements. Hierdoor zijn de requirements niet implementatie onafhankelijk opgesteld.
De meeste respondenten vinden de requirements goed verifieerbaar. De applicatietester vinden de softwareapplicaties goed te testen op basis van de requirements.

In het theoretische kader zijn er drie oorzaken, opgesteld door Kamsties (2005), beschreven waardoor requirements dubbelzinnig worden. Eén van deze oorzaken, conflicterende individuele meningen, komt in de resultaten van het onderzoek niet naar voren. De twee andere oorzaken zijn wel in dit onderzoek gevonden; dit zijn: requirements die incompleet zijn en de representatie vorm van de requirements (ambiguïteit door gebruik van natuurlijke talen). De requirements binnen VitalHealth worden opgesteld in een natuurlijke taal; dit is de oorzaak waardoor veel requirements ambigu zijn. Het type lexicale ambiguïteit is van de linguïstische ambiguïteiten het meest gevonden. De oorzaak waarom lexicale ambiguïteit zo vaak voorkomt is het ontbreken van een verklarende woordenlijst in de requirementsdocumenten; bij VitalHealth worden er zelden verklarende woordenlijsten opgenomen in een document. De IEEE Standaard 830 adviseert dan ook om verklarende woordenlijsten op te nemen om lexicale ambiguïteit in requirements te voorkomen.
Het risico op interpretatieproblemen is niet bij elk type linguïstische ambiguïteit even hoog. De betekenis van het requirement waarin een linguïstische ambiguïteit voorkomt kan in veel gevallen worden afgeleid uit de context waarin de requirement staat, of door de domeinkennis van de lezer. Berry en Kamsties (2003) merken op dat er geen requirements zijn die niet ambigu zijn; er is namelijk altijd iemand die het requirement op een andere manier interpreteert dan iemand anders. De respondenten hebben in het onderzoek syntactische en semantische ambiguïteit als grootste oorzaak opgegeven waardoor requirements verkeerd worden geïnterpreteerd. De respondenten geven aan dat dit vaker de oorzaak van interpretatieproblemen is dan lexicale ambiguïteit. Lexicale ambiguïteit, die het meest gevonden is in de requirementsdocumenten, wordt door de respondenten minder vaak gezien als oorzaak van dubbelzinnigheid dan syntactische en semantische ambiguïteit. Lexicale

ambiguïteit vormt een minder groot interpretatieprobleem dan syntactische en semantische ambiguïteit, de betekenis van een dubbelzinnig woord kan makkelijker vanuit de context van de requirement of door domeinkennis worden achterhaald dan de betekenis van een requirement die dubbelzinnig is.
Kamsties (2000) noemt software engineerings ambiguïteit de meest voorkomende ambiguïteit in de requirementsdocumenten. De respondenten van dit onderzoek zien software engineerings ambiguïteiten (requirement is ambigu in document of domein) minder vaak als oorzaak voor dubbelzinnigheid dan de linguïstische ambiguïteiten.
Het gebrek aan domeinkennis en het slecht lezen van de requirements zijn ook oorzaken waardoor er interpretatieproblemen ontstaan. In de resultaten van zijn onderzoek heeft Philippo (2013) een aantal factoren beschreven die invloed hebben op het verkeerd interpreteren van de requirements. Eén van de factoren die Philippo noemt is het gebrek aan domeinkennis. Wanneer een projectlid niet het domein en de terminologie daarvan begrijpt, dan is de kans op verkeerde interpretatie van woorden in een requirement groot. Door gebruik te maken van een verklarende woordenlijst kan de requirements engineer dit voorkomen; in de lijst moet worden aangeven wat domeinwoorden betekenen. Requirements engineers hebben in dit onderzoek aangegeven dat de requirements door stakeholders onnauwkeurig worden gelezen en door de applicatieontwikkelaars niet erg nauwkeurig. In dit onderzoek heeft een respondent aangegeven dat de gedetailleerdheid van de requirements invloed heeft op de nauwkeurigheid van lezen. Requirements die getailleerd zijn opgesteld worden door de applicatieontwikkelaars nauwkeuriger gelezen. Hieruit blijkt dat wanneer de requirements minder ambigu en completer worden opgesteld de stakeholders en de applicatieontwikkelaars de requirements beter lezen en minder last van gebrek aan domeinkennis hebben.

In het theoretische kader zijn een aantal redenen, van Kulak en Guiney (2004), beschreven waarom het belangrijk is om geen ontwerp- en implementatiedetails op te nemen in de requirements. Ten eerste vinden zij dat personen activiteiten moeten uitvoeren die passen bij de rol en vaardigheden die zij hebben. Uit interviews met de respondenten bleek dat de requirements engineers ook technische beslissingen nemen voor de softwareapplicaties. Omdat hun kennis hiervoor niet toereikend is doen zij dit in overleg met de applicatieontwikkelaars. Ten tweede vinden Kulak en Guiney dat het opstellen van de requirements het begrijpen en documenteren van het probleem is. In requirements waarin beschreven wordt hoe de softwareapplicatie moet gaan werken, wordt vaak voorbij gegaan aan het onderliggende probleem. Easterbrook (2004) bevestigt dit; wanneer er direct wordt begonnen aan het ontwerpen van de applicatie kunnen dingen gemist worden die bij een grondige probleemanalyse wel naar boven gekomen waren.

<u>Wat zijn de wensen voor de gebruiksvriendelijkheid van een gecontroleerde natuurlijke taal?</u>
In het onderzoek is gebleken dat de betrokkenen van de requirements een gecontroleerde taal willen die eenvoudig te gebruiken is. De taal moet voor de requirements engineers eenvoudig te leren en te schrijven zijn, en voor de voor de overige betrokkenen eenvoudig te lezen en te begrijpen. De gecontroleerde taal moet expressief genoeg zijn om op een natuurlijke wijze requirements op te stellen, maar de betrokkenen vinden het niet belangrijk dat de gecontroleerde taal ook figuurlijke taal (beeldspraak, gezegden, spreekwoorden, etc) ondersteunt. De betrokkenen vinden het belangrijk dat er een tool beschikbaar is waarmee zij met de gecontroleerde taal requirements kunnen opstellen. De meeste gecontroleerde talen ondersteunen alleen het Engels. De betrokkenen vinden het niet belangrijk dat de gecontroleerde taal ook andere talen ondersteunt.

In de requirements die onderzocht zijn in dit onderzoek kwam beeldspraak voor. Het gebruik van beeldspraak kan de communicatie tussen partijen verbeteren en kan helpen problemen inzichtelijker te maken. Het voordeel van beeldspraak is dat deze geschikt zijn om complexe abstracte concepten gedeeltelijk voor leken begrijpelijk te maken (Visscher, Kats, Terwijn, Arnoldus, & van den Berg, 2005). Beeldspraak kan de communicatie tussen de requirements engineer en de stakeholders vergemakkelijken. Mahemoff en Johnston (1998) geven echter aan dat het gebruik van beeldspraak ook problemen kan opleveren doordat een lezer de beeldspraak onjuist interpreteert. Dit kan doordat een lezer niet bekend genoeg is met de beeldspraak omdat deze bijvoorbeeld cultureel- of bedrijfsgebonden is. Uit de resultaten van dit onderzoek blijkt dat betrokkenen van de requirements beeldspraak niet belangrijk vinden voor de requirements. Eén van de respondenten heeft daarbij ook

aangegeven dat beeldspraak problemen kan opleveren tussen culturen. Uit de resultaten van dit onderzoek blijkt dat figuurlijke taal vaak niet is toegestaan in gecontroleerde talen. Taal eigenschappen zoals beeldspraak, gezegde en spreekwoorden worden niet ondersteunt. Door gebruik te maken van figuurlijke taal in requirements worden deze snel dubbelzinnig. Dit komt omdat figuurlijke taal is in zichzelf ambigu is. Figuurlijke taal heeft meerder betekenissen; een letterlijke en een figuurlijke. Het gebruik van figuurlijke taal heeft dus invloed op de kwaliteit van de requirements.

Welke geselecteerde gecontroleerde natuurlijke taal heeft de juiste eigenschappen om requirements mee te definiëren?
In het onderzoek is de gecontroleerde taal PENG gekozen als de beste van de vier geselecteerde talen. Met PENG kunnen op een gebruiksvriendelijke manier kwalitatieve requirements worden opgesteld. Met PENG wordt de kwaliteit van de requirements op dezelfde manier verbetert als de talen ACE en CLCE, maar PENG is de gebruikersvriendelijkste van deze twee talen.

PENG heeft een eigen lexicon om lexicale ambiguïteit in zinnen te verminderen. De lexicon bestaat uit voor gedefinieerde woorden en domeinwoorden; de domeinwoorden worden door de gebruiker toegevoegd aan PENG door aan de lexicon een nieuw woord en de betekenis toe te voegen. Hierdoor beperkt de gebruiker de betekenis van een woord. Zoals eerder beschreven vinden Gruzitis en Barzdins (2010) het probleem van talen zoals PENG en ACE dat de lexicon puur syntactische is, de woorden hebben geen betekenis of interpretatie. Het is aan de gebruiker van PENG om te definiëren welke betekenis een woord heeft. De gebruiker weet echter niet al de mogelijke betekenissen van een woord. Een gebruiker kan een woord aan de lexicon toevoegen die lexicaal ambigu is, ondanks dat een gebruiker een betekenis aan het woord geeft. Om lexicale ambiguïteit in PENG te voorkomen moet de gebruiker er zich van bewust zijn dat de ingevoerde woorden meer dan één betekenis kunnen hebben.
PENG heeft grammaticaregels om ambiguïteit in zinnen te voorkomen. Zinnen die in PENG grammaticaal juist opgesteld zijn, kunnen echter toch nog ambigu zijn. Om van deze zinnen de betekenis te bepalen heeft PENG ook regels voor de semantiek van een zin. Met de regels voor grammatica en semantiek worden syntactische, semantische en pragmatische ambiguïteiten in PENG voorkomen. PENG heeft zelf geen kennis van het applicatiedomein, van de software engineerings methode of van de wereld in het algemeen (Fuchs et al., 1990). Bij PENG worden door de regels op woordenschat, grammatica en semantiek de betekenis van een zin bepaalt. Hierdoor wordt software engineerings ambiguïteit met een CNL taal niet expliciet voorkomen.

Hoofdvraag: “Waar moet een gecontroleerde natuurlijke taal aan voldoen om de kwaliteit van software requirements in de praktijk op een gebruiksvriendelijke manier te verbeteren.”
Uit het onderzoek blijkt dat de requirements op vier kwaliteitscriteria moeten worden aangepast om de kwaliteit te verbeteren. De betrokkenen van de requirements vinden eenduidigheid het meest belangrijke criterium welke verbeterd moet worden. In de requirements komt veel ambiguïteit voor; om de requirements eenduidiger te maken moet ambiguïteit voorkomen worden. Om de requirements te verbeteren moet de gecontroleerde taal ambiguïteit in requirements voorkomen.
In het onderzoek is gebleken dat betrokkenen van de requirements een gecontroleerde taal willen die eenvoudig te gebruiken is. Daarnaast moet de gecontroleerde taal expressief genoeg zijn om requirements op een natuurlijke wijze op te stellen.
In dit onderzoek is de taal PENG uit vier gecontroleerde talen geselecteerd als beste om op een gebruiksvriendelijke manier kwalitatieve requirements op te stellen.

Met een gecontroleerde taal worden ambiguïteiten in requirements voorkomen. Om te laten zien dat dit ook daadwerkelijk werkt, zijn er bij de afronding van dit onderzoek zes requirements omgezet naar een gecontroleerde taal. De zes requirements komen uit requirementsdocumenten die al eerder zijn onderzocht. De requirements zijn omgezet naar ACE. Er is voor ACE gekozen omdat hier een online tool voor beschikbaar is. De syntax van ACE en PENG lijken veel op elkaar. Om ambiguïteit te voorkomen worden in de talen ongeveer dezelfde regels gebruikt. In tabel 6.1 is te zien hoeveel ambiguïteiten voorkwamen in het originele requirement en hoeveel ambiguïteiten er voorkomen in het geherformuleerde requirement. In bijlage *“E Requirements in een gecontroleerde natuurlijke taal”* zijn

de uitgewerkte requirements te zien. Alle acht ambiguïteiten die voorkwamen konden met ACE worden opgelost.

Ambiguïteit	Origineel	Geherformuleerd
Lexicale ambiguïteit	*1*	*0*
Syntactische ambiguïteit	*5*	*0*
Semantische ambiguïteit	*0*	*0*
Pragmatische ambiguïteit	*2*	*0*
Totaal	**8**	**0**

Tabel 6.1 Ambiguïteit in requirements die omgezet zijn naar een gecontroleerde taal.

Met een gecontroleerde taal wordt de kwaliteit van requirements verbetert door deze eenduidiger te maken. Uit de resultaten van dit onderzoek blijken er ook nog drie andere kwaliteitscriteria te zijn waarop de requirements verbeterd kunnen worden. In deze discussie zullen deze criteria kort genoemd worden.
Door geen conflicterende wensen van stakeholders op te nemen in de requirements zal de consistentie van de requirements verbeteren. De traceerbaarheid van de requirements kan verbeterd worden door de herkomst en de relaties per requirement te beschrijven in het requirementsdocument. De requirements moeten ook beter identificeerbaar worden; dit kan gedaan worden door de requirements unieke namen of nummers te geven. In het onderzoek vinden de respondenten implementatie onafhankelijk de minst belangrijke eigenschap voor requirements. Om requirements meer implementatie onafhankelijk te maken moet er onderscheid worden gemaakt tussen het probleem en de oplossing van de softwareapplicatie. Ook moet er meer onderscheid worden gemaakt in de werkzaamheden van de personen die betrokken zijn bij de requirements. De requirements engineer moet het probleem beschrijven, en in een functioneel ontwerp beschrijven hoe dit opgelost moet worden. De applicatieontwikkelaars moeten in het ontwerp de technische beslissingen nemen.

Dit onderzoek is uitgevoerd bij het softwarebedrijf VitalHealth. Het bedrijf bestaat nog maar vijf jaar. Pas de laatste jaren worden er activiteiten ondernomen om de kwaliteit van bedrijfsprocessen te verbeteren. Hierdoor zijn veel processen nog niet gestandaardiseerd zoals bij softwarebedrijven die al jaren bestaan. Het requirements engineerings proces is ook nog niet gestandaardiseerd. Veel wordt er nog ad-hoc ondernomen, en voor het opstellen van requirements zijn er nog geen richtlijnen opgesteld. Daarnaast hebben veel van de requirements engineers geen informatica achtergrond. Requirements engineers hebben bij ziekenhuizen of zorginstellingen gewerkt en hebben daardoor wel veel domeinkennis, maar minder ervaring met het opstellen van software requirements. Dit is waarschijnlijk ook één van de oorzaken waardoor er zoveel linguïstisch ambiguïteiten in de requirements voorkomen.
Om beter vast te stellen hoe vaak linguïstische ambiguïteiten gemiddeld voorkomen in de requirementsdocumenten zullen er ook documenten van andere bedrijven onderzocht moeten worden. Bij deze onderzoeken kan gebruik worden gemaakt van dezelfde technieken om linguïstische ambiguïteiten op te sporen. In dit onderzoek is voornamelijk onderzocht hoe gecontroleerde talen de requirements eenduidiger kunnen maken. Bij nader onderzoek is het mogelijk om te onderzoeken of een gecontroleerde taal ook invloed heeft op andere kwaliteitscriteria zoals consistent en verifieerbaar. De enquête en interviews zijn afgenomen bij medewerkers van VitalHealth, om het onderzoek breder te maken is het nodig om bij stakeholders en medewerkers van andere bedrijven ook een interview en enquête af te nemen om de wensen voor een gebruiksvriendelijke gecontroleerde taal in kaart te brengen. Bij een breder onderzoek kunnen dan tegelijk ook meerdere gecontroleerde talen worden onderzocht. In dit onderzoek zijn vier van de minstens 95 bestaande gecontroleerde talen onderzocht. Met de ontwikkelde eigenschappenlijsten kunnen meerdere gecontroleerde talen worden onderzocht op eigenschappen die kwaliteit van requirements verbeteren en gebruiksvriendelijk zijn in het opstellen van requirements. In dit onderzoek zijn de gecontroleerde talen niet getest in de praktijk. Een methode om dit in een vervolgonderzoek te doen is met een workshop. In een workshop kunnen aantal talen door gebruikers uit de praktijk worden getest om te onderzoeken of met de taal op een gebruiksvriendelijke wijze kwalitatieve requirements kunnen worden opgesteld. Door één requirement om te zetten naar meerdere gecontroleerde talen kan worden vergeleken welke taal in de praktijk het gebruikersvriendelijkst is en de beste kwaliteit op levert.

Bibliografie

Al-Rawas, A., & Easterbrook, S. (1996). Communication problems in requirements engineering: A field study. *First Westminster Conference on Professional Awareness in Software Engineering*, 46-60.

Berry, D. (2008). Ambiguity in natural language requirements documents. In B. Paech & C. Martell (Eds.), *Lecture Notes in Computer Science: Innovations for requirement analysis. From stakeholders needs to formal designs* (Vol. 5320, pp. 1-7). Berlin Heidelberg: Springer. doi: 10.1007/978-3-540-89778-1_1

Berry, D. M., Kamsties, E., & Krieger, M. M. (2003). *From contract drafting to software specification: Linguistic sources of ambiguity, A handbook* (pp. 1-80). Waterloo: University of Waterloo. Retrieved from http://se.uwaterloo.ca/~dberry/handbook/ambiguityHandbook.pdf

de Bruijn, F., & Dekkers, H. (2010). Ambiguity in natural language software requirements: A case study. In R. Wieringa & A. Persson (Eds.), *Lecture Notes in Computer Science: Requirements engineering: Foundation for software quality* (Vol. 6182, pp. 233-247). Springer. doi: 10.1007/978-3-642-14192-8_21

Chantree, F., Nuseibeh, B., De Roeck, A., & Willis, A. (2006). Identifying nocuous ambiguities in natural language requirements. *Requirements Engineering, 14th IEEE International Conference*, 59-68. doi:10.1109/RE.2006.31

Clark, P., Chaw, S. Y., Barker, K., Chaudhri, V., Harrison, P., Fan, J., . . . Thompson, J. (2007). Capturing and answering questions posed to a knowledge-based system. In *K-CAP '07: International conference on knowledge capture: Proceedings of the 4 th international conference on knowledge capture* (Vol. 28, pp. 63-70). Whistler, BC, Canada: ACM. doi:10.1145/1298406.1298419

Clark, P., Harrison, P., Jenkins, T., Thompson, J., & Wojcik, R. (2005). Acquiring and using world knowledge using a restricted subset of english. In *FLAIRS 2005: .* AAAI.

Clark, P., Murray, W., Harrison, P., & Thompson, J. (2010). Naturalness vs. Predictability: A key debate in controlled languages. In N. Fuchs (Ed.), *Lecture Notes in Computer Science: Controlled natural language* (Vol. 5972, pp. 65-81). Springer Berlin Heidelberg. doi:10.1007/978-3-642-14418-9_5

Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM*, *31*(11), 1268-1287. doi:10.1145/50087.50089

Easterbrook, S. M. (2004). What are requirements? Retrieved from http://www.cs.toronto.edu/~sme/papers/2004/FoRE-chapter02-v7.pdf

Ernst & Young. (2011). ICT barometer gezondheidszorg.

Fabbrini, F., Fusani, M., Gnesi, S., & Lami, G. (2001). The linguistic approach to the natural language requirements quality: Benefit of the use of an automatic tool. In *SEW '01: Proceedings of the 26th annual NASA goddard software engineering workshop* (pp. 97-105). IEEE Computer Society. doi: 10.1109/SEW.2001.992662

Fuchs, N., Höfler, S., Kaljurand, K., Rinaldi, F., & Schneider, G. (2005). Attempto controlled english: A knowledge representation language readable by humans and machines. In N. Eisinger & J. Maluszynski (Eds.), *Lecture Notes in Computer Science: Reasoning web* (Vol. 3564, pp. 213-250). Springer Berlin / Heidelberg. Retrieved from http://dx.doi.org/10.1007/11526988_6

Fuchs, N., Kaljurand, K., & Kuhn, T. (2008). Attempto controlled english for knowledge representation. In C. Baroglio,, P. Bonatti,, J. Małuszyński, M. Marchiori, A. Polleres, & S. Schaffert (Eds.), *Lecture Notes in Computer Science: Reasoning web* (Vol. 5224, pp. 104-124). Springer Berlin Heidelberg. doi:10.1007/978-3-540-85658-0_3

Fuchs, N., Kaljurand, K., Kuhn, T., & Schneider, G. (2006). Attempto controlled english and the semantic web. University of Zurich .

Fuchs, N. E., & Schwitter, R. (1996). Attempto controlled english (ace). In *First international workshop on controlled language applications* (pp. 124-136). Universiteit van Leuven, België.

Fuchs, N. E., Kaljurand, K., & Kuhn, T. (2009). ACE can be described by itself. In S. Clematide, M. Klenner, & M. Volk (Eds.), *Searching answers: Festschrift in honour of michael hess on the occasion of his 60th birthday* (pp. 45-48). Münster, Germany: Monsenstein und Vannerdat. Retrieved from http://dx.doi.org/10.5167/uzh-23868

Fuchs, N. E., Schwertel, U., & Schwitter, R. (1990). Attempto controlled english - not just another logic specification language. In *LOPSTR '98: Proceedings of the 8th international workshop on logic programming synthesis and transformation* (pp. 1-20). Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=646917.7090099

Fuchs, N. E., Schwertel, U., & Schwitter, R. (1999). *Attempto controlled english (ACE): Language manual version 3.0.* Universität Zürich. Institut für Informatik.

Gause, D. C., & Weinberg, G. M. (1989). *Exploring requirements: Quality before design.* Dorset House Publishing Co., Inc.

Gruzitis, N., & Barzdins, G. (2010). Polysemy in controlled natural language texts. In N. Fuchs (Ed.), *Lecture Notes in Computer Science: Controlled natural language* (Vol. 5972, pp. 102-120). Springer Berlin Heidelberg. doi:10.1007/978-3-642-14418-9_7

IEEE Guide to Software Requirements Specifications. (1984). *IEEE Std 830-1984.* doi:10.1109/IEEESTD.1984.119205

IEEE Standard Glossary of Software Engineering Terminology. (1990). *IEEE Std 610.12-1990*, 1. doi:10.1109/IEEESTD.1990.101064

Johnston, L. J., & Mahemoff, M. J. (1998). Software internationalisation: Implications for requirements engineering. In L. Dawson & D. Fowler (Eds.), *Proceedings of the third australian workshop on requirements engineering* (pp. 83-90). In Geelong, Australia, October 26-27, 1998: Deakin University. Retrieved from http://mahemoff.com/paper/reqsi18n/

Kaljurand, K. (2007). *Attempto controlled english as a semantic web language. (PhD thesis).* Faculty of Mathematics and Computer Science: University of Tartu.

Kamalrudin, M., Hosking, J., & Grundy, J. (2011). Improving requirements quality using essential use case interaction patterns. In *Software engineering (ICSE), 2011 33rd international conference on* (pp. 531-540). doi:10.1145/1985793.1985866

Kamsties, E. (2005). Understanding ambiguity in requirements engineering. In A. Aurum & C. Wohlin (Eds.), *Engineering and managing software requirements* (pp. 245-266). Springer Berlin Heidelberg. doi:10.1007/3-540-28244-0_11

Kamsties, E., & Peach, B. (2000). Taming ambiguity in natural language requirements. In *Proceedings of the thirteenth international conference on software and systems engineering and applications 2000.* Paris, France.

Kamsties, E., Berry, D. M., & Paech, B. (2001). Detecting ambiguities in requirements documents using inspections. In *Proceedings of the first workshop on inspection in software engineering (WISE'01}* (pp. 68-80). Paris, France.

Kaufmann, E., & Bernstein, A. (2010). Evaluating the usability of natural language query languages and interfaces to semantic web knowledge bases. In *Web semantics: Science, services and agents on the world wide web* (Vol. 8, pp. 377-393). Elsevier.

Kuhn, T. (2010). *Controlled english for knowledge representation. (PhD thesis).* Faculty of Economics, Business Administration and Information Technology of the University of Zurich.

Kuhn, T. (2012). *Survey and classification of controlled natural languages.* Retrieved from http://www.tkuhn.ch/pub/kuhn_draft_cnl.pdf

Kulak, D., & Guiney, E. (2004). *Use cases: Requirements in context.* Boston: Addison-Wesley.

Luisa, M., Mariangela, F., & Pierluigi, N. (2004). Market research for requirements analysis using linguistic tools. *Requirements Engineering*, *9*(1), 40-56. doi:10.1007/s00766-003-0179-8

Mich, L., & Garigliano, R. (2000). Ambiguity measures in requirement engineering. In *Proceedings of ICS 2000 16th IFIP WCC* (pp. 39-48). Beijing, China, 21–25 August 2000.

Mullery, G. (1996). The perfect requirement myth. *Requirements Engineering*, *1*(2), 132-134. doi:10.1007/BF01235906

Palyagar, B., & Richards, D. (2005). A communication protocol for requirements engineering processes. In *International workshop on requirements engineering: Foundations of software quality* (p. 16). Porto, Portugal.

Philippo, E. J., Heijstek, W., Kruiswijk, B., Chaudron, M. R. V., & Berry, D. M. (2013). Requirement ambiguity not as important as expected: Results of an empirical evaluation. In *Proceedings 19th international working conference on requirements engineering: Foundation for software quality (REFSQ 2013).*

PM SOLUTIONS Inc. (2011). Strategies for project recovery. Retrieved from http://www.pmsolutions.com/collateral/research/Strategies for Project Recovery 201.pdf

Popescu, D., Rugaber, S., Medvidovic, N., & Berry, D. (2008). Reducing ambiguities in requirements specifications via automatically created object-oriented models. In B. Paech & C. Martell (Eds.), *Lecture Notes in Computer Science: Innovations for requirement analysis. From stakeholders needs to formal designs* (Vol. 5320, pp. 103-124). Springer Berlin / Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-540-89778-1%5C_10

Pretorius, L., & Schwitter, R. (2009). Towards processable afrikaans. In *Workshop on controlled natural language (CNL 2009)* (pp. 1-5). Marettimo Island, Italy, 8-10 June, 2009.

Sawyer, P., Rayson, P., & Cosh, K. (2005). Shallow knowledge as an aid to deep understanding in early phase requirements engineering. *Software Engineering, IEEE Transactions on*, *31*(11), 969 - 981. doi:10.1109/TSE.2005.129

Schwitter, R. (2002). English as a formal specification language. *Database and Expert Systems Applications, 2002. Proceedings. 13th International Workshop*, 228-232. doi:10.1109/DEXA.2002.1045903

Schwitter, R. (2004). Representing knowledge in controlled natural language: A case study. In M. G. Negoita, R. J. Howlett, & L. C. Jain (Eds.), *Lecture Notes in Computer Science: Knowledge-Based intelligent information and engineering systems* (Vol. 3213, pp. 711-717). Springer Berlin Heidelberg. doi:10.1007/978-3-540-30132-5_97

Schwitter, R. (2010). Controlled natural languages for knowledge representation. In *COLING '10: Proceedings of the 23rd international conference on computational linguistics: Posters* (pp. 1113-1121). Stroudsburg, PA, USA: Association for Computational Linguistics.

Schwitter, R., & Fuchs, N. E. (1996). Attempto controlled english - A seemingly informal bridgehead in formal territory. In *Proceedings of JICSLP96* (pp. 169-180). Bonn, Duitsland, September 1996.

Schwitter, R., & Tilbrook, M. (2006). Annotating websites with machine-processable information in controlled natural language. In *AOW '06: Proceedings of the second australasian workshop on advances in ontologies* (Vol. 72, pp. 75-84). Hobart, Australia: Australian Computer Society, Inc. Retrieved from http://dl.acm.org/citation.cfm?id=1273659.1273669

Sowa, J. F. (2004). Common logic controlled english. *Technical report* [Web page]. Retrieved from http://www.jfsowa.com/clce/specs.htm

Sowa, J. F. (2007). Common logic controlled english. *Technical report* [Web page]. Retrieved from http://www.jfsowa.com/clce/clce07.htm

Standish Group. (2009). CHAOS report 2009. In Boston: The Standish Group International, Inc. Retrieved from www.standishgroup.com

de Swart, N. (2010). *Handboek requirements, brug tussen business en ICT.* Delft: Eburon Uitgeverij BV.

Visscher, W., Kats, J., Terwijn, B., Arnoldus, J., & van den Berg, D. (2005). Metaforen in requirements engineering: Metaforen hulpmiddel bij communicatie over it. *Informatie*, *2005*, 12-15.

Wiegers, K. E. (2009). *Software requirements.* Redmond, WA: Microsoft press.

Wyner, A., Angelov, K., Barzdins, G., Damljanovic, D., Davis, B., Fuchs, N., . . . Kuhn, T. (2010). On controlled natural languages: Properties and prospects. In N. Fuchs (Ed.), *Lecture Notes in Computer Science: Controlled natural language* (Vol. 5972, pp. 281-289). Springer Berlin Heidelberg. doi:10.1007/978-3-642-14418-9_17

Yang, H., Willis, A., De Roeck, A., & Nuseibeh, B. (2010). Automatic detection of nocuous coordination ambiguities in natural language requirements. In *ASE '10: Proceedings of the IEEE/ACM international conference on automated software engineering* (pp. 53-62). Antwerpen, België: ACM. doi:10.1145/1858996.1859007

Bijlagen

A Requirementsdocumenten onderzoek

A.1 Requirements validatie

Project 1

1. The main users that have access to the EHR are the technician and the optometrist.
2. The front desk users will use the ECP platform mainly, but will get a lite role into the EHR.
3. Other roles, like BPO, Dispensary, etc, will not have access to the EHR currently.
4. The front desk users will not have access to the patient medical data.
5. After an patient has been selected from patient slider as described in 4.2 “Appointments and scheduler” you will see the summary screen of the selected patient.
6. The content is organized in widgets that users can add/remove, resize and move around.
7. This portlet shows the allergy name with the severity.
8. Both medication and non-medication allergies are shown here.
9. The EHR presents one form for all data entered during an exam.
10. All medical data can be viewed in the context of an exam and can also be grouped by content type in the tabs.

Project 2

1. Indien status afgekeurd, in een tooltip de afkeur tekst tonen (tooltip verschijnt wanneer de muis over de tekst wordt bewogen).
2. Een knop in de menubalk van waaruit de inhoud van deze view kan worden geëxporteerd naar Excel, met inbouw van filtermogelijkheid, waardoor gefilterde exports kunnen worden gemaakt (filters nog bepalen), export wordt dan door de software aangemaakt en aangeboden, welke daarna lokaal is op te slaan dan wel te printen.
3. de functionaliteit van de zogenaamde toggle knop aanpassen, zodat naast overslaan hiermee ook de status van een declaratieregel kan worden omgezet van ‘incomplete COV’ naar ‘te verwerken’(exacte selectiemogelijkheden nog bepalen. Deze functionaliteit is nodig in geval men bewust met een ongeldige en/of niet-actuele COV-status een bepaalde patiënt wil declareren
4. Een knop in de menubalk ‘COV verversen’ waarmee, na gedane aanpassingen op praktijk niveau, de COV status in deze view kan worden ververst om het resultaat van de correcties te kunnen zien, alvorens de declaratie te verzenden.
5. De declaratie tool biedt per declaratie een logfile. Door op de knop “Toon log” te klikken, verschijnt een popup met declaratielog (zie onderstaande schermafdruk).
6. In deze log komen onder meer de onverzekerde patiënten naar voren, alsmede dubbele patiënten (zie onderstaande voorbeelden). Op dit moment wordt de fout met patiëntnaam en HAP in 1 regel weergegeven.
7. Voor elk attribuut (naam patiënt, BSN, geboortedatum, AGB HAP, naam HAP, foutmelding) een aparte kolom maken.
8. Een export knop met filteropties toevoegen, zodat een CSV bestand kan worden aangemaakt voor verdere opvolging/communicatie met de praktijken. Deze export wordt dan door de software aangemaakt en aangeboden, welke daarna lokaal is op te slaan dan wel te printen.
9. In de log ook patiënten opnemen waarbij BSN = 0 of leeg is en Patiënten die zijn geïncludeerd in declaratieperiode, maar geen intake datum hebben. (daarmee vervalt de noodzaak actief te zoeken op actieve patiënten zonder intake datum)
10. In de log ook patiënten opnemen waarbij (BSN = 0 of leeg is en Patiënten die zijn geïncludeerd in declaratieperiode), maar geen intake datum hebben.
11. In aanmaken declaratieregels rekening houden met inactieve praktijken die ten tijde van te declareren kwartaal (in verleden) nog wel actief waren. Hiertoe moet de datum inactief worden vastgelegd (inactive from date), vervolgens bij declaratie inactieve praktijken ‘inactive from date’ vergelijken met te declareren kwartaal□

Project 3

1. Members are external users of type Referring Physician or Support Staff.
2. Members are assigned to one practice, but can also work in other practices with the same network.
3. Both the Referring Physician and Support Staff can have the additional role of Practice Administrator.
4. Every new user will be added inth e "Registered" state, and they will be assigned the right roles. However, only after the user has accepted the Privacy & Confidentiality Agreement, they will be able to access the application.
5. Only after the user has been validated by the enrollment Coordinator, they will receive the full functionality, which includes viewing of results.
6. Please note that the Self-sign on form is filled in by an anonymous user. This is viewed as a separate portal, in which the user will only be able to access the self-sign on form.
7. Each site has a Referring Physician Office (RPO) that has a Director and Physician Liaisons. Physician Liaisons where previously known as field representative.
8. Each site has a Referring Physician Office (RPO) that has a Director and Physician Liaisons.
9. Each site also has an Appointment Office that processes the requests. Employees within this office are Appointment Coordinators. MCR has an International Appointment Coordinator that manages the international e- Consults. MCF and MCA have Transplant and International Appointment Coordinators who manage those requests separate from the general requests.
10. Each site also has an (Appointment Office) that processes the requests of Each site also has an (Appointment Office that processes the requests)
11. Furthermore, there is a Site Administrator (=Enrollment Coordinator) that enrolls new users.
12. The variation exists in whether the user is focused at one site, versus any site. Also, the Switch user function is not available to all users.

A.2 Resultaten checklist onderzoek

In de onderstaande tabellen staan de resultaten van het checklist onderzoek. In de tabellen worden de volgende afkortingen gebruikt:

PO: Polysemie
SP: Systematische polysemie
RA: Referentiële ambiguïteit
DA: Discourse ambiguïteit
DO: Domein ambiguïteit
1: De eerste persoon die de requirements heeft geanalyseerd met de checklist
2: Persoon die de requirements heeft geanalyseerd met de checklist.
3: Persoon die de requirements heeft geanalyseerd met de checklist.
4: De laatste persoon, deze heeft de requirements geanalyseerd met de vernieuwde checklist.

Project 1

Requirement	**1**	**2**	**3**	**4**
1. The main users that have access to the EHR are the technician and the optometrist. *users is meervoud, technician en optometrist is enkelvoud -> 2 users in het hele systeem*		SP		
2. The front desk users will use the ECP platform mainly, but will get a lite role into the EHR. *ECP Platform* *Gebruiken de frontdesk users het platform hoofdzakelijk of wordt het platform hoofdzakelijk door de frontdesk users gebruikt.*		PO RA		
3. Other roles, like BPO, Dispensary, etc, will not have access to the EHR currently. *Other roles* *Het woord role is ambigu, eerst met betrekking tot systeem daarna tot de reële wereld.*	DA PO	DA	DA	DA
4. The front desk users will not have access to the patient medical data. *Patiënt persoonlijk data of data van al de patiënten*	SP	SP	SP	
5. After an patient has been selected from patient slider as described in 4.2 "Appointments and scheduler" you will see the summary screen of the selected patient. *Wordt het selectie proces beschreven of de patient slider described in 4.2 "Appointments and scheduler"*		RA DA	DA	
6. The content is organized in widgets that users can add/remove, resize and move around. *Can users add remove widgets of Can widgets add remove users content*	RA	RA		DA

7. This portlet shows the allergy name with the severity.				
Wordt de severity door de portlet weergegeven.	RA	RA		
Welke portlet	DA		DA	DA
8. Both medication and non-medication allergies are shown here.				
Both (medication) and (non-medication allergies) are shown here of Both (medication and non-medication allergies) are shown here.	RA		RA	RA
Is het medication allergies	RA			RA
here	RA			RA
9. The EHR presents one form for all data entered during an exam.				
The EHR presents one form (for all data entered during an exam) of The EHR presents (one form for all data entered) (during an exam).	RA	RA		
Wat is een form, forumlier of een vorm			PO	
one form				SP
entered				PO
10. All medical data can be viewed in the context of an exam and can also be grouped by content type in the tabs.				
All medical data, de medische data van de gebruiker, van het systeem etc.	PO		PO	
an exam and can also be grouped by content type in the tabs. in de medical data				RA
Context type				PO
tabs				PO

Project 2

Requirement	**1**	**2**	**3**	**4**
11. Indien status afgekeurd, in een tooltip de afkeur tekst tonen (tooltip verschijnt wanneer de muis over de tekst wordt bewogen).				
Status afgekeurd (is de status afgekeurd of is de huidige status "afgekeurd")	SP			
Afkeur tekst, die tekst die afgekeurd is of de reden waarom de tekst is afgekeurd.	PO	PO		PO
Tooltip ambigu			PO	
12. Een knop in de menubalk van waaruit de inhoud van deze view kan worden geëxporteerd naar Excel, met inbouw van filtermogelijkheid, waardoor gefilterde exports kunnen worden gemaakt (filters nog bepalen), export wordt dan door de software aangemaakt en aangeboden, welke daarna lokaal is op te slaan dan wel te printen.				
Een knop, menubalk of Excel met inbouw van filtermogelijkheid.	RA		RA	RA
Het woord welke, kan verwijzen naar de menubalk, knop of export	RA			
Is de software op te slaan en uit te printen?		RA		
lokaal				SP
13. De functionaliteit van de zogenaamde toggle knop aanpassen, zodat naast overslaan hiermee ook de status van een declaratieregel kan worden omgezet van 'incomplete COV' naar 'te verwerken'(exacte selectiemogelijkheden nog bepalen. Deze functionaliteit is nodig in geval men bewust met een ongeldige en/of niet-actuele COV-status een bepaalde patiënt wil declareren				
Woord overslaan ambigu	PO	PO		
De verwijzing van de functionaliteit, in de zin deze functionaliteit	RA	RA		
De functionaliteit			SP	
de toggle knop				DA
14. Een knop in de menubalk 'COV verversen' waarmee, na gedane aanpassingen op praktijk niveau, de COV status in deze view kan worden ververst om het resultaat van de correcties te kunnen zien, alvorens de declaratie te verzenden.				
Met de menu balk of de knop	RA	RA	RA	
COV verversen			DO	
een knop in de menubalk 'COV verversen' waarmee, na gedane aanpassingen op praktijk niveau, de COV status in deze view kan worden ververst om het resultaat van de correcties te kunnen zien, (alvorens de declaratie te verzenden).				RA
15. De declaratie tool biedt per declaratie een logfile. Door op de knop "Toon log" te klikken, verschijnt een pop-up met declaratielog (zie onderstaande schermafdruk).				
Het woord declaratie is ambigu		PO		
popup met declaratie log of een popup en de declaratie log		RA		
klikken				DO
declaratie log en logfile				RA
16. In deze log komen ondermeer de onverzekerde patiënten naar voren, alsmede dubbele patiënten (zie onderstaande voorbeelden). Op dit moment wordt de fout met patiëntnaam en HAP in 1 regel weergegeven.				
(fout met patiëntnaam) en (HAP) of (fout met patiëntnaam en HAP)	RA	RA		

Naar voren is spreektaal	PO			
HAP			DO	
deze log			DO	
17. Voor elk attribuut (naam patiënt, BSN, geboortedatum, AGB HAP, naam HAP, foutmelding) een aparte kolom maken.				
Woord kolom is ambigu		PO	PO	PO
AGB HAP BSN domein specifieke woorden		DO		
18. Een export knop met filteropties toevoegen, zodat een CSV bestand kan worden aangemaakt voor verdere opvolging/communicatie met de praktijken. Deze export wordt dan door de software aangemaakt en aangeboden, welke daarna lokaal is op te slaan dan wel te printen.				
Door de software, welke software wordt bedoeld, model, platform of extern.	DO			DO
Kan de software worden opgeslagen en uitgeprint	RA	RA		
Een export knop met filteropties			RA	
filteropties				PO
lokaal				PO
19. In de log ook patiënten opnemen waarbij BSN = 0 of leeg is en Patiënten die zijn geïncludeerd in declaratieperiode, maar geen intake datum hebben. (daarmee vervalt de noodzaak actief te zoeken op actieve patiënten zonder intake datum)				
in de log ook patiënten opnemen waarbij (BSN = 0 of leeg is en Patiënten die zijn geïncludeerd in declaratieperiode), maar geen intake datum hebben.	RA	RA		
BSN = 0 of leeg			SP	
Wat zijn actieve patienten		PO		
20. In aanmaken declaratieregels rekening houden met inactieve praktijken die ten tijde van te declareren kwartaal (in verleden) nog wel actief waren. Hiertoe moet de datum inactief worden vastgelegd (inactive from date), vervolgens bij declaratie inactieve praktijken 'inactive from date' vergelijken met te declareren kwartaal□				
Het woord inactief is ambigu		PO		
Hiertoe moet de datum inactief worden vastgelegd (inactive from date)		RA		
rekening				DO

Project 3

Requirement	**1**	**2**	**3**	**4**
21. Members are external users of type Referring Physician or Support Staff.				
Members are (external users of type Referring Physician or Support Staff).	RA	RA	RA	
Members				DA
22. Members are assigned to one practice, but can also work in other practices with the same network.				
Woord network is ambigu	PO	PO	PO	
but can also work ()				DO
work				PO
23. Both the Referring Physician and Support Staff can have the additional role of Practice Administrator.				
additional role of Practice Administrator.		RA		
additional role				PO
24. Every new user will be added inth e "Registered" state, and they will be assigned the right roles. However, only after the user has accepted the Privacy & Confidentiality Agreement, they will be able to access the application.				
Enkelvoud en meervoud door elkaar		SP		
Wie zijn they	RA		RA	
Wat zijn right roles	PO			
however				RA
25. Only after the user has been validated by the enrollment Coordinator, they will receive the full functionality, which includes viewing of results.				
Enkelvoud en meervoud door elkaar		SP		
full functionality, which includes			DO	
of results	SP			
validated, receive, results				PO
26. Please note that the Self-sign on form is filled in by an anonymous user. This is viewed as a separate portal, in which the user will only be able to access the self-sign on form.				
De self-sign on is een form met een portal waarweer dezelfde form in zit.		RA		
De self-sign on form			DO	
self sign on form alleen te zien				SP
27. Each site has a Referring Physician Office (RPO) that has a Director and Physician Liaisons. Physician Liaisons where previously known as field representative.				

Wat is een site	PO			PO
Each site has a Referring Physician Office (RPO) that has a Director and Physician Liaisons.		RA		
RPO			DO	
previously				DA
28. Each site also has an Appointment Office that processes the requests. Employees within this office are Appointment Coordinators. MCR has an International Appointment Coordinator that manages the international e- Consults. MCF and MCA have Transplant and International Appointment Coordinators who manage those requests separate from the general requests.				
Each site also has an (Appointment Office) that processes the requests of Each site also has an (Appointment Office that processes the requests)	RA	RA		
Wat is een site	DO			DO
MCA have Transplant and International Appointment Coordinators who manage those requests separate from the general requests.				DO
29. Furthermore, there is a Site Administrator (=Enrollment Coordinator) that enrolls new users.				
Domein ambiguiteit		DO		
wat is enroll				DO
30. The variation exists in whether the user is focused at one site, versus any site. Also, the Switch user function is not available to all users.				
Switch ambigu woord	PO	PO		PO
the variation				PO
focussed				PO

B Interviews

B.1 Opzet interview

Situatie
Wil je kort vertellen in welk project je werkzaam bent.

Taak
Wat zijn je taken in dit project?
Hoe heb je te maken met requirements / requirements engineering in dat project.
Hoe heb je te maken met requirements.

Activiteit
Wil je kort vertellen hoe je het aangepakt hebt, het opstellen van de requirements.

Declaratief vs Procedureel
Wat heeft bij jou de voorkeur bij het opstellen van de requirements, de requirement declaratief of procedureel beschrijven? Dit is beschrijf je bij requirements wat het systeem moet doen of hoe het systeem moet werken?

Waarom kies je hiervoor?
Wat vind jij de voordelen van deze werkwijze?
Denk je dat er ook nadelen van deze werkwijze zijn?

Bij declaratief - Ik zag in de documenten die je gemaakt hebt dat je vooral de requirements procedureel beschrijft, waarom heb je er in die situatie voor gekozen om dit op die manier te doen?

Verklarende woordenlijst
Krijg je wel eens vragen van klanten dat ze woorden in de requirements niet begrijpen?
Krijg je wel eens vragen van ontwikkelaars dat ze woorden in de requirements niet begrijpen?
Is het de gewoonte om bij requirements beschrijvingen een verklarende woordenlijst op te nemen?

Ja -> Waarom wordt er in de meeste projecten dan voor gekozen om dit niet te doen?
Nee -> Hoe zorg je er dan voor dat de domeinkennis van bepaalde woorden wordt overgebracht op de ontwikkelaars?

Tekstueel
Wat denk je dat er makkelijker is voor een ontwikkelaar om te begrijpen bij het ontwikkelen van software:

1. Per onderwerp een stuk tekst met daarin de requirements.
2. Per onderwerp een lijst met requirements, voor elke zin een requirement.

Bij 1 -> Dit is inderdaad de vorm die vaak gekozen worden voor de requirements.

Waarom kies je hier voor?
Verwacht je dat ontwikkelaars deze vorm verkeerd interpreteren?\

Bij 2 -> Dit is niet de vorm die meestal wordt gebruikt

Waarom denk je dat deze vorm beter is?
Wat zijn de voordelen van het gebruik van deze vorm?

Resultaat
Is het je wel eens opgevallen dat je veel requirements op meerdere manieren kan interpreteren?

Nee -> Zelf voorbeeld geven en vragen of ze denken dat er een kans bestaat dat de ontwikkelaar dit niet goed interpreteert.
Ja -> Kan je een voorbeeld noemen?

Heb je zelf al een techniek waarmee je dit probeert te voorkomen?
Denk je dat dit invloed kan hebben op de ontwikkeling van de software applicaties?

Krijg je wel eens vragen van ontwikkelaars over requirements waarbij ze niet weten hoe ze deze moeten interpreteren?
Zie je zelf wel eens dat ontwikkelaars requirements op een verkeerde manier interpreteren?

Hoe controleer je dat alle requirements zijn opgenomen in de ontwikkelde software?

Reflectie
Iets uitleg geven over het onderzoek naar formaten, zodat ze het een beetje in de context kunnen beoordelen.

Requirements kwaliteit
Ranking van eigenschappen die de requirements kunnen verbeteren, door middel van kaartjes kan de respondent een keuze maken. Keuze uit de volgende eigenschappen:

- Ontwerp onafhankelijk (Een software requirement is vrij van ontwerp en implementatie beslissingen.)
- Eenduidig (requirements zijn voor maar één uitleg vatbaar.) (Niet ambigu)
- Consistent (specificatie zijn consistent als er geen requirements met elkaar conflicteren)
- Traceerbaar (ontwikkelde requirements zijn makkelijk te traceren vanuit de software naar de requirementsdocumenten).

Welke eigenschappen zijn er aanwezig in de requirements?
Welke eigenschappen worden er gemist in de huidige requirements?
Welke eigenschappen vind je belangrijk voor requirements, leg op volgorde?

Hoeveel tijd zou jij willen investeren om een requirements formaat, die de door jou gekozen eigenschappen in zich heeft, te leren lezen en schrijven?

Ondersteuning van de CNL taal voor requirements
De eigenschappen in dit deel van het interview komen uit de lijst van Wyner.

Wat gebruik je bij het maken van requirements, en leg op volgorde van het meest naar het minst gebruikt in de requirements:

- Beeldspraak (alle zeilen bijzetten / het systeem kijkt in de database)
- Gezegden (Aan het roer zitten)
- Spreekwoorden (Als het kalf verdronken is, dempt men de put.)
- Afwisseling in de structuur van de zinnen.
- Tussen zinnen. (De gebruiker, dit is een persoon die rol x heeft, moet inloggen op het systeem)

Kan je van elke gebruikte style iets vertellen en een voorbeeld geven?
Leg nu de kaartjes op volgorde van wat je belangrijk vindt om te gebruiken?
Verwijs je wel eens vanuit een requirement naar een voorgaande requirement?
Is dit essentieel voor jou bij het maken van requirements?

CNL Taal eigenschappen
De eigenschappen in dit deel van het interview komen uit de lijst van Wyner.

Welke eigenschappen vindt je belangrijk voor een CNL taal:

- Ondersteuning door een tool
- Meertalig zowel Engels als Nederlands
- Eenvoudig te leren, lezen en te schrijven
- Beschikbaarheid van een eenvoudige handleiding

Formele talen
Heb je wel eens wat gedaan gestructureerde talen?
Weet je wat van formele talen?
Gebruik je wel eens formele talen?

Ik zal eerst even een beetje introductie geven van het onderzoek. Ik ben bezig om requirements te analyseren en te kijken wat de kwaliteit van de requirements is en hoe de kwaliteit tussen de bepaalde groepen verloopt; dus als een consultant requirements opstelt en die stuurt zijn requirements naar ontwikkelaars om te kijken of die dan goed aankomen en of de mensen de requirements snappen en het doel is omdat te gaan verbeteren.

Die je van mij hebt gehad was meer wat van de klant naar mij wordt gestuurd.

Ja ik heb bij de documenten van jou het ontwerp geanalyseerd.

Ok, je had het ontwerp erbij.

Ja ik heb één van de documenten die ik van jou gehad heb geanalyseerd, ik heb hierin zin voor zin bekeken wat de kwaliteit van de requirements is. Ik wil dit interview een beetje via de STARR methode doen, ik weet niet of je die bekend voor komt?

Nee

Ik begin eerst met de situatie, in welke project je werkt, dit doe ik om dit interview een beetje in een bepaalde context te plaatsen, daarna de taken die jij hierin hebt, even kort. Om een beetje te weten te komen in welke context dit afspeelt. Dat is dus de T van taak, eerst de S van situatie dan heb je de A van activiteit dus dat houdt het requirements opstellen in, en dan de resultaten...

En taak en activiteiten, wat is het verschil daar tussen...

Je taken zijn wat je moet doen in het project en de activiteiten zijn de dingen die uitvoert in het project. En daarna de reflectie en wat verbeter punten op basis van de theorie die ik al gevonden heb. Wil je kort vertellen aan welk project je nu werkt?

Het <project 1> project is een jaar geleden gestart met een proof of concept voor..... *(deel van het interview weggehaald wegens bedrijfsvertrouwelijke informatie)* **project**.*EHR*

Stukje voorraad beheer?

Ja, voorraad beheer voor je contact lezen, brillen, dat is allemaal voorraad beheer. Dus dan moet je weten van oké welke heb ik nu op voorraad liggen, dus dat is eigenlijk de hele applicatie en het enige deel wat wij doen is, dus wij zijn onderdeel van een grote applicatie, voor ons project wij zijn een EHR aan het bouwen.

Wat is een EHR?

Dat is het medisch dossier, een term voor een EPD, dat is eigelijk de Engelse term. En wij integreren met die andere systemen.

En wat is jou taak in dit project?

Ja is een goeie.. Mijn hoofdtaak is het designen en dan gaat het niet over grafische design maar gewoon echt over functioneel design. En dan richt ik mij vooral op de buitenkant dus wat de user interactie gaat zijn. Echt de usabillity, user interactie. Dat je goed kijkt krijgt de gebruiker of een oogarts terwijl die gewoon in zijn dagelijks werk bezig is terwijl dat hij zijn bezoeken doet van een patiënt heeft hij dan ondersteuning van de software. Of is het puur ballast en administratieve rompslomp. Maar gaat het systeem hem dan toegevoegde waarde leveren. **rol**.*RE*

Of het systeem hem verbetert in zijn dagelijks werk en zijn werk processen...

Ja, dus efficiënt documenteren en helpen waar nodig, dus wat eigelijk een soort ontwikkel doelen zijn, want er zijn natuurlijke heel veel EHR, en ze gebruiken ook allemaal een EHR uit de DOS tijdperken of uit een begin windows tijdperk, groen, geel en oranje schermen, Maar wat wij juist willen doen is dat we de kwaliteit van de zorg proberen te verbeteren. Dat is het design deel wat ik doe daarbij stuur ik dan ook de developers aan in het team, in Nederland, om te bouwen dat zijn <applicatieontwikkelaar>[11] en <applicatieontwikkelaar> vooral en <applicatieontwikkelaar> die zijn vooral bezig met de uitvoer maar die begeleidt ik dan, dus dan kijk ik ook van dit is het design deels opgeschreven maar er zit ook nog een deel tussen je oren en dat zie je soms pas wanneer je hé hier mist nog iets, of hé dit had ik ook bedacht. **rol.***RE*

Je maakt dus een design op het bestaande product?

Ja, je maakt eerst een design op het bestaande product dus dat wordt dan je kader.

Op het hudson product? En daarboven bouw je dan een nieuw product, en daarvoor maak je ook weer opnieuw een design.

Ja, en het gaat heel veel over content dus, echt over wat is nu de dossier content voor een oogarts, dus wat doet een oogarts wat meet hij allemaal en wat wil hij gaan documenteren en aan welke voorwaarde moet het weer voldoen en wanneer hij het gedocumenteerd heeft hoe wil hij dan zien dat hij het gedocumenteerd heeft. Dus het is een heel stuk presentatie van hoe zien invoer schermen eruit, eigenlijk data entry en hoe presenteer je het daarna wanneer je het ingevoerd hebt. En dat hebben we heel erg gescheiden nu dus het is niet zo van jij hebt één scherm net zoals een papieren formuliertje, kijk omdat opnieuw te bouwen van je hebt gewoon een invul formuliertje en je maakt een scherm en je zorgt dat al de veldjes erop staan en dan

type maar in en save maar en later wanneer je het weer nodig hebt ga je weer daar naar toe en dan bekijk je het maar weer maar wij hebben nu echt in het design de insteek van we gaan kijken naar hun proces en we tonen die informatie gewoon wat zij opdat moment nodig hebben. En als ze iets willen invoeren krijgen ze een ander scherm dan als dat ze het tonen. Dus ze hebben eigenlijk het tonen en het invoeren is gewoon gescheiden voor hun.

Jij maakt daar dus de requirements voor, die stel jij op? **verzamelen.**

Ja en die bespreek ik dus ook met die mensen, ik heb ook overleg met vier oogartsen in de VS en ja die vier hebben ook nog wel eens natuurlijke allerlei eigen wensen. *overleg stakeholders*

Bij het opstellen van de requirements wat heeft bij jou de voorkeur, opstellen in declaratieve requirement of procedureel, dus of je opschrijft wat het systeem moet doen of hoe het systeem gaat werken.

Hoe, ja **procedureel**. *voorkeur*

Waarom heeft dit bij jou de voorkeur?

100% omdat als je alleen maar wat opschrijft dan krijg je misschien wel een systeem wat alles kan maar waar gebruikers niet blij van worden. **procedureel.** *positief*

Omdat ze niet weten hoe het systeem gaat werken?

Nou omdat je bij requirements moet je weten van welke requirement wordt door wie op welk moment is door wie op welk moment nodig. Als je dat niet weet dan krijg je per definitie geen goeie user interface design. Dan heb je misschien hele boom met knopje en dan zitten de belangrijke knopjes allemaal onderin, zoiets. **procedureel.** *positief*

Je kiest dus voor het hoe voor de user interface desig dat je dan daarmee beter kan designen?

Ja.... ja

Denk je dat er ook nadelen aanzitten, aan deze werkwijze?

Het is altijd schipperen want ik begin, dus mijn design heb je misschien wel gezien, met use cases die staan bij mij op nummer één. Je hebt use cases en je hebt nog wat andere system requirements. Beetje die.. dat is ook natuurlijk, hoe moet je het zeggen, je interesse of wat jij belangrijk vindt schrijf je op die manier op.

Het nadeel ervan vind ik wel is dat je, kijk als je bepaalde functie hebt, ik heb hier ook best wel lang mee zitten strubbelen met de structuur van mijn design document, mijn initiële design document. Want als je een functie beschrijft die heeft op heel veel plekken heeft die raakvlakken, dus stel je hebt over iets simpels als over appointments, is makkelijk heb je geen domeinkennis voor nodig, dan heb in het begin hebben ze al secretaresse, frontdesk medewerker die gaat zo'n appointment aanmaken. Die heeft daar een interactie met appointments. Dan heb je bijvoorbeeld ook nog een arts die zegt van oké maar er belt iemand secretaresse is er niet die arts wil dat ook nog, die heeft ook nog een interactie met de appointment, voor dat die patiënt er nog is dan komt die patiënt binnen dan wil je ook weer iets met die appointments gaan doen want dan triggert die appointment weer bijvoorbeeld een lijstje wat je continue in beeld hebt van oké deze patiënten moet ik vandaag nog doen. Klik je op zo'n appointment ga je het dossier in dus je ziet dat appointments wordt soms weer in een ander navigatie gebruikt. Als je dat helemaal mooi in appointments kan beschrijven dan ben je wel heel compleet over appointments alleen. **procedureel.** *negatief*

Maar je weet nog niet wat ze er mee gaan doen bedoel je?

Precies, en je weet niet die losse use cases welke processen die weer aan het gebruiken zijn. En als je de grote dingen beschrijft, dan beschrijf je weer allemaal kleine stukjes van het systeem. Dan beschrijf je hier een klein stukje van appointments en daar vandaan. Nou en als je, je kan ook zeggen ik gebruik een combinatie, dat wordt een drama want dan is je design document continue out of sync. Want dan ben je het daar een beetje aan het doen, en daar heb je het net anders beschreven.

En voor de communicatie met de klant, wat vind je dan makkelijker? Het hoe beschrijven of wat? Dus dat je het declaratief op schrijft of procedureel? Heb je dan daar een voorkeur voor?

De hoofd dingen zijn bij mij altijd procedureel, en daarbinnen zeg je van... Dus bijvoorbeeld ik schrijf op ik ga patiënt gegevens aanpassen, ik wil patiënt gegeven kunnen aanpassen. En daaronder als subset van deze en deze en deze gegeven mag je aanpassen en die andere niet zoiets. Of ik ga de status van de afspraken aanpassen dat moet op een bepaald moment nou dan moet daaronder zeg je ook deze en deze en deze statussen kun je aanpassen of als ze bestaan als een afspraak al verlopen is bijvoorbeeld kun je nooit meer opnieuw openen dan moet je altijd weer een nieuwe aanmaken. **procedureel.** *voorkeur*

Heb jij een onderscheid tussen een functioneel ontwerp en een requirementsdocument?

Nee. **functioneel.***nee*

Dus jou intentie is het om het gelijk in één document te stoppen.

Ja, wat ik eigenlijk zie is use cases, Ik zie use cases als een manier om requirements op te schrijven, en dan puur, hoe moet je het zeggen, de titel van een use case, of het onderwerp van een use case. Dat is eigenlijk mijn manier om requirements op te schrijven vanuit gebruiker perspectief. **notatie.***ja*

Hoe bedoel je dat, de titel? De titel geeft je requirement aan?

Kijk mijn use case kan je natuurlijke helemaal specificeren, een use case kan zijn nieuwe patiënt aanmaken, dat is een use case en dan kun je daar specificatie van nieuwe patiënt aanmaken dat zeg meer je hoe, maar nieuwe patiënt aanmaken op zich zegt voor mij wat de requirement is. Dus dat is mijn requirement, de uitwerking van mijn use case dat is mijn functioneelontwerp. **functioneel***.nee*
Wat ik dus eigenlijk zie in mijn documenten ik heb dus mijn requirements deel zit dus in de eerste, in het eerste hoofdstuk, use cases en systeem requirements. En het leuke daarvan is vaak kan je requirements ook als je ze aan het beschrijven bent dan kun je ook eigenlijk in gedachten zeggen van oké de user moet dit kunnen en bijvoorbeeld dan moet het systeem dit doen. Dus je hebt eigenlijk continue, als je al de requirements op een rij zal zetten dan kan je heel veel requirements hebben die beginnen met de user moet dit kunnen en het systeem moet dit doen opdat moment. Veel andere requirements heb je niet.

Jij wilt dus gelijk ook kunnen beschrijven wat het systeem moet doen?

Ja dus een requirement kan bijvoorbeeld zijn, systeem moet checks doen op valide data, dus dat is een hele globale requirement. **functioneel.***nee*
Als ik dan in een bepaald onderdeel zit dan ben ik bijvoorbeeld allemaal metingen aan het doen, gewoon de basis metingen, temperatuur, bloeddruk, je hartslag al die dingen dan zijn mijn subrequirements zijn mijn temperatuur die moet gecheckt worden op deze range want wanneer een temperatuur onder de nul kan wel maar is niet logische dat je die ooit gaat invoeren. Een patiënt met een temperatuur onder de nul, dan heb je een bevroren patiënt, het lijkt me niet dat die nog reageert. Of een ieder geval zo. Dus dat is dan een requirement van mij een systeem requirement van hij moet dus valide invoer checken maar dan specificeer ik de requirement wel per veld dan, als ik dan in het patiënt scherm zit dan zeg ik wel van bloeddruk heeft deze minimum en maximum range, en die heeft ook nog een warning range, dus die zegt als je boven dit zit, kan wel lijkt me niet aannemelijk krijg je een waarschuwing van je mag door maar let op. Dat specificeer je dan wel per veld. Je doet dus wel al een stukje maar je zegt nog niet precies hoe dat je het doet. Hoe dat je het doet, dat heb ik vaak in mijn in een volgend hoofdstuk, een requirements mapping. Dus dan heb ik een mapping hoofdstuk, of paragraaf. Wat ik dus eigenlijk wil, ik zag dit ook in andere templates, misschien moet ik het even laten zien. Kijk dan maak ik echt een mapping van al mijn use cases in al mijn andere requirements naar wat gaat het systeem nou hierop reageren.

Van je requirement direct naar je functionele deel van je beschrijving. Wat vinden de ontwikkelaars, wat denk je dat die makkelijker vinden?

Ontwikkelaars die willen gewoon een mondelinge uitleg, die willen helemaal niet zo'n document. **domeinkennis.** *hoe*

Die willen geen requirements?

Nee die willen geen requirementsdocument. Die willen ze als naslagwerk misschien erbij hebben, maar ze willen gewoon mondelinge uitleg. Het ligt misschien ook aan de ontwikkelaar. **domeinkennis.** *hoe*

Dus je gebruikt die documenten eigenlijk voor je contact met de opdrachtgever, dus het contact tussen VitalHealth en de opdrachtgever?

Ja, ook nog niet eens echt. Het is ook, je moet ze hebben ook voor de toekomst, dus daarom maak ik ze maar ze lopen heel snel, die dingen lopen zo snel achter wij kunnen namelijk heel snel modelleren. **doel.***geen*
Dan zegt iemand wil je even dat labeltje of iets anders aanpassen. Dan passen we het even in het model aan. Dat is minder werk dan in het design document aanpassen. Dus je mockups gaan heel snel uit de maat lopen.

De modelleur maakt eigenlijk geen gebruik van het design document?

Ze krijgen ze wel maar mijn ervaring is dat ze het slecht lezen. Want dan heb je het uitgelegd en dan hebben ze wel het document er naast liggen en dan kijken ze een keer naar het mockupje. **domeinkennis.** *negatief*

Gebruiken ze het wel om de applicatie te checken?

Ja.

Krijg je wel eens van klanten vragen dat ze bepaalde woorden in de requirementsdocumenten niet begrijpen?

Nee niet echt maar ik denk dat dat komt doordat ze slecht lezen en alles wat ze niet snappen dan denken ze dat het aan hun ligt wanneer ze het niet snappen.

lex.
stakeholder.
niet
reden.*slecht lezen*
reden.*kennisgebrek*

En van de modelleurs (ontwikkelaars)? Krijg je daar wel eens vragen van, van ik begrijp dit woord niet, wat wordt hiermee bedoeld, in welke context moet ik dit lezen?

Nee niet echt.

lex.
ontwikkelaar.
niet

Is het het de gewoonte om bij een requirementsdocument een verklarende woordenlijst mee te leveren? Waarin je domein specifieke woorden in uitlegt.

Het is wel een beetje de opzet omdat te doen. Wat ik nu gedaan heb is we hebben een high level design document dat beschrijft echt op heel hoog niveau, redelijk hoog niveau de applicatie en dat is ook een stukje van het contract. Wij gaan een applicatie voor jullie bouwen en die gaat dit en dit en dit doen en gaat er ongeveer zo uitzien en nadere details gaan we later invullen. En daar zit wel een hele proces beschrijving in.

woordenlijst.
wel

Maar met de woordenlijst probeer je een beetje de domeinkennis die jij hebt over te brengen naar de ontwikkelaars toe.

ontwikkelaar.
hoe

Ja maar heel veel gaat ook mondeling. Het is vooral voor de ontwikkelaars, en voor mijzelf ook. Het is wel ook een. Kijk hier heb ik een voorbeeld van een verklarende woordenlijst. Dit document is het meest uitgebreide document.

woordenlijst.
wel

Wat denk jij dat het meest makkelijk is voor een ontwikkelaar om te begrijpen bij het ontwikkelen van de software. Dat je de requirements in een stuk tekst definieert of dat je de requirements per requirement definieert in een lijst, en dan genummerd hebt of zo?

Tekst gaan ontwikkelaars niet zitten lezen, die lezen zo slecht dus wat dat betreft is een lijstje waar ze een soort checklist aan het einde hebben van requirements heeft wel een voordeel.

reden.
slechtlezen
tekstueel.*afkeur*
lijst.*voorkeur*

Want wat ik nu vaak zie is dat er gekozen wordt voor tekst, wat is de reden daarvan? Heb je hier een bepaalde keuze ingemaakt?

Ja dat doe ik omdat er vaak al veel uitleg bij zit. Wat ik hier gedaan heb is in tekst. Je zo het ook in een lijstje kunnen opschrijven. Wat ik een keer gedaan heb bij het CWZ project is dat ik ze in een lijstje had opgesteld en allemaal genummerd. Kijk wat ik hier gedaan heb is de requirement opgesteld en daaronder een beschrijving erbij gegeven. En die heb ik in mij requirements mapping genummerd met kopjes.

project.*PP*

Heeft dat voordelen voor een ontwikkelaar, wanneer de requirements zijn genummerd?

Niet echt, want als je een heel groot document hebt, kijk ik heb wel eens meegemaakt van het revalidatie EPD daar zaten vijfhonderdtachtig requirements in, genummerd van E.1 tot E.500, maar ja zonder dan al te veel tekst kom je dan als ontwikkelaar ook niet heel snel veel verder.

lijst.*afkeur*

Komt het voor dat wanneer het in een tekst zit de ontwikkelaar een bepaalde zin verkeerd interpreteert omdat hij dit in het verkeerde verband leest?

Ja dat kan in principe altijd. En hier is dus de combinatie gekozen met een stukje uitleg erbij. Ik merk toch dat iedereen er nog wel uitleg bij nodig heeft. Of dat nou aan de gemakzucht van een ontwikkelaar ligt of het slecht opschrijven van mij of een combinatie van die twee.

bewust.*wel*

Heb je het wel eens meegemaakt dat een ontwikkelaar een requirement verkeerd interpreteert?

Oh ja zat.

interpretatie ontw.*wel*

Heb je hier misschien een voorbeeld van?

Dit is wel een leuk voorbeeld, als je hier aan de linkerkant van dit scherm heb je een lijstje met aandoeningen die moet je hier vandaan kunnen drag en droppen op zo'n oog. Dus je krijgt een lijst en als je moet slepen dan, er gebeuren hier best wel dingen want als een ding dan op een oog komt dan moet het hier zichtbaar zijn, dus als je het geselecteerd hebt dan moet je daar een lijst met details krijgen. Maar oké welke items die hierin moeten komen bijvoorbeeld, dat staat dan heel kort beschreven van dat zijn de favorieten daarmee gekenmerkt. Maar in eerste instantie kreeg ik daar dus ja wel favorieten maar ook over veel te veel levels heen. Ik moest alleen van af een bepaalde tak in een tree moest je daarvan de favorieten zien. En als je die gaat zoeken dan moet je ook alleen binnen die tak gaan zoeken. Want als je de buitenkant van je oog gaat bekijken dan heb je alleen een bepaalt deel van de problemen die je daar kunt hebben, je kan geen problemen op je netvlies hier opschrijven.

ambiguïteit.
voorbeeld

De ontwikkelaar had dus al de items in de lijst gezet?

Ja, hij was er eerst mee bezig om daar alles in te zetten.

Waarom had hij dat gedaan? Had hij het verkeerd gelezen?

Ja, of verkeerd van mij gehoord of een combinatie hiervan. Het gaat er ook om, ik denk wat het belangrijkste is waarom dingen fout gaan is omdat een ontwikkelaar soms alleen de eis of de wens ziet en niet snapt zelf wat ermee moet gebeuren.

Wat zou de oorzaak daarvan zijn?

Kijk **domeinkennis** gaat meer over die en die inhoud, maar je kunt natuurlijke technisch heel goed uitleggen ik heb hier een lijstje met problemen ik sleep die erop en je wilt het probleem gedocumenteerd hebben. Kijk dat kan iedere ontwikkelaar snappen maar als je dan hier bijvoorbeeld een lijst met achthonderd waardes gaat zien, dan moet je als ontwikkelaar gaan snappen van dit gaat niet werken. **reden.***inzicht*

Een ontwikkelaar zou eigenlijk moeten weten dat die lijst heel lang gaat worden?

Of op dat moment zal een ontwikkelaar ook een trigger moeten hebben van de lijst wordt achthonderd lang, is dat de bedoeling. Want daar is vaak al een oplossing voor bedacht, soms ook niet soms mist dat ook. En in dat geval het meedenken van de ontwikkelaar, van dit lijkt me niet logisch of dit lijkt mij niet handig, of dit lijkt een gedrocht van een tool.

Controleer jij de requirements dan ook of ze op meerdere manieren kunnen worden geïnterpreteerd?

Nee, ik steek het liefst zo veel mogelijk energie in een mondelinge uitleg en het tijdens het proces in de gaten houden en achter af weer, maar dat is meer omdat het efficiënter is in ons team. **ambiguïteit.** *voorkomen*

In mijn onderzoek ben ik bezig om te kijken of er bepaald formaten zijn waarmee requirements kunnen worden opgesteld. In een bepaalde structuur of formaat om zo de kwaliteit van de requirements te kunnen verbeteren. Ik heb hier een aantal kaartjes, dan vraag ik aan jou of je deze even wilt bekijken. Of je wilt bekijken of deze al aanwezig zijn in de requirements? Het hoeft niet speciaal jou documenten te zijn maar in het algemeen binnen VitalHealth.

Die is niet aanwezig.

Dat is eenduidig.

Niet altijd eenduidig, **eenduidig**.*niet*

Implementatie onafhankelijk.

Nee ook niet, het is altijd afhankelijk. Bijna alles in het design wat beschreven staat is juist ontwerp afhankelijk. Wat bedoel je met ontwerp afhankelijk, is er een stukje implementatie beschreven zeg maar al. **implementatie.** *niet*

Vrij van implementatie.

Nee nooit. Kijk in mijn ontwerpdocument heb ik die twee hoofdstukken nu dus zitten. In het ene hoofdstuk is echt requirements daarin probeer ik echt te focussen op requirements. En in het andere hoofdstuk de implementatie. Ik zou zeggen de requirements moeten eigenlijk zo ontwerp onafhankelijk mogelijk zijn en in het tweede hoofdstuk niet. Want daarin wil je juist het op die manier op gaan schrijven die het best past, dat is eigenlijk het designen.

Conflicteert niet met een ander requirement, eigenlijk wel. Eigenlijk zijn user interface requirements altijd conflicterend.

Hoe bedoel je? Je hebt altijd, ik zie heel vaak dat als je alle dingen samen neemt, dus een gebruiker die snel in een systeem gewend wil zijn is een wizzard bijvoorbeeld heel handig. Maar een andere gebruiker die al wat langer in het systeem zit die wil zijn scherm misschien helemaal vol met knopjes. Want dan heeft hij maar één scherm. Je hebt bijvoorbeeld een requirement dat een applicatie heel makkelijk aan te leren moet zijn. En hele high level requirement natuurlijk, want wat zegt het uiteindelijk. Je moet dus heel snel een bepaalde taak kunnen doen of heel makkelijk een bepaalde taak kunnen doen. Dat strijd soms wel eens met elkaar. Want bij het makkelijk moet het ook nog snel gaan. **consistent.***niet*

Binnen één document kunnen requirements daardoor wel met elkaar gaan conflicteren.

Ja. want soms wil je het scherm ook niet helemaal vol zetten met knopjes. Maar dan blijven sommige mensen wel allemaal requirements aanleveren want die heeft die feature in 0,075% van de gevallen nodig dus die wil er een knop voor. Gaan je dan die knop maken of niet, dat is dan de afweging van de requirement om die knop daar te hebben.

Maar hoe doe je dat, schrijf je dan beide requirements op in het document?

Ja ik schrijf ze wel beide op.

Moet de ontwikkelaar dan de afweging maken?

Nee dat moet de designer doen, dat beschrijf je dan in het functionele deel.

Verifieerbaar, is de requirement te verifieren in het systeem, kunnen bijvoorbeeld testers de requirement terug vinden in de applicatie om te testen.

Dat is wel een ja denk ik. Dat is denk ik ook het grote voordeel vanuit use cases denken, want een tester denkt ook in use cases. **verifieerbaar.** *wel*

De respondent heeft de volgende kaartjes uitgekozen:

Wel in de requirements: Verifieerbaar

Niet in de requirement: Consistent, Ontwerp onafhankelijk, Eenduidig.

Wanneer je nadenkt welke eigenschappen je wel belangrijk vindt, kan je ze dan op volgorde leggen, van welke je heel belangrijk vindt en welke wat minder.

Ja, belangrijkheid voor wie, want ik zie wel verschillende doelen, belangrijk voor degenen die, voor de toekomst bijvoorbeeld om te weten hoe een bepaalt design tot stand gekomen is kan het handig zijn om te weten wat waren de requirements erachter, voor deze bijvoorbeeld de ontwerp onafhankelijk. Dat is dan heel erg onafhankelijk zijn vind ik belangrijk. Zodra ze naar de ontwikkelaar gaan dan boeit het me eigenlijk niet zo veel. Dus laten we het meer in het algemeen nemen.

Je mag ze ook naast elkaar leggen waneer je ze even belangrijk vindt.

Deze staat zeker aan de top (verifieerbaar), ik denk dat die er ook wel redelijk in zit. Wat is van de andere drie nu belangrijker... Deze vind ik namelijk heel goed als je nog een keer terug gaat (onafhankelijk) en als je met meerdere mensen aan het ontwerpen bent. Dus laten we die toch iets omhoog leggen. En deze denk ik als minste onderaan omdat, dat doel haal je toch nooit, dus doe het gewoon zoals je..... Want dan krijg je weer tegenstrijdige belangen want dan krijg je een heel uitgebreid document maar dan leest weer niemand het. Dus dan kan het wel heel eenduidig zijn maar wordt niet gebruikt.

verifieerbaar. *belangrijk*
implementatie. *belangrijk*
eenduidig.*onbelangrijk*

De respondenten heeft de kaartjes op de volgende volgorde gelegd, van belangrijk tot minder belangrijk:

Traceerbaar -> Ontwerp onafhankelijk -> Consistent -> Eenduidig.

Ik heb hier nog een paar kaartjes, wil je vertellen of je dit gebruikt? Als je beeldspraak hebt, gebruik je dat wel eens.

Ja

beeldspraak. *wel*

Een gezegde?

Nee, ik ben niet zo taalkundig aangelegd.

gezegde.*niet*

Spreekwoord?

Nee

spreekwoord. *niet*

Tussenzin?

Ja, genoeg.

tussenzinnen. *wel*

Afwisseling van structuur van zinnen?

Dat zal automatische wel.

afw. structuur zinnen.*wel*

De respondent heeft de volgende kaartjes uitgekozen:

Wel in de requirements: Beeldspraak, Tussenzinnen en Afwisseling in structuur zinnen.

Niet in de requirement: Gezegde en spreekwoord.

Heb je nog een bepaalde voorkeur voor een ranking, van welke je belangrijk vindt?

Nee

En degenen die je nu gebruikt, die wil je graag blijven gebruiken?

Ja, maakt mij niet zo veel uit. Voor mij hoeft taal niet mooi te zijn, als het maar functioneel is. Als een tussenzin handig is om te doen moet je het doen en anders niet.

Verwijs je wel eens vanuit een requirement naar een ander requirement?

Ja, veel.

verwijzingen. *wel*

Is dit essentieel? Voor het maken van jou requirements?

Ja, ik denk dat juist dat het is, als je dat niet zou doen, als je requirements niet zou verwijzen naar elkaar dan krijg je geen goed systeem.

verwijzingen. *belangrijk*

Ik heb hier nog vier andere kaartjes, en dat gaat meer over eigenschappen die de talen zelf hebben. Het zijn losse eigenschappen. Welke vindt je belangrijk als je een taal zou gebruiken?

Meertalig?

Als Engels wordt ondersteunt dan is het goed, Nederlands vind ik totaal onzinnig voor een requirement. Misschien voor Nederlandse klanten die dat willen.

meertalig. *onbelangrijk*

Een handleiding beschikbaar voor de taal

Maakt me niet zoveel uit, de requirements moeten worden ingevoerd in een tool.

handleiding. *onbelangrijk*

Een tool voor de ondersteuning, die kijkt dan of de taal die je gebruikt consistent is.

Ik weet het niet, het gaat om ondersteuning, die ondersteuning levert je wat op, het levert je tijd op. Maar als de tijd die je erin moet steken zolang die minder is dan is het niet erg. Maar als dat meer wordt wel. Ik zie wel dat de tool meer waarde oplevert. Je zou de meer waarde moeten kunnen uitreken in tijd en dan de tijd weer kunnen uitreken van hoeveel tijd stop je erin. Omdat er uit te krijgen, het moet wel iets opleveren er moet een winst factor in zitten. Bij een uitgebreide tool die allemaal checks gaat doen vraag ik mij af, want daar moet je ook veel voor gaan invoeren.

Wanneer je feedback op je requirements krijgt die je invoert, zal je dat wat vinden?

Op zich lijkt mij dat wel grappig, alleen door een platte tekst in te voeren krijg je geen feedback. Dus je moet allerlei dingen gestructureerd gaan invoeren en het ligt aan die tool hoe je het doet en hoeveel tijd het gaat kosten wanneer je goeie feedback terug krijgt.

Het is dus het leren van een taal hoeveel tijd dat kost voordat je zo'n tool kan gebruiken.

Het leren boeit mij niet, ook al duurt het een halfjaar. Als het na die tijd na de leer curf maar genoeg oplevert. Je ziet natuurlijk heel veel dingen die gewoon veel tijd kosten op te registreren en die weinig opleveren.

tijd.*wel*

Eenvoudig te leren, vind je dat belangrijk?

Ja met een handleiding.

eenvoudig. *belangrijk*

De respondent heeft de volgende kaartjes uitgekozen, op volgorde van belangrijk naar minder belangrijk:

Eenvoudig te leren -> Tool ondersteuning -> Hangleiding -> Meertalig.

Nog een paar korte vragen, het gaat hierbij meer over de talen. Heb je wel eens met een gestructureerde taal gewerkt?

Nee

gestructureerd e.*niet*

En met een formele taal?

Nee

formele.*niet*

Kennis daarvan?

Nee.

Dan bedank ik je voor dit interview.

B.3 Interview 2 Applicatieontwikkelaar die requirements opstelt

Bedankt dat je mee wil werken aan dit interview. De bedoeling is dat ik dit interview via de STARR methode doe. Eerst wil ik de situatie bespreken, dan de taken waarmee je bezig bent, om een beetje een beeld te krijgen in welke context je werkt. Dan ga ik verder op de activiteiten die toegespitst op de requirements, welke activiteiten je daarin uitvoert. Dan hebben we de R voor resultaat, de uitkomst van de requirements en daarna nog een stuk van een reflectie. In dit interview gaat het niet over een specifiek project binnen VitalHealth maar over de requirements in het algemeen binnen VitalHealth.

De eerste vraag is, bij welke projecten ben je nu werkzaam?

Dat zijn met name de patiënten portalen, dan hebben we eigenlijk twee klanten, Isala patiënten portaal en CWZ het Canisius-Wilhelmina Ziekenhuis. En apart daarvan hebben we ook nog een patiënten portaal van Abott. Dat is een farmaceutisch bedrijf die humira produceert. We proberen Isala en CWZ op een soort basis patiënten portaal te krijgen. Omdat we een stuk basis functionaliteit hebben met een stuk specifieke functionaliteit voor elke klant. **project.***PP*

Wil je de twee patiënten portalen gaan samenvoegen?

De moeilijkheid waar wij continue mee zitten is de requirements zijn ongeveer voor tachtig procent wel gelijk, maar die twintig procent daar gaat het om. En hoe ontwikkel je nou de software zo slim mogelijk zodat je toch een soort gemeenschappelijke basis kunt neerleggen maar dat je toch klant specifieke oplossingen kunt toepassen bij een bepaalde klant.

De requirements krijg je dan van verschillende klanten binnen en dan ga je die met elkaar vergelijken?

Ja, wat we nu doen is we hebben een gebruikers overleg, daar zitten nu dus twee klanten in, CWZ en Isala, die dienen daar requirements in en vervolgens gaan we kijken zitten daar gemeenschappelijke requirements in of zijn het allemaal specifieke. Vervolgens gaan we over die gemeenschappelijke requirements met elkaar in gesprek om te kijken of ze echt wel gemeenschappelijk zijn, bedoelen we nu echt wel het zelfde met elkaar. En dat is echt niet altijd zo, vervolgens brengen we dan een offerte uit en de kosten van het project worden dan gedeeld door beide partijen. **verzamelen.** *overleg stakeholder*

Jullie zijn dus bezig om een applicatie te bouwen op basis van de gemeenschappelijke requirements.

Ja, wij hebben dus een basis patiënten portaal, daar zit dus bij allebei een stukje specifieke software in. En vervolgens hebben ze gemeenschappelijke wensen. Ze hebben ook met elkaar contact, buiten ons om dus dan spreken ze elkaar ook.

Wat doen jullie dan met de specifieke requirements?

Die bouwen we dus ook in maar die worden in de specifieke laag, voor Isala hebben we een basis laag en een specifieke laag en daar bouwen wij het dan in, maar daar moeten ze ook apart voor betalen.

Met <project 3>, ben je daar ook mee bezig?

Nee, <project 3> dat is, daar ben ik wel heel erg druk mee geweest omdat op te zetten en aan de eerste release hebben we gewerkt, in vier vijf maanden hebben we daar versie één neergezet. Inmiddels ligt het volledig in India. De opdrachtgever voor <project 3> is Mayo Clinic in de VS. **project.***MP*

Wat zijn jou taken binnen de projecten?

Wij hebben het allemaal niet zo heel erg gescheiden, als we praten over patiënten portalen dan hebben we een functioneel consultant en mijn rol is meer een architecten rol maar dat is niet altijd duidelijk te scheiden. Want functioneel spreek ik ook met de klant, dus ja, de specificaties proberen duidelijk te krijgen bij de klant dat is altijd nog wel een moeilijk probleem. Want die klant die weet het vaak ook niet, en vervolgens dat op papier zetten en proberen te begroten, om daar een prijs kaartje aan te hangen en dan vervolgens het realiseren van de software. En dan ook kijken of het allemaal wel technische kan. **rol.***RE*

Want stel jij zelf ook de requirements op? Die je dan met een klant bespreekt.

De klant die heeft natuurlijk een wens of die heeft een probleem en wil daar een oplossing voor maar hij weet alleen het probleem, dit loopt niet. Vervolgens vraag hij aan ons van hoe kan ik dat nou oplossen in dat patiënten portaal. Dan moeten wij een oplossing bedenken die ook past functioneel maar ook in de applicatie, niet strijdig is met de architectuur. En dat is een beetje heel divers want ik bouw ook zelf mee. Dus eigelijk doe ik al de facetten. **rol.***ontwikkelaar*

Wat heeft bij jou de voorkeur bij het opstellen van de requirements, de requirement declaratief of procedureel opstellen? Dit is beschrijf je bij requirements wat het systeem moet doen of hoe het systeem moet werken?

Ik snap niet helemaal het verschil.

Je hebt dus verschil, in de requirements kan je gaan beschrijven van wat het systeem moet gaan doen en je kan ook gelijk al gaan beschrijven hoe het systeem moet gaan werken.

Dus eigenlijk zeg je van wat willen we bereiken en hoe gaan we dat doen, de oplossingsrichting zelf.

Ja vooral hoe noteer je dat in de requirements. Beschrijf je tijdens het requirements engineeren van hoe je het probleem gaat oplossen.

Als we kijken hoe dit bij Mayo is gegaan hebben we daar iedere week, of twee keer in de week, hebben we daar telefonische call gehad van twee uur, zeg maar vier uur per week en dan met z'n tweeën met de collega's van mayo en dan hebben we de functionaliteit die zij wensten besproken. Maar zij zeggen van wij willen bijvoorbeeld meerdere filialen kunnen aansturen en vervolgens moet wij dan is de vraag eigelijk aan ons hoe zouden we dat kunnen doen. En vervolgens hebben wij dat dan verwerkt in een functioneel design en gezegd van volgens mij zal het zo kunnen. **verzamelen.** *overleg stakeholder*

Dus heb je dan beschreven hoe je het gaat oplossen?

precies, hoe je het gaat oplossen en het probleem zelf is besproken geweest in die conference calls.

Dus die zijn niet expliciet opgesteld of genoteerd in de requirements?

Nee, dat is niet expliciet genoteerd. Er is wel bij het begin van het hele traject dat moet ik er eigenlijk wel bij vertellen hebben we wel een soort high level requirementsdocument gekregen van mayo zelf van we willen een nieuw systeem waarmee we patiënten kunnen doorverwijzen naar mayo vanuit huisartsen. En dat moet voldoen aan een heleboel aspecten, dat was een enorme lijst waarin ook opmerkingen staan als security en performance maar ook functionele requirements. Maar wel op een heel hoog niveau maar wel wat ze wilde. **verzamelen.** *document*

Maar wat zou jou voorkeur hebben wanneer je voor een project requirements gaat opstellen? Dat je de requirements dan opstelt in wat je wilt of hoe je het wil gaan oplossen?

Ik denk dat je altijd moet beginnen met wat wil die klant, wat is eigenlijk het probleem. Wat gaan we oplossen wat is eigenlijk het issue wat is de business case die er ligt. Maar heel vaak weet een klant het nog niet goed en dan moet je met hem in gesprek om de requirements eruit te halen. **declaratief.** *voorkeur* **verzamelen.** *overleg stakeholder*

In de requirementsdocumenten die ik heb onderzocht wordt er soms geen gebruik gemaakt van een verklarende woordenlijst, is het een beetje standaard om verklarende woordenlijsten te gebruiken. In een verklarende woordenlijst leg je uit wat bepaalde domein woorden betekenen.

Weet je wat ook lastig is, in het geval van <project 3> is dat je een heel groot project hebt en die ga je dan vervolgens op delen in deelgebieden. En die deelgebieden daar hebben we een ontwerp voor geschreven. Maar die hebben natuurlijk wel relaties met elkaar. En als je dan één zo'n los ontwerp eruit trekt dan zie je veel begrippen en die kom je dan tegen in andere documenten worden die dan uitgelegd. Dat vond ik één van de meest lastige dingen omdat hele project er ligt allemaal requirements omdat nou logisch op te splitsen in brokken die je kunt handelen en ook kan managen qua planning maar ook qua ontwerp en techniek maar toch het is nooit een helemaal op zich zelf staand stukje, het heeft altijd een relatie met een ander deel. Wat we dan deden in ons ontwerpen is dat we dan de relaties vastleggen naar andere documenten. Maar we hebben niet een begrippenlijst gehanteerd. **woordenlijst.***wel*

Krijg je wel eens vragen van klanten dat ze bepaalde begrippen of woorden in het document niet snappen?

Ja, dat gebeurt weleens. Maar ja, je neemt heel gauw de begrippen van die klant aan. Wij praten bijvoorbeeld in ons patiënten portaal over zorgverleners en bij de klant hebben ze het over healthcare professionals dat zijn ook wel zorgverleners maar die begrippen neem je dan over in je documenten omdat dat de kreten zijn die zij zo goed kennen. En afkortingen is ook zo iets, die proberen we in onze documenten wel te listen. **lex. stakeholder.** *wel* **woordenlijst.***wel*

Die leg je wel vast?

Ja, voor zo ver het nodig is.

En wanneer het document dan naar een ontwikkelaar gaat, krijg je dan weleens vragen over begrippen die men niet snapt?

Ja, niet alleen de begrippen maar ook requirements en ook de oplossingsrichting. Maar dat is volgens mij de kern van veel problemen hoe schrijf je nou, voor wie schrijf je ook dat is ook belangrijk. Van als je in een functioneel ontwerp die we maakte voor <project 3> en die gingen naar de klant en die klant moet zich daarin herkennen. Van inderdaad dit is mijn probleem en zo kunnen we het oplossen maar dat wil nog niet zeggen dat dat ook voldoende is om aan een ontwikkelaar te geven. **lex. ontwikkelaar.** *wel* **interpretatie ontwikkelaar.** *wel* **doel**.*beide*

Denk je dat je dat kan oplossen?

Er zijn natuurlijk mensen die dat vervolgens gaan vertalen in een technische design ik geloof daar niet zo heel erg in want die ontwikkelaar moet ook functioneel snappen wat hij aan het doen is. Wat wij deden in het geval van <project 3> dan dat wij de functionele designs doorstuurden naar in dit geval het ontwikkel team in India en die gingen zich daarin verdiepen en doormiddel van calls gingen ze dat ook uitleggen totdat ze het echt begrepen. **ambiguïteit.** *voorkomen*

Het mondeling toelichten van de requirements?

Jazeker het mondeling toelichten van de requirements. Ik denk dat je daar bijna niet aan ontkomt of je gaat in zo'n detail graad of niveau dat het bijna voor de klant en de ontwikkelaar niet te begrijpen is. Het blijft dus altijd wel de vraag voor wie schrijf je. Dat vind ik wel een lastige.

En als je dan kijkt hoe je de requirements opstelt, wat zou bij jou de voorkeur hebben. De requirements opstellen in tekst, dus stukken tekst. Of dat je het gestructureerd genummerd onder elkaar opschrijft, per requirement een zin.

Ik zit even te denken hoe we dat bij mayo bijvoorbeeld gedaan hebben. We hebben eigenlijk een soort document waarin requirements gegroepeerd waren met een nummer en dan waren er ook subnummers. Dat waren meer steekwoorden of zinnen en daarnaast hadden we een document waarin de requirement een wat breder werd uitgelegd. We hadden een soort overall lijst waarin we ook konden zeggen van nummertje acht punt twee die requirement daar willen we het over hebben.

Die waren dus weergegeven in een lijst?

Ja, maar goed dat is dus de input geweest, die requirements lijst dus de beschrijving die ook nog wel vrij globaal was, is voor ons de input geweest om naar zo'n functional design te gaan. Ik moet er eigenlijk nog één ding bij zeggen, daartussen hebben we ook nog iets gedaan. Wij zijn op basis van die requirements lijst die van mayo kwam zijn we daar geweest, in Minnesota, en hebben we die lijst helemaal door gesproken en dat hebben we vertaald in een high level document.

verzamelen. *overleg stakeholder*

Je hebt dus eerst die requirements doorgesproken en dat heb omgezet naar het functionele ontwerp?

Precies, we hebben nog een slagje dieper, de hele week hebben we daar gezeten en de hele week gepraat over die requirements en iedere requirement een slagje dieper proberen te bespreken met een bord erbij van begrijpen we elkaar en dat hebben we allemaal in een document gezet. Dat is dan het HLD, high level design geweest. En op basis daarvan hebben we de breakdown structure later gemaakt. Om te zeggen van hoe gaan, als dit het high level design is hoe gaan we dat dan opsplitsen in beheersbaren brokken die we ook kunnen managen en voor ieder brok maken we dan een echt design functioneel design die weer terug gaat naar mayo voor akkoord en dan gaan we naar de ontwikkelaar.

Heb je dan per requirement een brok gekregen of zitten dan meerdere requirements in een brok?

Meerdere requirements, omdat het zo complex was. Dat vond ik ook lastig je hebt requirements en je hebt een nette lijst van requirements gegroepeerd en dan lijkt het net of dat allemaal brokjes zijn en dat is ook best wel gegroepeerd maar ook heel veel dingen hebben met elkaar te maken. En als jij dan zelf zo'n lijst door gaat dan moet je toch als je gaat ontwikkelen een paar dingen samen nemen in een brok waar je dan een ontwerp voor maakt. Dat is een lastig stuk trouwens vond ik, omdat op te splitsen in beheersbare delen. In componenten zodat je dat kan ontwerpen maar ook kan realiseren en ook nog in de juiste volgorde want dingen hebben weer een afhankelijkheid met elkaar. Dat vond ik erg lastig.

Je hebt dan een requirement genomen en die heb je dan verder uitgewerkt en die heb je een stukken tekst genoteerd?

Ja, wij hebben onze requirements hebben we heel tekstueel beschreven en toen ik daarmee bezig was heb ik daar wel vaak gedacht van kunnen we niet veel meer grafische schema's, workflows of zo opnemen, diagrammen. Onze requirement zijn heel tekstueel geweest en als ik daar nog eens op terug kijk dan denk ik soms doet een plaatje wonderen en dat hebben we ook steeds meer proberen te doen maar dat zou ik nu misschien anders gedaan hebben.

Wat denk je voor een ontwikkelaar, wat zou dan makkelijker zijn, per requirements of tekstueel?

Ja dat is een beetje moeilijk, dat ligt er helemaal aan. Ik denk dat een combinatie een flow van een proces of zo proces beschrijving je kan het helemaal prima beschrijven maar als je daar een plaatje bij doet met een flow dan gaat dat veel meer leven. Daar zijn wij denk ik nog niet, ik denk dat er daarin een verschil is tussen <project 3> en <project 1> als het daarom gaat volgens mij doen ze het bij <project 1> al meer met grafische weergaven. Daar zit wel wat verschil denk ik wij zijn heel erg tekstueel bezig geweest.

lijst.tekstueel. *geen voorkeur*

Heb je het weleens meegemaakt dat een ontwikkelaar een requirement op meerdere manieren interpreteerde?

Ja, dat zeker.

interpretatie ontwikkelaar. *wel*

Kan je hier een voorbeeld van geven.

Ik heb niet echt een specifiek voorbeeld maar het is wel zo iets dat eigenlijk altijd het geval is want hoe je het ook opschrijft, dat maakt wel uit hoe je het opschrijft, er zijn altijd wel het kan altijd anders geïnterpreteerd worden dan jij bedoelt. Dat is ook de kunst omdat goed te doen en ik heb niet direct een voorbeeld zo hiervan.

reden interpretatie. *zinsstructuur*

Er zijn genoeg voorbeelden maar weet je het gaat vaak om hele complexe, wat ik al zei het is een hele complexe applicatie met allerlei verbanden en we hebben daar dan een brokje van genomen en daar hebben we een ontwerp van gemaakt van dat moet jij gaan maken maar het heeft hier raakvlakken met andere brokjes en die persoon moet dat wel snappen.

De ontwikkelaar moet dan kennis van het domein hebben? **reden interpretatie.** *domeinkennis*

Ja, en daarom ontstaan er best wel eens misverstanden.

Denk je dat je dit kan oplossen?

Ik geloof persoonlijk niet dat je alles in een document kunt zetten. Ik geloof wel dat je dat moet doen en bij voorbeeld wat wij dan deden iedere dag met elkaar een call er dan helemaal door heen lopen van dit hebben wij bedoeld, dit is de relatie met andere producten. Dan liepen we door de requirements heen met de ontwikkelaars. En dan werd het duidelijk dus een stukje toelichten. En ik ben trouwens ook in India geweest toen we met <project 3> starten om een week gewoon dat met een whiteboard enzo uit te leggen en dat doet gewoon veel, dat doet echt veel. **ambiguïteit voorkomen.***wel*

Merk je dan ook dat er verschil in Nederland en India zit, dat ze in India de requirements op een andere manier kunnen interpreteren als in Nederland?

Nou nee, ik geloof niet dat Nederlanders het anders interpreteren. Je hebt een cultuur verschil en de taal is verschillend, dat heb je dat maakt het niet makkelijker maar ik denk als je gewoon aan een Nederlandse ontwikkelaar had gegeven zonder betrokken te zijn bij het project en al de gesprekken dat hij net zo veel tijd in moeten steken. Kijk je moet niet vergeten wij waren al twee maanden bezig met al die gesprekken daar en omdat allemaal op papier te zetten en om te discussiëren met elkaar. Dan gaat het naar een ontwikkelaar en die heeft die voorkennis niet. **domeinkennis.***hoe*

De domeinkennis probeer je dan met die sessie over te brengen naar de ontwikkelaars? **domeinkennis.***hoe*

Ja want er zitten natuurlijk ook heel veel in zo'n sessie in zo'n praat sessie blijkt dat een klant het ook niet helemaal precies weet. Dat is echt waar, klanten weten ook niet wat ze willen. Door met elkaar te praten komen zij ook tot andere inzichten of tot... Op het moment dat je een ontwikkelaar hebt en die is die fase helemaal voorbij, dus dan kan het veel sneller zeg maar.

Heb je zelf technieken waarmee je probeert te voorkomen in een document dat requirements op meerdere manieren kunnen worden geïnterpreteerd?

Nee niet echt, Ik vind dat wij een verbeterslag, wat we ook naar een klant sturen van de week had ik daar nog een voorbeeld van. Dan sturen we een offerte naar de klant en dan staat dan één zinnentje die is gewoon voor heel veel uitleg vatbaar. Dat is en daar ontstaan de discussie punten en dat is niet nodig als je drie zinnen extra eraan wijdt. Het is zo belangrijk dat je in jou communicatie naar klanten gewoon duidelijk zegt wat je doet en vooral wat je niet doet. Wat ik vroeger, of vroeger in mijn vorige werk kring deden we dat gewoon ook heel expliciet in de offertes dit doen we en dat betekent dat we dit en dit en dit in ieder geval niet doen. Ik zie hier weleens documenten of specificatie of requirements naar de klant gaan dus een offerte en dan wordt er verwezen naar de requirements deze requirements gaan we doen voor die prijs. Dat ik denk van ja, ik kan hier een groot verhaal van maken dan klopt die prijs echt niet meer. **ambiguïteit voorkomen.***niet* **invloed ambiguïteit***.wel*

Heb je wel eens gezien dat het ook gelijk invloed heeft op de software, dus dat je zag dat er verkeerde software werd gemaakt op basis van requirements die je op meerdere manieren kan interpreteren.

Ja, dat is wel gebeurd ja maar dat komt met name voor omdat die requirements zijn verzameld door een functioneel consultant vaak met de opdrachtgever en die worden op een vrij hoog niveau bedacht. En wat er dan gebeurde dat ze die doorstuurde naar de ontwikkelaar op de afdeling in India. Als je dat op een hoog niveau laat staan en je kan zelf al vijfentwintig vragen stellen, dan moet je niet denken dat je iets heel consistents terug krijgt. Wat we daarin miste in <project 3> dat is dat we op detail niveau beschreven hebben hoe een proces gaat lopen in die applicatie bijvoorbeeld wanneer je dat niet doet en je stuurt alleen maar die requirements, ik heb dingen gezien van "econsult moet worden geïmplementeerd worden" punt. Als requirement en dan nog drie zinnentjes erbij en dat was het dan en dat gaat niet goed. Ik denk zelf dat als je dat vertaald in een goed functioneel ontwerp los je veel van dit soort problemen op. **invloed ambiguïteit***.wel*

Hoe controleer jij dat je al de requirements die je in een document hebt opgesteld dat die ook worden opgenomen in de software?

Als we dan bijvoorbeeld terug gaan naar <project 3> dan maken we een highlevel design dat hebben we uiteen geslagen in allemaal brokken, ongeveer dertien en nog wel meer denk ik. En voor ieder brokje hebben we een functioneel design gemaakt en dat functioneel design is een iteratief proces. Dat maak je en bespreek je in een call. Op een gegeven moment is die definitief en dan vragen we gewoon een akkoord, een formeel akkoord.

Het gebeurde regelmatig dat men later tot andere inzichten kwam en dat kan gewoon dat hebben we allemaal, maar dan konden we heel duidelijk zeggen van oké maar we hebben dit afgesproken en dit is het akkoord dus dan gaan we kijken wat de impact is in tijd en geld. Dat is op zich redelijk beheersbaar op deze manier. Maar hoe controleer je het, je moet gewoon de klant laten tekenen. De klant moet dan bij de acceptatie weer tekenen of het klopt wat hij in zijn requirements heeft getekend.

Ik heb de documenten geanalyseerd en daarin een paar dingen gevonden deze hebben we net ook besproken. Ik heb hier vier kaartjes die ik één voor één aan jou geef. Hier staan eigenschappen van requirements op. En dan vraag ik aan jou of je wilt beoordelen of deze al in de documenten aanwezig zijn of niet. In de documenten in het algemeen van VitalHealth.

Als ik bijvoorbeeld verifieerbaar heb, zie je dat terug in de requirements.

Dat is best wel eens lastig om dit terug te vinden. Want de applicaties zijn dermate complex aan de applicaties zelf kan je die requirements moeilijk terug vinden. Dit is best een moeilijke vraag. Bijvoorbeeld als je een heel ingewikkeld autorisatie model hebt en je hebt dat geïmplementeerd en je hebt dat laten ontwikkelen hoe kan je dan heel snel in zo'n applicatie zien dat dat erin zit. Dat die requirements zijn opgelost. Dat zou alleen maar kunnen dan zou je eigenlijk een soort hoe noem je dat een soort test scenario op nemen in je documenten. Dan kan je aan het einde de scenario's uitvoeren om te checken of die requirements zijn afgedekt in de applicatie, daar kunnen wij nog wel een slag maken misschien. **verifieerbaar.***niet*

En bij consistent, dat je requirements die met elkaar kunnen conflicteren?

Dat komt zeker voor, maar dat zit hem met name bij de klant. Als ik maar weer naar mayo ga in die grote lijst waar ik het net over had daar stonden echt dingen in die met elkaar conflicteerde. En dat ziet de klant ook niet altijd, dan ga je in gesprek en dan komt hij ook totdat inzicht. **consistent.***niet*

En wanneer kom je hier meestal achter, tijdens het ontwikkelproces of al eerder?

We proberen dat tijdens de analyse fase te voorkomen, maar ik denk dat we in onze ontwikkelfase hier nog best wel eens tegen aan lopen.

En als je hier hebt eenduidig, zijn de requirements dat, dat ze niet op meerder manieren kunnen worden geïnterpreteerd?

Nee, dat is wel vaak zo. **eenduidig.***niet*

Dan hebben we ontwerp onafhankelijk, dat je tijdens je requirementsfase nog niet helemaal beschrijft hoe het ontwerp eruit gaat zien?

Wat bedoel je hiermee?

Dat de requirements los van de implementatie worden opgesteld?

Zo werkt het meestal niet, want wij krijgen de requirement van de klant en die zijn ontwerp onafhankelijk die zeggen ik heb een probleem met dit en daar wil ik een oplossing voor. En wat wij dan altijd doen is toch wel gaan nadenken hoe we dat dan kunnen implementeren. En hoe we de oplossing kunnen bieden. **implementatie.***niet*

Als je nu die kaartjes bekijkt, welke vind je dan belangrijk, kan je ze een beetje op volgorde leggen van welke voor je belangrijk zijn en welke wat minder.

Deze vind ik belangrijk, eenduidigheid.

En dan consistentie.

Traceerbaarheid

Ontwerponafhankelijk

De respondenten heeft de kaartjes op de volgende volgorde gelegd, van belangrijk tot minder belangrijk:

Eenduidig -> Consistent -> Traceerbaar -> Ontwerp onafhankelijk.

Ik heb in mijn onderzoek een aantal talen geselecteerd waarmee bepaalde dingen opgelost kunnen worden in de requirements. Dat zijn gestructureerde talen, die beschrijven dan hoe je de requirements moet opstellen, dus welke woorden je mag gebruiken en hoe je zin gestructureerd is. Om de eigenschappen die voor jou belangrijk zijn om die op te lossen in de requirements, hoeveel tijd zal je dan ongeveer overhebben om een taal te leren, de tijd die je erin wil investeren?

Als dat deze problemen oplost, dan is dat goed. Dan denk ik dat ik daar wel voor opensta. Want wij lopen hier vaak tegenaan in de requirements. **tijd.***wel*

Dan heb ik hier een ander stapeltje met kaartjes, om te kijken of je van deze eigenschappen gebruik maakt in de requirements. Als je bijvoorbeeld beeldspraak hebt, maak je daar weleens gebruik van in de requirements?

Nee, ik zelf niet. **beeldspraak.***niet*

Een gezegde?

Nee — **gezegde.***niet*

Afwisseling in de structuur van een zin, hetzelfde opschrijven maar in een andere structuur?

Ja, ik herken dit wel. — **afw. structuur.***wel*

Gebruik maken van tussenzinnen?

Ja — **tussenzinnen.***wel*

En spreekwoorden?

Nee, dat ook een beetje persoonlijke stijl dat mensen hebben. — **spreekwoord.***niet*

De respondent heeft de volgende kaartjes uitgekozen:
Wel in de requirements: Afwisseling in structuur zinnen, tussenzinnen
Niet in de requirement: Beeldspraak, spreekwoord, gezegde.

Welke eigenschappen zal zo'n taal moeten ondersteunen, degenen die je nu ook al gebruikt.

Waar het om gaat is in welke vorm maak ik mijn requirements het meest helder. Je hebt het over taal maar wat ik zelf ook vaak gebruik zijn opsommingen. Is dat ook wat je bedoelt in de zin van taal. — **opsommingen.***wel*

Het is niet een eigenschap die ik hier tussen heb liggen, op zich zal dat er wel eentje kunnen zijn. Ik zal de vraag opnieuw stellen. Van deze twee maak jij nu gebruik, wil je die blijven gebruiken met een gestructureerde taal?

Ja.

En die andere?

Een gezegde kan wel kernachtig zijn dat je met een kort gezegde iets heel goed duidelijk maakt. Dus daar kan ik mij iets bij voorstellen.

Wil je ze ook rangschikken op welke je belangrijk vind?

Ja is goed

De respondenten heeft de kaartjes op de volgende volgorde gelegd, van belangrijk tot minder belangrijk:

Tussenzinnen -> Afwisseling in structuur zinnen -> gezegde.

Verwijs je wel eens vanuit een requirement naar een andere requirement.

Ja, dat komt wel vaak voor. — **verwijzing.***wel*

Is dit essentieel voor jou omdat te blijven doen?

Ja ik denk het wel, daar ontkom je niet aan. — **verwijzing.** *belangrijk*

Ik heb je al een beetje over de talen verteld. Hier heb ik nog een aantal eigenschappen die een taal kan hebben, dit zijn losse eigenschappen. De eerste is dat er een eenvoudige handleiding beschikbaar is voor de taal zodat je deze snel kan leren. We gaan ze zo weer op volgorde leggen. Dat de taal meertalig is, dus zowel Nederlands als Engels en eenvoudig te leren.

De respondenten heeft de kaartjes op de volgende volgorde gelegd, van belangrijk tot minder belangrijk: Meertalig -> Eenvoudig te leren -> Handleiding -> Tool ondersteuning.

En dan nog drie korte vragen, heb je wel eens gewerkt met een gestructureerde taal, dat je die gebruikt hebt als taal voor bijvoorbeeld requirements te specificeren.

Nee, ik heb daar geen ervaring mee. — **gestructureerd.** *niet*

Formele talen, daar weleens meegewerkt?

Nee — **formeel.***niet*

Dan bedank ik je voor het interview.

B.4 Interview 3 Applicatieontwikkelaar

Ik wil dit interview via de STARR methode doen. We bespreken eerst de situatie, waar je mee bezig bent en welke projecten je bij betrokken bent. En dan de T van taak, dus welke taak je hebt en dat in relatie met requirements. Daarna activiteiten, welke activiteiten je in projecten uitvoert. De R van resultaat, de resultaten van de requirements en hierna een reflectie hoe dingen verbetert kunnen worden.

Wil je kort vertellen in welke projecten je nu werkzaam bent?

Ja nu zit ik in..., projecten wij noemen het eigenlijk meer, het project is eigenlijk meer het hele CHM, collaborate health management, en voor het product VitalHealth CHM KISS daar doen wij dus allerlei product ontwikkeling voor en nu ben ik bezig met een integratie met een oogartsen applicatie die heb ik nu bijna afgerond. We hebben net gisteren afgerond voor <project 2> een stukje maatwerk, voor declaraties. Twee jaar geleden hebben ze ons product aangeschaft en daar zijn ze nu ook continue mee bezig en daar zit een heel proces achter omdat daar ook veel artsen mee werken. Daarvoor hebben we stukje maatwerk gedaan om het declaratie proces te optimaliseren zodat één of twee personen dat af kunnen handelen. Elk kwartaal komen de kwartaal controles, of die declareren ze weer. En ik ben nu bezig, ga ik morgen of volgende week mee beginnen voor het SHL, ook een klant van ons, die gebruikt ook ons KISS. Verschillende zorggroepen hebben ons KISS nu aangeschaft, twee nieuwe zorggroepen.

project.*KIS*

Is dit allemaal hetzelfde project, SHL en CHM?

Ja, ze gebruiken allemaal het KIS allemaal het project CHM. Alleen de ene heeft meer, een stukje maatwerk, het zit allemaal in één product en het zit allemaal in het CHM project.

project.*KIS*

En dan voor verschillende klanten?

Ja dat klopt.

Wat zijn jouw taken in de projecten, welke rollen heb jij?

Ontwikkelen, ik ben een ontwikkelaar. Via de consultants, zeg maar <productmanager>[11], <consultant>[12] en <consultant> stellen vaak een, en vooral <productmanager> die bepaalt de requirements want die doet overleg met de klant wat ze willen en dan komt die bij ons en dan bespreken we de planning van dit gaan we nu doen of dat gaan we over twee drie weken doen, ik heb nu net een paar voorbeelden genoemd die gaan we dan uitvoeren. En dan ga ik echt ontwikkelen.

rol.*ontwikkelaar*

En hoe krijg jij de requirements, krijg je die van hun?

Ja per mail, of via het document en dan worden ze mondeling toegelicht.

aanlevering.*mail*
aanlevering. *document*

Maar wanneer ze iets mondeling toelichten, heb je dan ook al een document ontvangen van hun?

Ja

Dus je doet dat meestal op basis van het document?

Ja, soms is dat gewoon een korte mail of zo of een word document. Het ligt dus wel vast, negen van de tien keer staat het vast. En we werken ook veel met issues, dan hebben we een issue en dan staat daar het hele stuk in en dan heb ik een vraag en dan reageert <productmanager> er weer op.

aanlevering. *supportsysteem*

Hoe zet hij de requirements om naar de issues?

Hij maakt eigelijk gelijk een issue aan, ja sommige.... , kijk het is een beetje soms maakt hij zelf een issue aan maar wat hij soms ook wel eens doet is, klanten loggen zelf ook wensen en dat wordt het eerst een ticket, klanten komen eerst bij ons een ticket loggen, die wordt automatische omgezet naar een issue en <productmanager> die beoordeelt alle issues van al de klanten wensen. En dan zet hij er soms commentaar bij en die verzamelen we vaak tot, twee keer per jaar krijgen namelijk al de klanten een functionele release update. Dan kunnen ze zelf, we hebben ook een gebruikers overleg van te voren en dan kunnen ze zelf aangeven, dan krijg je bijvoorbeeld zoveel dagen aan ontwikkel tijd wat wij mogen besteden voor de functionele release en dan kunnen ze er zelf prio's van dat issue die willen we heel graag hebben en die wens, die een klant zelf gelogd heeft, van die moet echt acuut in de volgende update aanwezig zijn.

aanlevering. *supportsysteem*

De requirements die komen binnen via de klant, dus die maakt de klant zelf eigenlijk?

Ja kijk, sommige ook direct want <productmanager> en de consultants komen ook vaak bij de klanten zelf en dan hebben ze ook wel weer van, dan zijn ze opdat moment bezig en dan zeggen we willen graag dit en soms wil een klant zelf bepaalde functionaliteit en dan is het speciaal maatwerk, dan wordt het ingepland en uurtje factuurtje geregeld. Want nu bijvoorbeeld <project 2> waar ik nu mee bezig ben geweest, die hebben gewoon een wens, dat is een behoorlijke wens voor hun geweest. Dit speelt alweer wat langer want hier zijn ze dan ook op gesprek voor geweest.

verzamelen. *overleg stakeholder*

Hoe kwam die wens binnen, via de requirements of via een wens?

Dit is meer hoe de consultant dat registreert, die heeft gewoon contact met de klant gehad. Die hebben dan vergaderingen hier over gehad. Dat ze daar een keer geweest zijn en het hele proces een keer besproken hebben.

Heeft hij hiervoor ook iets opgesteld?

Ja, daarvoor heeft <productmanager> een document opgesteld. En daar hebben wij, dat klopt. Daar heb ik een heel goed voorbeeld van dat is het complete declaratie proces in dit geval helemaal compleet uitgewerkt en wij hadden het document nou die heeft <productmanager> met <applicatieontwikkelaar> en ik met z'n drieën hebben we dat besproken, dat hebben we dan in fases gemaakt. Puntje één hebben we in de afgelopen week gebouwd en in het nieuwe jaar gaan we de volgende fase twee en drie doen. **domeinkennis.***hoe*

Ok, hij heeft de requirements opgedeeld in een paar fase.

Ja, dit wilde ze bij <project 2> namelijk zelf ook. Dat doet bij ons vooral de consultant.

Dus die stelt echt de requirements op.

Ja, dan samen met de klant zelf

Want stelt zelf geen requirements op?

Nee, heel soms maar eigenlijk niet. Soms maken we voor ons zelf een issue aan van dit moet in de toekomst een keer gebeuren. Een wens voor ons zelf, die gaan in een verzamelbak, het voorbeeld van <project 2> is een goed voorbeeld van klanten die requirements opstellen en wij hebben dan twee keer per jaar die functionele release dat klanten ook zelf wensen kunnen indienen en dan gaat <productmanager> met ons samen erdoor heen, nou jongens dit is de bak voor nu, voor deze functionele release, en dan versie 1.3 of zo. En daar worden alle wensen in opgenomen. Wat haalbaar is zeg maar. Dus twee stromen direct met de klant en de klant via het systeem. Precies een klant kan dus ook direct een wens loggen.

En wanneer je dan zo'n requirementsdocument hebt, wat heeft bij jou dan de voorkeur het opstellen van wat het systeem moet gaan doen en wat het systeem moet gaan maken. Dus het opstellen van de requirements declaratie of procedureel. Beschrijf je van wat het systeem moet gaan doen, dus wat het systeem moet gaan oplossen. Of beschrijf je al direct hoe je dat gaat doen?

Ik hou meer van het hoe, dus procedureel. Hoe het moet gaan doen. En de laatste keer is dat ook meer gebeurd van hoe het systeem gaat, van hoe het gaat werken. Met allemaal plaatjes er al bij van daar moet het knopje, dat moet zus en dat zo. Procedureel dus meer. **procedureel.** *voorkeur*

Is er ook onderscheid tussen functioneel en requirements, is dat opgesteld van wat het systeem moet doen. En dat daarna de stap wordt genomen naar hoe het systeem gemaakt moet worden?

In de meeste gevallen wordt gelijk het hoe beschreven, ik heb geen andere voorbeelden. Dat is mijn ervaring. **functioneel.***nee*

Denk je dat er ook nadelen aanzitten, aan het opstellen via het hoe, in plaats van het wat. Dus zitten er nadelen aan het functioneel opstellen?

Ik kijk er meer positief tegen aan, ik zie niet echt nadelen ik vind het wel fijn hoe de consultants nu werken. In een document en dan het hoe beschrijven vooral met plaatjes. Soms wordt het wat technisch. Als <productmanager> op een gegeven moment met het technische er zelf niet uitkomt dan beschrijft hij wel meer het proces, hoe het dan moet gebeuren en dan lichte wij het technisch toe. Van jongens we gaan het zo bouwen. Soms weet een consultant niet hoe technisch iets gemaakt moet worden. Dan hebben ze wel ongeveer een idee komt hij bij ons en dan gaan wij overleggen en dan gaan we een beetje brainstormen van welke gedachte welke kant gaan we op zeg maar. Soms komen ze met echt wensen van we willen dit en we willen dat daar koppelen. Dat betekent dan voor het hele systeem dat het best wel impact heeft. **procedureel.** *positief* **domeinkennis.** *hoe*

Dus de consultant gaat dan alvast nadenken hoe je dat gaat oplossen en wanneer hij er niet uitkomt vraagt hij het aan jullie?

Ja precies. Dus wel de oplossingsrichting want hun kennen ook het product. En dan proberen ze te kijken of het past binnen de huidige structuur.

Heb je in een requirementsdocument wel eens woorden die je niet begrijpt? Uitdrukkingen die er in voorkomen.

Nee, heel soms iets en heel zelden als ik een medische term niet weet. Meer van het is een bepaalde ziekte ofzo en dan vraag ik het de consultant die weet dat meestal wel. **Lex-ontwikkelaar.***soms*

Is het de gewoonte dat een consultant bij een requirementsdocument een verklarende woordenlijst opneemt waarin staat beschreven wat bepaalde domein woorden betekenen.

Als ik er om vraag zullen ze dat wel doen, meestal is het niet nodig. **woordenlijst.** *niet standaard*

Maar het is niet de gewoonte?

Nee het is niet de gewoonte. **woordenlijst.** *niet standaard*

Hoe zorg jij dat je de domeinkennis die een consultant bijvoorbeeld weet dat jij die ook daar kennis van krijgt, dat je die specifieke kennis van woorden krijgt.

Dat vraag ik aan de consultant zelf. En soms krijg ik bepaalde documenten over specifiek dat domein. Die ga ik dan zelf ook inlezen, die lees ik dan door. **domeinkennis.***hoe*

En wanneer je dan iets tegenkomt, dan vraag je het?

Ja als ik iets tegenkom dan vraag ik het eerst aan een consultant en soms weet een consultant het ook niet dan bel ik zelf even de klant op. Een goed voorbeeld is met dat telemc platform met die oogapplicatie functioneel is het helemaal beschreven van zo moet het worden. Maar ja dan wordt het technisch uitgewerkt en dan wordt er weer rechtstreeks met de ontwikkelaar gepraat. Want dan houdt het bij de consultant ook wel eens op, maar dat is meer technisch. Functioneel dan vraag ik eigenlijk altijd bij de consultant. **domeinkennis.***hoe* **domeinkennis.** *negatief*

Want de consultant die heeft meestal wel de functionele kennis van hoe het systeem moet gaan werken.

Ja, en anders krijgen we daar documenten van. Maar dat weet een consultant vaak zelf, die bereid zich er ook in voor. Je ziet eigenlijk wel scheiding tussen zeg maar het functionele zit echt bij de consultant en technisch is echt bij de ontwikkelaar.

Je hebt dan een document gekregen van de consultant, wat vind je dan prettiger wanneer de consultant de requirements heeft opgeschreven in een lijstje met aparte requirement of dat ze een stuk tekst hebben opgenomen met beschrijvende tekst waarin staat beschreven hoe het systeem moet gaan werken. Dus het verschil of je tekstueel je requirement opschrijft of dat je per requirement een lijstje hebt.

Meestal is het nu beschrijvend met plaatjes wel met bullets en zo maar niet met nummertjes en prioriteiten die zitten er wel maar ja het is een beetje een combinatie van. Bij ons is het een beetje een combinatie van.

En wat vind je zelf prettiger?

Ik denk dat ik met prioriteiten en met nummers dat ik dat wel fijner vind. Ja dat denk ik wel, dan moeten we misschien nog wel meer verbeteren in ons geval. Als ik terug kijk in de voorbeelden die wij hebben. Dat je bij ons iets meer met dat is prio één. Van de week kwam het voor dat ik drie dingen op mijn bord had en dan vraag, <productmanager> wat vindt je nou het belangrijkste voor de klant, wat moet het eerste gebeuren en dan van die doe ik dan en die dan. **lijst.***voorkeur*

En die requirements kwamen uit een document?

Ja, en daar zat ook wat meer de planning bij, van wat moet het eerste gebeuren.

En dat vind je dus ook een voordeel van wanneer je het in een lijst opschrijft.

Ja

Heb jij wel eens requirements waarvan je niet precies begrijpt hoe die geïnterpreteerd moeten worden, dus dat je denkt dat je het op een verkeerde manier interpreteert dan dat een consultant dat bedoeld?

Ja, soms heb je dat wel ja. Dan trek ik direct aan de bel van hé verklaar dat nader. Als ik even een bepaalde requirement niet begrijp dan vraag ik dat direct. **interpretatie ontwikkelaar.***wel*

Heb je hier een voorbeeld van?

Nee dat is lastig, soms bepaalde dingen ook als ik zeg dat we dan bepaalde requirement krijgen van de consultants dan lees je dat zelf in maar vaak bespreken we het ook altijd even. Dan lopen we ze stap voor stap door en dan gaan we er opdat moment mee aan de slag. Vaak lees je dat van te voren en dan hebben een halfuurtje overleg van we gaan het zo en zo doen. Dan wordt het al snel duidelijk zeg maar. **ambiguïteit. voorkomen.***wel*

Zo probeer je het te voorkomen dat je het verkeerd interpreteert?

Precies, dat ik bewust dat echt alles bespreken van de requirements met de consultants voordat we echt gaan bouwen, altijd. We plannen altijd een meeting en dan bespreken we het even. En we doen een stukje bouwen en dan laten we het ook testen. Eerst de consultant van is het zo goed. **ambiguïteit. voorkomen.***wel*

Maar wanneer jij een requirement verkeerd interpreteert past degenen die het heeft gemaakt ook het document aan, hoe bewaak je dat het de volgende keer weer fout gaar?

Ja, of dat altijd helemaal gebeurd weet ik niet, het is wel de intentie. Met het <project 2> is dat wel gebeurd, daar zat ik niet zelf bij maar bij mijn collega was dat wel. Er waren een aantal requirements paar werden er niet begrepen ook weer terug naar de klant van dat zijn de consequenties en die zijn daarna wel weer aangepast. Toen waren er echt wel een aantal verschillende versies van documenten. **ambiguïteit aanpassen.***wel*

Heb je wel eens meegemaakt dat je requirements verkeerd geïnterpreteerd had en die verwerkt in de software, dat je er later pas achter kwam dat de requirement verkeerd er in zat.

Allicht, ik heb geen voorbeeld maar dat is zeker wel eens een keer voorgekomen. Dat is wel een keer gebeurd, miscommunicatie of iets. **invloed ambiguïteit.***wel*

Wanneer je het document van een consultant krijgt en je hebt de requirements gebouwd hoe controleer je dan dat al de requirements uit het document of die ook opgenomen zijn in de software?

Ja zelf controleer ik al. Ik controleer het altijd aan de hand van het document, dat wel, maar ja daarna als ik denk van dat is goed dan gaat altijd de consultant het eerst testen. Dat is een beetje de controle.

De consultant controleert ook op basis van de requirements?

Ja, zeker. Het wordt eigenlijk twee keer gecheckt. Als ontwikkelaar vergeet je ook wel eens wat, dan komt de consultant van dat moest er nog in. Of soms heb je het niet helemaal begrepen en komt er nog een requirement bij.

Ik heb hier een aantal kaartjes. Ik ben bezig met een onderzoek om de requirements te gaan verbeteren. Daarvoor heb ik een aantal requirementsdocumenten die heb ik bekeken op kwaliteit. Daarin zijn een paar dingen opgevallen, daarom wil ik aan jou vragen welke eigenschappen jij denkt dat er in de requirements aanwezig zijn. Dus de documenten die jij zelf gezien hebt binnen VitalHealth.

Wanneer je dan eenduidigheid hebt, komt die voor in de requirements. Dus dat een requirement maar voor één uitleg vatbaar is zo dat die niet op meerdere manieren geïnterpreteerd kan worden. Zijn de requirements in de VitalHealth documenten voor één uitleg vatbaar?

Niet allemaal, wat ik net als voorbeeld gaf soms heb je een requirement dan denk je dat het dat is en dan bespreken we dat, en dan hoor je van nee dat bedoel ik zus of zo. Versta je dat een beetje onder eenduidig?

Ja klopt.

Dan zijn de requirements niet altijd eenduidig. **eenduidig.***niet*

Consistent, de requirements conflicteren niet met elkaar?

Ja, dat zit er wel in, een requirement conflicteert niet met een andere. **consistent.***wel*

Implementatie onafhankelijk? De requirements zijn vrij van ontwerp en implementatie beslissingen. Bij het maken van de requirements worden nog geen technische beslissingen gemaakt hoe het systeem moet gaan werken.

Nee, want die worden eerst opgesteld en dan worden ze met ons besproken en dan bekeken of het technisch haalbaar is. De requirements zijn niet ontwerp onafhankelijk. **implementatie.***niet*

Verifieerbaar, de requirement is makkelijk te traceren in de applicatie.

Je bedoelt dat wij het in de code snel kunnen opzoeken of echt in de software zelf?

Functioneel, dus als je het gemaakt hebt en je legt je requirements ernaast of je dan snel kan zien of de requirements zijn opgenomen.

Bij <project 2> bijvoorbeeld kan je dat heel goed zien. Dat is een heel concreet voorbeeld want dan zie je ze ook echt, declaratie proces in dit geval kan je dat goed zien. Want bij bepaalde requirements zitten zelfs printscreens. Dan kan je het echt valideren, van dit moet het zijn. Dat is ook een beetje hoe ik het valideer. Dat is niet altijd zo want in dit voorbeeld was het een vrij groot stukje maatwerk en als je bepaalde kleine wensen hebt dan zijn die niet zo uitgebreid uitgewerkt. **verifieerbaar.***wel*

De respondent heeft de volgende kaartjes uitgekozen:
Wel in de requirements: Verifieerbaar, Consistent.
Niet in de requirement: Ontwerp onafhankelijk, eenduidig.

Wil je de kaartjes op volgorde leggen welke voor jou belangrijk zijn voor de requirements? Dus welke jij echt wil dat die in de requirements voorkomen.

Van deze vier kiezen welke ik het belangrijkste vind. Het maakt niet uit of ze er nu in zitten of niet, los daarvan welke voor jou als ontwikkelaar belangrijk zijn.

Dus consistent vind je het belangrijkste.

Ja, dan eenduidig denk ik en dan ontwerp onafhankelijk en dan traceerbaar. Ik denk dat het zo goed is.

De respondenten heeft de kaartjes op de volgende volgorde gelegd, van belangrijk tot minder belangrijk: Consistent -> Eenduidig -> Ontwerp onafhankelijk -> Traceerbaar.

Ik ben bezig om te onderzoek of er een formaat bestaat waarmee de requirements kunnen worden verbeterd en ook opgesteld kunnen worden. Op het niveau van een requirement dat je bepaalde zinsstructuren moet maken, gebruik maken van bepaalde kernwoorden of sommige woorden juist niet omdat die voor meerdere uitleg vatbaar kunnen zijn.

Als je de eigenschappen die je net op volgorde hebt gelegd, zou je tijd willen investeren om zo'n formaat te leren om ermee te kunnen werken? Dus die deze eigenschappen op lossen.

Ja daar zal ik zeker wel tijd in willen investeren, dit vind ik wel belangrijk. Niet dat ik altijd die requirements opstel, dat doen vaak consultants, maar waneer ze dat volgens een goed proces doen met deze eigenschappen daarin dan zal het zeker het proces verbeteren. Ook naar de klant toe, dat de communicatie met de klant beter wordt. Dan hoe je niet in ons geval drie versies te hebben. Het ligt soms ook wel aan de klant. De klant die is soms ook een beetje eigenwijs want dan hebben we iets gemaakt en dan willen ze toch weer iets anders, het blijft lastig. Je kan een prefect document hebben en dan is de praktijk toch weerbarstig. **tijd.***wel*

Hoe veel tijd zal je erin willen investeren?

Hoelang ben je ermee bezig, in totaal denk je. Als je eenmaal zo iets hebt opgestart dat je weet van zo moet ik de requirements opstellen. Kost dat een keer tien uur? **tijd.***wel*

Paar dagen wat ik nu in de literatuur lees.

Laten we een week zeggen, vijf dagen dan zal ik daar wel een halve dag per week iets aan willen doen, na acht weken ben je dan al klaar. Dan heb je een compleet iets ook voor de toekomst dat heb ik er dan wel voor over. De consultants doen het requirements opstellen nu vooral maar ik zal dat ook zelf wel willen leren. Om te kijken of wij onze technische bevindingen met de functionele wensen van de consultant een goed document kunnen opstellen.

Dan heb ik hier nog een stapeltje met kaartjes. Daar staan ook eigenschappen van taal op en dan vraag ik of jij even wilt kijken of je die wel eens tegenkomt in de requirementsdocumenten. Als je beeldspraak hebt, kom je dat wel eens tegen? Dat wanneer ze dingen gebruiken als "een kapstok gebruiken" of het systeem moet in de database "kijken".

Ja, dit komt wel eens voor ja. Die zie ik wel eens. Kapstok die hoor ik heel vaak, ons systeem bestaat uit allemaal kapstokjes. **beeldspraak.***wel*

Een gezegde, kom je die wel eens tegen.

Nee, die heb ik niet gezien. **gezegde.***niet*

Spreekwoord?

Nee **spreekwoord.***niet*

Afwisseling van structuur van zinnen, dus dat je zinnen op een andere manier opschrijft.

Ja, dat zie ik wel. **afw. structuur.***wel*

En tussenzinnen, een requirement en daartussen weer een requirement.

Ja, dat zie ik ook. Ik ben daar soms zelf ook een beetje een ster in. **tussenzinnen.***wel*

De respondent heeft de volgende kaartjes uitgekozen:
Wel in de requirements: Beeldspraak, Afwisseling structuur zinnen, Tussenzinnen.
Niet in de requirement: Gezegde, spreekwoord.

Wil je deze ook op volgorde leggen welke voor jou belangrijk zijn, welke zou jij graag blijven terug zien in de requirement.

Ok, dan deze beeldspraak vind ik wel makkelijk. Tussenzinnen vind ik niet zo handig en in verschillende structuur dat mag. Tussenzinnen dus niet, die vind ik niet makkelijk. Dan ben je bezig met bouwen en dan moet dat er nog bij. Dan moet je het interpreteren dat vind ik helemaal niet fijn. Sommige tussenzinnen ook dan is het zo algemeen, ja wat bedoel je er dan mee. Daar kan je alles mee, dat zie ik wel als gevaar. **tussenzinnen.** *negatief*

Zie je wel eens dat ze in de requirementsdocumenten vanuit een requirement verwijzen naar een ander requirement.

Ja, dat gebeurd vaak. **verwijzing.***wel*

Dan heb ik nog wat losse eigenschappen voor een taal waarmee je requirements kan maken. Zou je het makkelijk vinden wanneer de taal wordt ondersteunt door een tool?

Ik denk het wel. Wanneer iedereen een beetje conform die tool werkt dan krijg je natuurlijk een beetje dezelfde structuur van requirementsdocumenten en dan kan je daarmee weer heel makkelijk verbeteringen doorvoeren. Dan dat iedereen in zijn eigen Word document zit te werken. **verwijzing.***belangrijk*

Eenvoudig te leren?

Ook wel belangrijk. **eenvoudig.***belangrijk*

Meertalig, dus dat je met een taal gestructureerde Nederlandse zinnen kan maken maar ook Engelse.

Dat vind ik belangrijk. **meertalig.***belangrijk*

Makkelijk handleiding?

Ja dat lijkt me ook wel belangrijk. **handleiding.***belangrijk*

Wil je ze op volgorde leggen.

De respondenten heeft de kaartjes op de volgende volgorde gelegd, van belangrijk tot minder belangrijk: Tool ondersteuning -> Meertalig -> Eenvoudig te leren -> Handleiding.

Dan nog een paar korte vragen, heb je wel eens met een gestructureerde taal gewerkt.

Ja in het ontwikkelen wel maar dat is heel wat anders.

In principe is een programmeertaal ook een gestructureerde taal, daar zitten ook restricties op.

gestructureerd.*weel*

Dat is dan het enige waarmee ik gewerkt heb voor een gestructureerde taal.

Formele taal?

formeel.*niet*

Nee.

Dan bedank ik jou voor het interview.

B.5 Interview 4 Applicatieontwikkelaar

Het doel van dit interview is om te kijken hoe het proces van requirements engineering binnen VitalHealth gaat, kijken waar de knelpunten zitten. Ik heb een paar talen geselecteerd waarmee je requirements kan opstellen, ik wil onderzoeken of deze ook gebruiksvriendelijk zijn. Dit zijn formele talen waarin een restrictie zit op de grammatica en woordenschat. In deze talen is beschreven hoe je zinnen kan opstellen, zodat elke zin dezelfde structuur heeft. Ik doe dit interview via de STARR methode. De S staat voor situatie, om te kijken waar je werkzaam mee bent. Daarna de T van taak, wat je rollen in projecten zijn. Daarna activiteiten, dit is hoe jij omgaat met de requirements. Hierna een stukje reflectie om te kijken hoe je die talen kan inzetten, wat belangrijke eigenschappen daarvoor zijn.

Kun je kort vertellen in welke projecten je nu werkzaam bent?

<project 1>, Hudson, ZBB en <project 3>, daar ben ik ook mee bezig geweest. Dus voornamelijk die eerste drie. — **project.***EHR* **project.***PP*

Wat zijn jou taken in deze projecten?

Dat verschilt heel erg, voornamelijk UI gerelateerd taken. Maar bij ZBB, dat kan ook het beschrijven zijn van requirements. Bij Hudson is het ook, daarna de requirements uitleggen en het laten uitvoeren door andere collega's.

Je bent dus ook een deel bezig met het opstellen van requirements?

Ja. — **rol.***RE*

Heb je ook te maken met andere requirements, van andere collega's?

Ja, bij ZBB voornamelijk en ook wel bij andere projecten. <project 1> ook, <requirements engineer> stelt dan bijvoorbeeld bij <project 1> de requirements op die bespreekt hij met de klant en ik voer zijn uitgewerkte requirements uit. — **rol.***ontwikkelaar*

Hoe bedoel je uitvoeren, die werk je uit in de applicatie?

Ja, dan ben ik uitvoerend. En bij ZBB doe ik het zelf. Dan krijgen we requirements, bespreken de requirements en dan maken we een scope en daarna gaan we er mee bezig.

Hoe krijg jij de requirements aangeleverd, via de opdrachtgever?

Voornamelijk wel.

Heb je dan ook gesprekken met de opdrachtgever over de requirements?

Nee, nou nu niet, ik weet niet of dat gaat veranderen maar eigenlijk niet. Er is een functioneel persoon die dus met de klant praat en dan ook het hele proces opstelt. Van de klant wenst dit kan dit, zo ja prima dan gaan we daar een requirement van maken.

Wat is die persoon, is die consultant?

Nou geen consultant, functioneel coördinator of zo, ik weet niet hoe we dat noemen.

Is dat een intern of een extern persoon?

Intern, bijvoorbeeld bij ZBB is dat <KR>.

En die stelt de requirements op?

Ja.

Hoe krijg jij de requirements van hem aangeleverd?

In een document, een functioneel document. — **aanlevering.***document*

Maken ze daarin gebruik van bepaalde notatietechnieken, zoals bijvoorbeeld use cases of UML diagrammen.

Nee, soms staat er een diagrammetje of een tekeningetje hoe het scherm eruit moet zien. — **notatietechniek.***niet*

Niet echt een standaard techniek?

Nee geen standaard techniek.

De requirements die worden opgesteld, worden die opgesteld voor de applicatieontwikkelaars of voor de opdrachtgevers?

Voornamelijk voor de ontwikkelaars, ik weet ook niet of dat gelijk een leidraad is voor de klant. Door te zeggen dit heb je als requirement het staat nu vast. Geen idee. — **doel rol.***ontwikkelaar*

Maar zie je wel dat de requirements zijn op gesteld voor de ontwikkelaars, zodat ze daarmee een applicatie kunnen bouwen?

Ja, voornamelijk wel ja.

Ik heb een paar requirementsdocumenten onderzocht, wat daarin op viel was dat veel requirements procedureel waren opgesteld in plaats van declaratief. Wat heeft bij jou de voorkeur bij het opstellen van de requirements?

Absoluut geen oplossingsgericht, hoe zij je dat ook al weer... — **procedureel.***afkeur*

Declaratie of procedureel, de eerste is je beschrijft wat het systeem moet doen, dus dat is declaratief en bij de tweede, procedureel, beschrijf je hoe het systeem moet gaan werken. Bij declaratief beschrijf je het probleem wat je gaat oplossen en bij procedureel beschrijf je hoe het systeem moet gaan werken.

Ja niet die tweede, die eerste voornamelijk. Zo wil ik het graag, ik wil weten hoe het systeem zou moeten werken en de oplossing daar mogen we zelf over nadenken daarvoor zijn wij technische ontwikkelaars in principe. Dat wij iets technische gaan maken. Op functioneel niveau kan je dan nog veel wensen hebben omdat je niet hebt nagedacht over de oplossing, je zit dan niet ik je visie te kijken van dit is alleen maar mogelijk. **declaratief.***voorkeur* **declaratief**.*positief*

Je definieert eerst het probleem wat je gaat oplossen?

Ja, er zijn natuurlijk wel afbakeningen enzo.

Denk je dat er ook nadelen aanzitten, aan declaratief werken?

Nadelen, proces kan misschien langer duren doordat je requirements wanneer er vragen zijn dan moet er een laagje extra door als het zo is dat je een klant hebt door requirements die zijn er, er wordt een functioneel design van gemaakt door een functionele man die geeft het over aan een technisch man en die maakt er een technisch design van. Als de technisch design man bijvoorbeeld vragen heeft dan moeten er dus twee laagjes door zeg maar. **declaratief**.*negatief*

Dus je hebt meer communicatie. Voor de rest lijkt het mij alleen maar voordelig want je hebt alles vast staan en gedocumenteerd. En daarbij is het ook nog zo het proces daar wordt beter over nagedacht als je het maar heel vluchtig en oplossingsgericht opschrijft dan gaat die ander er niet heel diep over nadenken en die implementeert het gewoon terwijl het misschien hele andere oplossingen hadden kunnen zijn, die veel beter zijn. **declaratief**.*positief*

Zie jij in de huidige documenten dat er erg procedureel wordt gewerkt?

Ja.

Wanneer je een requirementsdocument hebt, heb je dan wel eens bepaalde woorden die je niet begrijpt, dat je iets tegenkomt en je denkt deze heeft meerdere betekenissen.

Ja, dat is vaak wel zo en dan moet je gokken aan de hand van het voorgaande of het nakomende wat het dan is. Maar vaak is dat nog niet eens het probleem, vaak is het summiere beschrijving het probleem. **lex.ontwikkelaar.***wel* **reden interpretatie.***incompleet*

Van de requirement zelf of van de onduidelijke woorden?

Van de requirement zelf. **interpretatie ontwikkelaar.***wel*

Hoe komt het dan, je zei al summiere beschrijving, maar worden die woorden helemaal niet beschreven?

Nee, zo ver ik weet niet, eigenlijk is het qua woorden niet zo dat ik een voorbeeld kan geven.

Wat mij in de requirementsdocumenten opviel was dat er veel gebruik wordt gemaakt van afkortingen en domein woorden en dat er bij de documenten geen verklarende woordenlijst is opgenomen. Is het standaard om bij de requirementsdocumenten verklarende woordenlijsten op te nemen?

Ja, bij het ene document zie je het wel en bij een ander document zie je het niet, je moet er ook even aandenken. **woordenlijst.***soms*

Wanneer je een requirementsdocument hebt dan worden er requirements opgesteld vanuit een bepaalde domeinkennis van de opdrachtgever. Hoe zorgt de requirements engineer ervoor dat de domeinkennis van de opdrachtgever ook bij jou komt.

Voornamelijk door het document, het functioneel ontwerp. Voor de rest, wanneer het gaat om één nieuwe requirement dan wordt het vaak via een issue of een wens in het ticket systeem gedaan, of via de mail. **domeinkennis.***hoe*

Wat vind jij makkelijker als applicatieontwikkelaar bij een document, wanneer de requirements zijn opgesteld in een lijstje, per requirement een item of een zin of dat ze zijn opgesteld in een beschrijvend stukje tekst.

Dat ligt eigenlijk per requirement. Het kan best zijn dat één requirement drie pagina's beschrijving nodig heeft anders snap je het niet maar een ander requirement is een nieuw veldje toevoegen, dan begrijp je dit in één zin.

Wanneer je meerdere requirements in één stukje tekst hebt?

Dan heb je de requirements niet goed opgesteld. Je kan wel een hoofd-requirement hebben met sub-requirements, als ik me nog goed herinner heb ik dat zo geleerd. **tekstueel.***afkeur*

Wat ik veelal in de requirementsdocumenten heb gezien is zijn dat requirements in stukken tekst worden opgenomen, met een kopje erboven, en daar zitten dan meerdere requirements in verborgen.

Dat is niet altijd even prettig. Het mooiste is een lijstje met een tabelletje met al die requirements met een nummertje ervoor en dan eigenlijk een prio erachter. **lijst.***voorkeur*

Komt het weleens voor dat je een requirement hebt die je op meerdere manieren kan interpreteren? Die naar jou idee meerdere betekenissen heeft.

Ja, dat is wel eens ja. Ik heb eigenlijk hele slechte voorbeelden het is meer eigenlijk technische en dat het al een functionele oplossing is. **interpretatie ontwikkelaar.***wel*

Maar je denkt dat het wel voorkomt?

Het komt zeker voor ja.

Denk je dat het ook invloed heeft gehad op de ontwikkeling van de softwareapplicaties?

Ja absoluut, het geeft verwarring en frustratie. **invloed ambiguïteit.***wel*

Heb jij misschien een voorbeeld uit de praktijk?

Ja het is misschien wel een hele flauwe maar in het functioneel design van <K> stond we willen graag medicatie hebben, dat was in het ZBB project, we willen medicatie hebben net zoals in het project CHM. Maar ja, dat was de enige functionele omschrijving voor het medicatie stuk. Daar kan je van alles mee doen natuurlijk, niemand die controleert of het werkelijk zo werkt als in het CHM. **invloed ambiguïteit.***wel*

En die requirement had hij opgesteld voor applicatieontwikkelaars?

Ja.

Had hij deze ook met de opdrachtgever besproken, op dezelfde manier?

Ja.

Dan wist de klant blijkbaar wat CHM was?

Waarschijnlijk wel, hij heeft waarschijnlijk een schermpje laten zien ofzo, een keer een demotje gegeven van CHM hoe dat werkt. De klant die ziet dat eigenlijk een functioneel persoon die stuurt eigenlijk erop wat de klant gaat kiezen, in dat opzicht heeft hij het niet verkeerd gedaan.

Wanneer je een requirement tegenkomt die onduidelijk is, wat doe je dan?

Ja eerst vragen stellen, wat er nou precies mee moet.

Aan de persoon die het document heeft opgesteld?

In dit geval ben ik bezig gegaan heb ik vragen gesteld, ook bij het CHM groepje wie daar kennis van heeft en toen ook letterlijk alles gekopieerd *(vanuit het CHM model naar het nieuwe model)* en tijdens de demo kwamen er veel vragen naar boven, wat dus de eigenlijke requirements waren, dat kon dus niet omdat het zo uit CHM was geplakt. Voortaan eerst helemaal uitvragen en dan pas implementeren. **ambiguïteit voorkomen.***hoe*

Dat is jou oplossing om te voorkomen dat je dubbelzinnige requirements gaat implementeren?

Ja, zo doe ik dat.

Op welke manier controleer jij dat al de requirements zijn opgenomen in de softwareapplicatie, heb je daar een bepaalde methode of techniek voor?

Nee

Ik heb hier een aantal kaartjes waarop eigenschappen van kwaliteit van requirements opstaan. Wil jij per kaartje bekijken of deze voor komt in de huidige requirementsdocumenten. Zijn de requirements consistent?

Dat zo en zo niet. **consistent.***niet*

Eenduidig, zijn de requirements voor meerdere uitleg vatbaar, waar wij het net over hadden.

Dat is niet altijd zo, in mijn geval zijn ze niet altijd heel duidelijk beschreven, ik zal deze op nee zetten. **eenduidig.***niet*

Zijn de requirements implementatie onafhankelijk? Vrij van ontwerp en implementatie beslissingen, in de requirements is nog niet nagedacht over hoe je het gaat implementeren.

Nou dat is dus wel gedaan, dan leg ik deze ook onder nee. **implementatie.***niet*

Traceerbaar, de relatie tussen de onderlinge requirements zijn goed helder en zichtbaar. Wanneer je de ene requirements wijzigt, dan weet je ook wat de gevolgen zijn voor de andere requirements.

Ja, dat zou ik zeggen van wel, inderdaad maar dat staat niet in de requirements beschreven. Er is wel een onderlinge relatie tussen de requirements, ik denk dat deze wel in de requirementsdocumenten zit. **traceerbaar.***wel*

Verifieerbaar, de applicatie kan op basis van de requirements worden getest of de applicatie aan de wensen van de opdrachtgever voldoet?

Dat gebeurt vaak wel, testers die doen dat op die manier. **verifieerbaar.***wel*

Die testen echt op basis van de requirements?

Ja op basis van het document.

De respondent heeft de volgende kaartjes uitgekozen:

Wel in de requirements:	Traceerbaar, Verifieerbaar.
Niet in de requirement:	Consistent, Ontwerp onafhankelijk, Eenduidig.

Welke eigenschappen vind jij belangrijk voor de requirements? Wil je die op volgorde leggen, welke je echt belangrijk vind en welke minder.

Die traceerbaar is voornamelijk ook de kennis van de persoon, wanneer dat niet in het document staat. Degenen die het dan gaat uitwerken die moet dat dan gelijk kunnen zien.

De respondenten heeft de kaartjes op de volgende volgorde gelegd, van belangrijk tot minder belangrijk: Ontwerp onafhankelijk -> Consistent -> Verifieerbaar -> Eenduidig -> Traceerbaar.

Ik ben bezig met een onderzoek om te kijken of de requirements kunnen worden opgesteld met een gecontroleerde taal. Dit is een taal waarop restricties zitten op bijvoorbeeld de grammatica en woordenschat. Ik onderzoek hoe deze talen kunnen worden geïmplementeerd voor de requirements. Wanneer je de eigenschappen die je net hebt geselecteerd, wil jij dan tijd investeren om de taal te leren, zodat in de requirements de eigenschappen zitten die je net hebt uitgekozen. Dus als je zo'n taal gebruikt dat je requirements de eigenschappen krijgen die jij graag wilt?

Ik denk het wel ja, ik zou zeggen leer het iedereen dan kunnen we tenminste de documentatie op orde krijgen. Wanneer dat een heel groot voordeel heeft voor de documentatie. — **tijd.***wel*

Hoe veel tijd zou jij ongeveer willen investeren om een gecontroleerde taal te leren? Een werkdag, twee werkdagen een week of langer..

Ja, mij maakt het niet zo veel uit hoeveel tijd ik eraan besteed. Het is maar net hoeveel tijd je ervoor vrij krijgt. Als je graag wilt dat je goede documenten maakt dan zal ik in dat geval gewoon veel tijd er aan besteden net zo lang tot ik het goed kan. Is dat één dag is dat één week, ik denk dat je met een week training wel aardig op weg bent.

Jij maakt ook requirements, dan heb ik hier een ander stapeltje met kaartjes. Een paar taal eigenschappen, om te kijken waar jij gebruik van maakt in de requirements. Maak jij gebruik van tussenzinnen in een requirement?

Ja, volgens mij maak ik hier wel gebruik van. — **tussenzinnen.***wel*

En zie je dit ook vaak terug komen in andere documenten?

Volgens mij wel.

Beeldspraak?

Ik denk dat die er ook wel in zit ja. — **beeldspraak.***wel*

Gezegde?

Ik denk het niet. Misschien dat ik wel gezegde gebruik zonder het te weten. — **gezegde.***niet*

Spreekwoord?

Ik probeer voornamelijk objectief te zijn, gebruik ik niet. — **spreekwoord.***niet*

Afwisseling in structuur van zinnen?

Waarschijnlijk wel.

De respondent heeft de volgende kaartjes uitgekozen:
Wel in de requirements: Afwisseling structuur zinnen, Tussenzinnen, Beeldspraak.
Niet in de requirement: Gezegde, spreekwoord.

Welke eigenschappen vind jij belangrijk om te gebruiken?

Om te gebruiken of te vermijden, ik kan me voorstellen dat je tussenzinnen niet graag wil omdat die niet heel helder is. Ik weet niet of ik afwisseling in structuur van zinnen zou willen gebruiken, waarom zou je dit willen doen, het moet consistent zijn. — **tussenzinnen.** *negatief*

Dat leest makkelijker dan wanneer je bij iedere zin dezelfde structuur hebt.

Beeldspraak vind ik een beetje lastig, ik zit een beetje te twijfelen met die taal barrière, het verschil tussen een Indiër en een Nederlander bijvoorbeeld. Dat je allemaal verschillende beeldspraken heb. Dan zou je zeggen in dat geval die wel, en de rest niet. — **beeldspraak.** *negatief*

De respondent heeft de volgende kaartjes uitgekozen:
Wel in de requirements: Afwisseling structuur zinnen.
Niet in de requirement: Gezegde, spreekwoord, Tussenzinnen, Beeldspraak.

Maak jij gebruik van verwijzingen in de requirements, dat je vanuit een requirement echt verwijst naar een andere requirement.

Dat heb ik niet gedaan. — **verwijzing.***niet*

Zie je dat weleens voorkomen?

Dat zie ik weleen voorkomen ja. — **verwijzing.***belangrijk*

Denk je dat dit essentieel is voor het maken van de requirements?

Het kan soms handig zijn, wanneer het leunt op een ander requirement.

Ik heb hier nog een laatste stapeltje, dit zijn kaartjes voor algemene eigenschappen van een taal, een gecontroleerde taal. Zou jij het belangrijk vinden wanneer deze wordt ondersteunt door een tool?

Ja dat zou handig zijn. Maar dat is dan een soort van word spelling controller op een requirement. Het is niet nodig maar het is wel handig. Ik zou in eerste instantie zeggen dat is niet heel erg nodig. — **tool.***belangrijk*

Meertalig, dat je zowel Nederlandse als Engelse requirements kan definiëren met de gecontroleerde taal.

Ik zou eerst beginnen met het Engels, dus dan is het nee. — **meertalig.***nee*

Eenvoudig te leren?

Dat wel. — **eenvoudig.***wel*

Een handleiding beschikbaar?

Dat lijkt mij wel handig ja. Een handleiding is altijd makkelijk wanneer je uitzonderingen hebt of zo. — **handleiding.***wel*

De respondenten heeft de kaartjes op de volgende volgorde gelegd, van belangrijk tot minder belangrijk: Eenvoudig te leren -> Handleiding -> Tool ondersteuning -> Meertalig.

Heb jij wel eens met een gecontroleerde of gestructureerde taal gewerkt?

Nee.

gestructuureerd.niet

Een formele taal?

Nee.

formeel.*niet*

Dan bedank ik jou voor het interview.

B.6 Interview 5 Applicatietester

Het doel van dit interview is om te onderzoeken hoe jij als applicatietester met de requirements omgaat die opgesteld worden door requirements engineers, zoals bijvoorbeeld consultants. Daarnaast heb ik wat eigenschappen van requirements gedefinieerd om te kijken welke eigenschappen jij belangrijk vindt voor requirements. In mijn onderzoek ben ik bezig met een gecontroleerde taal. Dat is een speciale taal waarmee je de requirements kan opstellen om de requirements nauwkeuriger te maken. In dit interview wil ik onderzoeken of dit in jou rol als tester ook voordelen heeft.

Ja, nee dat sowieso want wij komen aan het einde van de ontwikkel straat dus wanneer het aan de voorkant onduidelijkheden schept dan schept dat aan de achterkant ook onduidelijkheden en dan heb je dus vlakke waarop je problemen kan constateren.

In welke projecten ben jij werkzaam, waar ben je bij betrokken?

QuestManager daar ben ik een key-tester. <applicatietester> die is voor KIS. En daarnaast zijn wij beide betrokken bij Hudson, maar daar trek ik meer de kar dus het coördinerende stuk. Verder bij MS-subportaal, dat is bij het VUMC. Daar ben ik voor werkzaam.

project.*QM*
project.*EHR*
project.*MP*
rol.*tester*

Is VUMC een patientenportaal?

Nee het is meer een customproject meer maatwerk. En bij het patiëntenportaal heb ik een rol gespeeld. Maar dat staat op een lager pitje. Dat zijn wel de hoofdprojecten. En dan nog het supportsysteem daar hebben we een fase gehad waar ik best wel bij betrokken ben als tester. Met daarbij de coördinatie naar India toe.

De coördinatie doe jij ook?

Ja, dat doe ik in overleg met iemand daar. Een collega in India stuurt de mensen daaraan en we hebben wekelijks een call van wat zit er aan te komen voor jullie. Vooral vanuit hier de ondersteuning daarin naar hun toe.

Hoe test jij dan, test jij op basis van requirements?

Voor QuestManager werken we nu via de sprints en daar hebben we eindelijk dat het echt allemaal netjes via SCRUM gaat. Je hebt nu echt dat de requirements die komen bij consultantie vandaan, die vertaald deze al direct naar een design dus ik heb daar niet daadwerkelijk met requirements te maken ik heb daar al met een design te maken, dus nog een stapje verder. De requirements die bedenkt hij zelf en die zet hij zelf om naar het design. Bij hudson is het wel zo dat het requirements zijn. Het is binnen VitalHealth vaak een combi tussen functioneel ontwerp en requirements. Het is niet echt dat er een scheiding gemaakt wordt tussen beide. Bij hudson is het een combinatie van, de ene keer krijg ik een stuk functionaliteit om te testen waarbij je alleen de geformuleerde requirements hebt dan heb je vaak een stuk kennis overdracht nodig want vaak is het één twee zinnen en dat is alles. Of meer wel een uitgewerkt stuk en dat is dan meer een uitgewerkt conceptueel ontwerp.

functioneel.*nee*

Voor de requirements heb je wel echt domeinkennis nodig?

Ja klopt, omdat je key-tester bent ken je het domein behoorlijk alleen dan net dat stukje, in één twee zinnen kun je niet de lading overbrengen wat er overgedragen is aan de developer. Want binnen hudson werken we vaak dan in een issue in het support systeem voor de ontwikkelaars, daar staat wat in. En van daaruit kun je separate test issues generen en die komt dan weer bij mij terecht en dat is gewoon een kopietje daarvan maar toch omdat stuk te scheiden.

reden interpretatie. *domeinkennis*
reden interpretatie. *incompleet*
aanlevering.*issue*

En de issue is al een requirement?

Ja, dan is het al als wens gelogd als een activiteit naar de ontwikkelaar en dat wordt dan één op één overgezet naar de tester toe.

Wanneer jij domeinkennis mist, hoe krijg je die dan, hoe verzamel je die.

Je hebt van die kennis sessie, Als je gaat testen dan hebben we vooraf af en toe is een sessie. Voor QuestManager hebben we dat echt omdat het volgens de methodiek SCRUM heb ik vooraf een lijst van dit zijn de requirements en dit moet er getest worden. En dan ga ik echt vooraf zitten met een applicatieontwikkelaar en dan gaan we ze één voor één langs. En dan bespreken we wat de inhoud is. En bij Hudson is het vaak, dan zijn het er zoveel als je toe bent aan een stuk, want daar hebben we vaak een scheiding in de test cycle tussen bestaande en nieuwe. De nieuwe zijn dan de requirements, en dan ga je echt één voor één, want het zijn echt verschillende ontwikkelaars en dan weet je wie hem opgepakt heeft gelukkig en dan ga je gewoon even zitten.

domeinkennis.*hoe*

En bij QuestManager wanneer je die requirements hebt, zijn die opgesteld door de applicatieontwikkelaar?

Die komen vanuit de sprint backlog, dat wordt bepaalt aan de hand van wat komt er binnen aan wensen van de klant plus de roadmap functionaliteit van de productowner, van de consultant. Dus daar komt het dan vandaan.

En de consultant die kijkt dan wat er in komt, of doen jullie dat samen.

Dat gaat vooraf aan een SCRUM sprint, daar hebben we één overleg en daarin wordt bepaalt dit gaan we in deze sprint doen. Maar daar zit ik niet bij want dat hoe ik niet, ik hoef daarin geen bepalende rol te hebben. Maar daar zitten ook mijn issues in vanuit een vorige sprint waarbij ik zeg dit is wel zo mooi ontwikkeld bij op deze manier maar het is niet gebruiksvriendelijk. Pas had ik een stuk, dat was ontwikkeld maar er ontbrak gewoon een heel stuk gerelateerd daaraan. En dat wordt dan in die sprint backlog gestopt en in een planningsmeeting vooraf wordt vastgesteld wat we gaan oppakken.

Jij krijgt dus de requirements aangeleverd via het supportsysteem...

Ja of via mail waarin het lijstje staat. Bij QuestManager krijg ik het via de mail of ze hebben een Excel sheet waarin alles staat.

aanlevering.*support systeem*
aanlevering*.mail*

Dus het is verschillend hoe je het krijgt aangeleverd, het kan mail zijn

Ja bij QuestManager is het een Excel sheet maar het is per project verschillend. Maar daar zit wel vaak een issue aangekoppeld uit het supporsysteem. Ik weet wel uit het KIS project en daarin is een tester bezig en die heeft gewoon een mailtje gekregen van een applicatieontwikkelaar van dit zit er in en dan verwijst hij veelal naar issues uit het support systeem.

aanlevering*. spreadsheet*

Bij de requirements die worden opgesteld wordt daar gebruik gemaakt van een notatietechniek?

Nee, dat heb ik alleen bij MS-subportaal gezien maar die waren verwerkt in het design. Dat is echt een dik papier en die hadden ze gelijk verwerkt en helemaal netjes uitgewerkt. Maar dan zie je nog van dit is dan een use case maar hoe de vertaling is naar de software dat is niet altijd één op één, nog veel grijze gebieden.

notatietechniek.*niet*

Is het voor het testen makkelijker wanner je het in een use case hebt?

Dat is afhankelijk van de complexiteit van het stuk wat er getest moet worden. Daarom is het belangrijk, met een use case alleen en een stuk requirements ben je er vaak niet. Om het verband te kunnen begrijpen en de achtergrond van de requirements moet je gaan zitten.

Voor het testen heb jij echt domeinkennis nodig?

Ja.

Bij het opstellen van de requirements kan de requirements engineer de requirements opstellen voor een opdrachtgever zodat deze ze goed begrijpt of voor een applicatieontwikkelaar. Zie je dat verschil terug in de requirements dat je denkt die zijn echt opgesteld voor een bepaalde groep?

Nee, het zou niet moeten. Ik zie nooit van dit is echt zo technische omschreven dat is echt voor een ontwikkelaar. Ik ben een functionele tester en de consultants zijn meestal ook functionele mensen dus omschrijvingen die je terug ziet is wel functioneel en niet technisch. En de opdrachtgever heeft meestal ook wanneer die iets in Jip en Janneke taal zeg maar kan begrijpen dan is het niet technisch.

doel rol.*beide*

Kom je weleens requirements tegen waar bepaalde woorden inzitten die je niet begrijpt.

Nee niet echt, dan zou het echt specifiek voor een nieuw stuk functionaliteit zijn dat voor een nieuw domein is. Heel af en toe heb je binnen QuestManager bepaalde terminologie binnen de GGZ die je niet kent, dat is toch een ander soort wereldje, bij zulk soort woorden dan ga je even google en dan weet je het ook wel weer hoe het zit.

lex tester. *soms*

En die woorden, worden die opgenomen in een verklarende woordenlijst?

Nee.

woordenlijst.*niet standaard*

Zie je dat weleens terug in de requirementsdocumenten?

Nooit nooit.

Valt het je weleens op dat sommige requirements dubbel te interpreteren zijn?

Ja, ja dat ik de manier van hoe zet je ze neer. Maar dat is sowieso bij tekst vaak zo de manier waarop je iets formuleert. Want een mail die je van persoon X naar persoon Y stuurt de ontvanger kan het op een hele andere manier interpreteren als het bedoeld is. En daarom zullen die notatie technieken dat zal wel goed zijn. Je komt dat wel eens tegen maar ja.

interpretatie tester.*wel*

Wat doe je dan als je dit tegenkomt?

Owh ja gelukkig kan ik ze of via skype benaderen of mondeling even toelichten.

En hoe vaak kom je dit ongeveer tegen? Kan je hier een indicatie voor geven.

Dat is misschien 5 procent van alles wat ik langs zie komen. Niet heel veel.

interpretatie tester.*wel*

Hoe komt het dan dat je het niet begrijpt, is het omdat het een dubbele betekenis heeft in het domein of dat het taalkundig dubbelzinnig is.

Taalkundig. dat het niet netjes is.

reden interpretatie. *zinsstructuur*

Dus de structuur van zinnen is slecht?

Nou ja, dat kom je wel vaker tegen. Dat zie ik ook wel bij de documentatie die door verschillende collega's opgesteld worden. Dan denk je ga maar even een cursus Nederlands volgen want dit is afschuwelijk.

Requirements kan je declaratief opstellen, dan beschrijf je wat het systeem moet doen en je kan ze procedureel opstellen dus dan beschrijf je hoe het systeem moet gaan werken. Wat heeft bij jou de voorkeur als tester?

Hoe het moet gaan werken, dus echt procedureel. Dat heeft ook met mijn achtergrond te maken. Ik ben hiervoor ontwerper geweest en dan denk je vaak, in mijn opleiding ook procesmatig. Dus dan is die stap snel gemaakt. **procedureel.** *voorkeur*

Kom je dit ook meestal tegen in de requirements of kom je ook die andere variant tegen?

Ja, dat is niet helemaal waar. Dat kan ik moeilijk zeggen die slag die maak ik zelf in mijn hoofd. Zo moet het procesmatig werken. Daar let ik eigenlijk niet zo heel vaak op omdat het automatisme is. Wanneer je tussen de regels door, ik vind eigenlijk dat het een combi moet zijn tussen een soort opsomming van dit zijn de requirements plus een klein stukje toelichting. Dat zou mijn voorkeur hebben.

Ik heb een paar documenten vergeleken, een paar requirementsdocumenten van VitalHealth wat ik daar heel vaak in tegen kwam is dat er een stuk tekst wordt opgenomen in het document en in die tekst zitten dan al de requirements.

Ja, dat kan je doen als je de use case gaat uitleggen. Daarom die combinatie van dat je ze eerst opsomt en vervolgens de uitleg erbij ja dan zie je van dit is daar verwerkt en dit is daar verwerkt zo in je stukje uitleg zie je dat meteen terug. Dit heeft zeker mijn voorkeur want dan kan je het ook minder snel op een andere manier interpreteren. **lijst.***voorkeur*

En voor het testen, is het daarvoor ook makkelijker?

Ja natuurlijk, dan hoef je ook minder vaak bij iemand langs te gaan. Dat scheelt weer tijd als je het in één keer goed doet.

Zie je weleens dat bepaalde requirements ambigu zijn, dus de requirements kunnen op meerdere manieren worden opgevat dat dit invloed heeft op de kwaliteit softwareapplicaties?

Ja, dat is.... eigelijk zou je dat al aan de voorkant moeten zien wanneer een ontwikkelaar daar mee aan de gang gaat en die ziet van ik kan dit op die manier interpreteren of op die manier of weer op een andere manier. Wanneer hij dat niet ziet dan loop ik er wel tegenaan dus eigenlijk een dubbel check. Maar eigenlijk zou ik daar dan niet meer tegenaan moeten lopen. **invloed ambiguiteit.***wel*

Wat wanneer je daar tegenaan loopt dan is de applicatieontwikkelaar daar meestal nog niet tegen aan gelopen?

Nee, die heeft het op een bepaalde manier geïnterpreteerd en die gaat ermee aan de slag. Die heeft het dan zo ingebouwd en dan kom ik het tegen als tester. En dan is het even schakelen met de product owner. Maar ja het komt ook weleens voor dat de ontwikkelaar met dezelfde punten zat maar daarna is niet het requirements aangepast. Dat blijft zoals het afgesproken is zeker vanuit de klant. **aanpassen.***niet*

Wanneer jij iets tegenkomt, wordt het dan wel aangepast?

Nee ook niet. Het requirement is zoals het is. Het requirement blijft staan.

En wat doen ze dan om te voorkomen wanneer het nog een keer voorkomt?

Nou vaak, voor QuestManager is het ook zo dat ik de release notes opstel en die komen vaak uit de requirements. Gewoon kort en bondig beschrijven wat nieuw is en dan maak ik mijn verhaal dusdanig met een paar zinnen dat de klant het snapt. De klant die test vaak op basis van de requirements of op basis van de release notes wanneer ze die meekrijgen. Een breedte test door de hele applicatie heen maar ook specifiek op de nieuwe punten.

Dus de requirements worden ook nog een keer door de klant zelf getest?

Nou ja in de vorm van de release notes dan. Want die zijn vaak één op één aan elkaar gekoppeld. Dus ja dan maak ik mijn verhaal dusdanig dat deze maar op één manier interpreteerbaar is.

Zij jij dat er iets gedaan wordt door de ontwikkelaar om te voorkomen dat de requirements op meerdere manieren worden geïnterpreteerd?

Nee, daar heb ik totaal geen zicht op.

En door de consultant?

Nee, ik denk het niet dat het gebeurt want ik zie nooit dat het dan als ik met zo'n punt kom dat daarna het requirement aangepast is. Dus het wordt niet gedaan. **ambiguïteit voorkomen.***niet*

Er wordt ook geen check gedaan om het te voorkomen dat het bij jou komt?

Ja dat zou mooi zijn, zo zou het wel moeten zijn. De requirement lijst zou in feite door iemand gereviewd moeten worden en het ontwerp ook. Maar het begint aan de voorkant bij het opstellen van de requirements. Dat is wel een punt waar het mee begint. Wanneer je die laat reviewen door een ander met een objectieve blik erop dan zou je die dingen wel eruit kunnen halen. Of het samen doorspreken van de requirements door de ontwikkelaar en de opsteller. Ik niet hoe het bij KIS bijvoorbeeld gaat, vaak is het voor hun ook kort schakelen met de opsteller maar of het daarna aangepast wordt dat weet ik niet. **ambiguïteit voorkomen.***hoe*

Ik hier een aantal kaartjes en daar staan kwaliteitseigenschappen voor requirements op. Het is aan jou de vraag om te kijken of deze voor komen in de huidige requirements. Dus zijn de requirements verifieerbaar?

Ja ze zijn verifieerbaar.

Zijn de requirements door jou goed te testen?

Wanneer de requirements verbeterd door wat ik net zei meer opsomming plus een stukje uitleg in plaats van twee drie zinnen en dat is alles en zoek het zelf maar uit. Dat zie je bij ontwikkelaars ook hoor. Die worstelen dan eigenlijk met een grijs gebied die ze zelf mogen invullen. Dan wordt er wel binnen onze organisatie gezegd van je moet ze niet alles voorkauwen, en dan vooral de Indiërs. Maar goed het moet wel helder zijn. En gelukkig hebben we korte lijntjes. Verifieerbaar is het wel degelijk. Het scheelt tijd wanneer het vooraf al helder is.

Eenduidig, daar hebben wij het al een beetje overgehad.

Dat is dus niet altijd zo. Maar vaak wel. **eenduidig.***niet*

Consistent? Dat is dat de requirements niet met elkaar conflicteren.

Mijn ervaring is dat ik daar niet zo vaak tegenaan loop. Omdat het vaak om wijzigingen gaat en nieuwe stukken dus dat kan niet met elkaar conflicteren. Wel af en toe zie je dat een wijziging conflicteert met een requirement die al veel eerder of door een andere klant. Die wilde wat anders ten opzichte van wat een andere klant opeens wil. **consistent.***niet*

Maar loop jij daar als tester ook tegenaan of juist de applicatieontwikkelaar?

Ja die de applicatieontwikkelaar. Maar ja als je wat langer mee loopt in een bepaald domein dan weet je het ook.

Traceerbaar. De relatie tussen de verschillende requirements is duidelijk.

Jawel er is niet altijd samenhang omdat het soms dit is een nieuw stuk en dit is weer een nieuw stuk dan is er totaal geen samenhang. **traceerbaar.***wel*

Maar wanneer er wel samenhang is dan zie je dit terug?

Ja dan staan ze onder elkaar. Dan zie je dat gebied wel.

Implementatie onafhankelijk, in de requirements worden nog geen implementatie keuzes gemaakt.

Ja geregeld dat je tegen een oud document met requirements zit te testen. Dat kom je ook vaak tegen bij VitalHealth. Ohw ze zijn inmiddels weer veranderd, binnen hudson speelt dat enorm. Ik hoop dat dat nu inmiddels wel aan het veranderen is. **implementatie.***niet*

Zie jij dat er in een requirement al implementatie beslissingen zijn genomen. Dat er al beschreven wordt hoe de applicatieontwikkelaar het moet gaan maken.

Ja dan is hij daar halverwege mee. En dan in één keer wordt het een hele andere richting opgestuurd. Dat allemaal onnodig werk. Nee dat is wel zeker van toepassing.

Wil je de kaartjes op volgorde leggen welke voor jou belangrijk zijn.

ja is goed.

Deze vind jij het belangrijkste, verifieerbaar.

Ja want wanneer ik niet kan testen dan houdt het op.

De respondent heeft de kaartjes op de volgende manier geordend:
verifieerbaar, (consistent, eenduidig en ontwerp onafhankelijk) drie naast elkaar, Traceerbaar.

Ik ben bezig geweest met een onderzoek naar gecontroleerde talen. Met deze talen zijn bepaalde structuur in zinnen verplicht. Daarmee kan je dan sommige van deze kwaliteitseigenschappen oplossen, een deel ervan. Zou jij er tijd in willen investeren omdat te leren?

Nou nee, dat vind ik meer dat dat aan de voorkant thuis hoort. Ik stel nooit zelf de requirements op. Dat is het verschil en als je dat wel doet en ook als je ontwerper bent en dat hoort denk ik echt thuis bij consultantie. Het opstellen van de requirements. **tijd.***niet*

En het lezen?

Nee, volgens mij moet het dusdanig duidelijk zijn zodat het bij iedereen goed leesbaar wordt. Je kan je daar niet specifiek in training voor nodig hebt.

Ik heb hier een stapeltje met een aantal andere kaartjes. Dit zijn meer taaleigenschappen van de requirements. Wil jij kijken of deze in de requirements voorkomen die jij gebruikt. Zie jij weleens een requirement waarin beeldspraak wordt gebruikt.

Nee gelukkig niet. Dan kan je het weer op meerdere manieren opvatten. Dan moet je eerst weten of je begrijpt wat de beeldspraak betekent. Dat is wel vaak zo maar als je dat niet weet moet je het eerst gaan opzoeken. Maar dat zal de een slechte requirement zijn. Gelukkig zie ik dit niet. **beeldspraak.***niet*

Tussenzinnen?

Ja. **tussenzinnen.***wel*

Afwisseling in structuur van zinnen. Dat is dat je een zelfde requirement op een andere manier kan definiëren.

Nou gelukkig is het meestal wel zo dat dezelfde persoon ze opstelt. Want als je door twee personen requirements laat opstellen dan zie je twee verschillende manieren van notatie. Dus daarin verwacht ik dat dit terug komt maar nee. **afw. structuur zinnen.***niet*

Gezegde?

Nee dat is dezelfde categorie een beetje als de beeldspraak. Nee dat zie ik gelukkig niet. **gezegde.***niet*

Spreekwoord?

Nee ook niet. Wat je hoog uit tegenkomt is dit, tussenzinnen. En zeker als het om uitleggen gaat van een stuk dan zou dit wel kunnen voorkomen. Dit daarna al heel af en toe. **spreekwoord.***niet*

En welke vind jij belangrijk die voor mogen komen?

Nou dit vind ik meestal geen probleem.

De respondent heeft de kaartjes op de volgende manier geordend: afwisseling in structuur van zinnen wel belangrijk.
Tussenzinnen niet van toepassing, komt de respondent nooit tegen.
Niet belangrijk: beeldspraak, gezegde en spreekwoord.

Spreekwoorden ben ik nog nooit tegengekomen in de requirements. Het zou kunnen maar dan krijg je wel erg wollig taalgebruik.

Kom je dat weleens tegen, wollig taalgebruik?

Ja, daar zijn bepaalde personen heel sterk in. Het is ook puur wie schrijft de requirements. Daarom vind ik het goed wanneer je zowel een stuk opsomming en een stuk korte uitleg doet. Want als je de opsomming hebt en je begrijpt wat er staat als je dat al gelijk weet dan kan je de uitleg al overslaan.

Zie je weleens dat er verwijzingen worden gemaakt van het ene requirement naar een ander requirement?

Ja dan wordt er terug verwezen. Niet naar een andere maar dat is bijna nooit zo. Want meestal staan de requirements op zich. Of ze moeten een relatie met elkaar hebben dan wordt er weleens verwezen. **verwijzing.***wel*

Ik heb hier nog een aantal kaartjes waar een aantal algemene eigenschappen van requirements op staan. Vind je het belangrijk dat een requirement meertalig is. Dat je er zowel Nederlandse als Engelse zinnen daarmee moet kunnen opstellen?

Nee, meestal is het zo dat je de requirements in samenwerking met de klant opstelt en als je weet of het een Nederlands of Engels talige klant is dan weet je al welke taal je requirements moeten. In het vervolg het design als je die in het Engels is om dat je niet weet of die in Nederland of in India wordt ontwikkeld dat dat één taal wordt in de toekomst dat zal wel goed zijn. Maar als je ook nog eens goedkeuring van een Nederlandse klant moet hebben dan is het logische dat je het in het Nederlands doet. **meertalig.***belangrijk*

Tool ondersteuning, zodat je door een tool wordt geholpen om de requirements op te stellen?

Wanneer er trainingen voor zijn dan lijkt mij het dat de trainingen voldoende moeten zijn. **tool.***onbelangrijk*

Handleiding beschikbaar?

Wanneer je een training hebt gedaan dan lijkt mij dat wel een naslagwerk. **handleiding.***wel*

Eenvoudig te leren?

Ja ook belangrijk, ik moet er geen weken tijd in moeten besteden om het te leren. **eenvoudig.***wel*

Wil je ze op volgorde leggen van welke voor jou belangrijk zijn?

Ja..... als je deze, meertalig, ook in India en in Amerika wilt doorvoeren dan wordt het een heel ander verhaald.

De respondent heeft de kaartjes op de volgende manier geordend:
Eenvoudig te leren, (Handleiding en tool ondersteuning), meertalig.

Nog wat laatste vragen, heb jij weleens met een gestructureerde taal gewerkt?

Nee. **gestructureerd.***niet*

Formele taal?

Nee. **formeel.***niet*

Dan bedank ik jou voor het interview.

C Enquête

C.1 *Vragenlijst requirements engineer en applicatieontwikkelaar*

Dankjewel dat je mee wilt werken aan dit onderzoek door middel van het invullen van deze enquête. Het doel van dit onderzoek is om te onderzoeken of bepaalde requirements technieken de software requirements binnen VitalHealth kunnen verbeteren.

In de software requirements zijn wensen en eisen beschreven voor een softwareapplicatie. Anders gezegd, in software requirements worden de eigenschappen gedefinieerd die een softwareapplicatie moet hebben om succesvol te zijn. Onderzoek heeft aangetoond dat er een relatie bestaat tussen de kwaliteit van software requirements en het slagen van een softwareproject. Met de enquête probeer ik te achterhalen hoe bepaalde zaken tijdens de requirements engineering verlopen, het doel hiervan is om te onderzoeken welke factoren van invloed zijn op de kwaliteit van de requirements. Daarnaast wil ik jullie persoonlijke mening vragen over een aantal technieken die software requirements kunnen verbeteren.

De enquête bestaat uit twee delen. In het eerste deel van de enquête wordt je gevraagd naar jouw ervaringen met het opstellen van requirements en het bouwen van een softwareapplicatie in projecten waarin jij werkzaam bent geweest. In het tweede deel van de enquête wordt er naar jouw persoonlijke mening gevraagd over bepaalde eigenschappen van requirements en wordt er gevraagd hoe jij denkt over bepaalde technieken waarmee requirements kunnen worden opgesteld.

De enquête bestaat voornamelijk uit meerkeuzevragen, met daarnaast een aantal open vragen. Het kost ongeveer vijftien tot twintig minuten tijd om deze enquête in te vullen.

Verklarende woordenlijst

In de enquête wordt gebruik gemaakt van een aantal termen, hieronder staat de betekenis daarvan:

Software requirements	De wensen en eisen voor de softwareapplicatie. Binnen VitalHealth worden deze vastgelegd in; design documenten, high level design, functional design en andere documenten waarin beschreven staat wat of hoe het systeem moet doen / werken. Requirements kunnen ook zijn vastgelegd als wens in het VitalHealth Support systeem.
Opdrachtgever	Onder deze rol vallen personen die belang hebben bij de applicatie of kennis hebben van het domein die als input voor de requirements heeft gediend. Dit kunnen o.a. zijn: opdrachtgever, partners.
Requirements engineer	Onder deze rol vallen personen die software requirements hebben opgesteld voor een applicatie. Dit kunnen o.a. zijn: consultants, software architecten etc.
Applicatieontwikkelaar	Onder deze rol vallen personen die de applicatie hebben ontwikkeld. Dit kunnen o.a. zijn: applicatieontwikkelaars, modeleurs, toolsontwikkelaars.

* **Vereist**

1. Bij welke VitalHealth projecten ben je het afgelopen jaar betrokken geweest?*
Geef de projecten op waarbij je het meest betrokken was, kies er maximaal drie.

- ☐ <project 1>
- ☐ CHM-KISS
- ☐ Hudson
- ☐ <project 3>
- ☐ Anders:
- ☐ QuestManager
- ☐ Abott
- ☐ CWZ
- ☐ Isala

2. Wat was je rol in deze projecten?*
Het kan voorkomen dat je meerdere rollen hebt vervuld in de projecten, geef maximaal twee rollen op.

- ☐ Opdrachtgever of belanghebbende.
- ☐ Requirements engineer.
- ☐ Applicatieontwikkelaar
- ☐ Applicatietester
- ☐ Anders:_____

3.* Hoe vaak heb je in deze projecten te maken gehad met software requirements?

☐ Nooit ☐ Soms ☐ Vaak ☐ heel vaak

4. Heb je in deze projecten software requirements opgesteld?*

☐ Ja ☐ Nee

INDIEN JA -> Ga verder naar de volgende pagina
INDIEN NEE -> Ga verder naar pagina 12

5% ingevuld van de enquête
De volgende vragen gaan over de projecten waarin jij werkzaam bent geweest. Richt je op de projecten die je in vraag één hebt aangegeven.

5. In welke mate hebben onderstaande betrokkenen gebruik gemaakt van jouw software requirements?*

	Nooit	Soms	Vaak	Heel vaak
Opdrachtgever of belanghebbenden	☐	☐	☐	☐
Requirements Engineers	☐	☐	☐	☐
Applicatieontwikkelaars	☐	☐	☐	☐

Het kan lastig zijn om de software requirements zo op te stellen zodat iedereen deze begrijpt. Bij een project zijn meerdere partijen betrokken die belang hebben bij de requirements, dit zijn bijvoorbeeld opdrachtgevers en applicatieontwikkelaars. Wanneer een opdrachtgever een software requirement begrijpt, wil dit nog niet zeggen dat de requirement voldoende duidelijk is voor een applicatieontwikkelaar.

6. Kun je op de onderstaande schaal aangeven hoe bij jouw de verhouding bij het opstellen van de software requirements is tussen helemaal links *"requirements zijn begrijpelijk voor de opdrachtgever"* en helemaal rechts *"requirements zijn begrijpelijk voor de applicatieontwikkelaar"*?*

	1	2	3	4	5	6	7	
Opdrachtgever	☐	☐	☐	☐	☐	☐	☐	Applicatieontwikkelaar

10% ingevuld van de enquête
De volgende vragen gaan over de projecten waarin jij werkzaam bent geweest. Richt je op de projecten die je in vraag één hebt aangegeven.

7. Hoe vaak communiceer jij over de software requirements die jij opstelt met de opdrachtgever op de volgende manieren? *

	nooit	1x per maand	1x per twee weken	1x per week	meerdere malen per week
Mondeling contact in gesprekken met de opdrachtgever	☐	☐	☐	☐	☐
Mail contact met de opdrachtgever	☐	☐	☐	☐	☐
Communicatie via het VitalHealth Support systeem	☐	☐	☐	☐	☐

8. Zijn er nog andere manieren waarop jij communiceert met de opdrachtgever?

_____

15% ingevuld van de enquête
De volgende vragen gaan over de projecten waarin jij werkzaam bent geweest. Richt je op de projecten die je in vraag één hebt aangegeven.

9. Was de opdrachtgever in het bezit van de laatste versie van de requirements? *

☐ Nooit ☐ Soms ☐ Vaak ☐ Altijd

10. Is het voorgekomen dat de opdrachtgever de requirements die jij had opgesteld niet had doorgelezen? *

☐ Ja ☐ Nee

11. Kun je op een schaal van één tot vijf aangeven hoe nauwkeurig de opdrachtgever in de projecten de software requirements las die door jou waren opgesteld? *

	1	2	3	4	5	6	7	
zeer onnauwkeurig	☐	☐	☐	☐	☐	☐	☐	zeer nauwkeurig

12. Kun je het antwoord op de vorige vraag toelichten? *

_____

20% ingevuld van de enquête
De volgende vragen gaan over de projecten waarin jij werkzaam bent geweest. Richt je op de projecten die je in vraag één hebt aangegeven.

13. Hoe vaak komt het voor dat een opdrachtgever een woord in de software requirements niet begrijpt? *

☐ Nooit ☐ Soms ☐ Vaak ☐ Heel vaak

14. Hoe vaak komt het voor dat een opdrachtgever een software requirement anders interpreteert dan bedoeld? *

☐ Nooit ☐ Soms ☐ Vaak ☐ Heel vaak

25% ingevuld van de enquête
De volgende vragen gaan over de projecten waarin jij werkzaam bent geweest. Richt je op de projecten die je in vraag één hebt aangegeven.

15. Hoe vaak communiceer je over de software requirements die jij opstelt met de applicatieontwikkelaars op de volgende manieren? *

	nooit	1x per maand	1x per twee weken	1x per week	meerdere malen per week
Mondeling contact in gesprekken met de applicatieontwikkelaars	☐	☐	☐	☐	☐
Mail contact met de applicatieontwikkelaars	☐	☐	☐	☐	☐
Communicatie via het VitalHealth Support systeem met de applicatieontwikkelaars	☐	☐	☐	☐	☐

16. Zijn er nog andere manieren waarop jij communiceert met de applicatieontwikkelaars?

_____

30% ingevuld van de enquête
De volgende vragen gaan over de projecten waarin jij werkzaam bent geweest. Richt je op de projecten die je in vraag één hebt aangegeven.

17. Waren de applicatieontwikkelaars in het bezit van de laatste versie van de requirements? *

☐ Nooit ☐ Soms ☐ Vaak ☐ Altijd

18. Is het voorgekomen dat de applicatieontwikkelaars de requirements die jij had opgesteld niet hadden doorgelezen? *

☐ Ja ☐ Nee

19. Kun je op een schaal van één tot vijf aangeven hoe nauwkeurig de applicatieontwikkelaars in de projecten de software requirements lazen die jij had opgesteld? *

	1	2	3	4	5	6	7	
zeer onnauwkeurig	☐	☐	☐	☐	☐	☐	☐	zeer nauwkeurig

20. Kun je het antwoord op de vorige vraag toelichten? *

_____

35% ingevuld van de enquête
De volgende vragen gaan over de projecten waarin jij werkzaam bent geweest. Richt je op de projecten die je in vraag één hebt aangegeven.

21. Hoe vaak komt het voor dat een applicatieontwikkelaar een woord in de software requirements niet begrijpt? *

☐ Nooit ☐ Soms ☐ Vaak ☐ Heel vaak

22. Hoe vaak komt het voor dat een applicatieontwikkelaar een software requirement anders interpreteert als bedoeld? *

☐ Nooit ☐ Soms ☐ Vaak ☐ Heel vaak

40% ingevuld van de enquête
De volgende vragen gaan over de projecten waarin jij werkzaam bent geweest. Richt je op de projecten die je in vraag één hebt aangegeven.

Door gesprekken met de opdrachtgevers en andere belanghebbenden krijg jij kennis van het domein. Om de requirements in de applicatie te maken heeft de applicatieontwikkelaar soms ook kennis nodig van het domein.

23. Hoe vaak heb jij in de projecten de volgende activiteiten toegepast om de domeinkennis over te dragen? *

	Nooit	Soms	Vaak	Heel vaak
Mondelinge toelichting geven aan de applicatieontwikkelaars bij de requirements.	☐	☐	☐	☐
Een toelichting schrijven bij de requirements met extra domein-informatie voor de applicatieontwikkelaar.	☐	☐	☐	☐
Opnemen van een woordenlijst bij de requirements waarin domein-specifieke woorden zijn uitgelegd.	☐	☐	☐	☐

24. Zijn er nog andere belangrijke activiteiten waarvan je gebruik hebt gemaakt in de projecten om domeinkennis over te brengen op de applicatieontwikkelaars?

_____

45% ingevuld van de enquête
De volgende vragen gaan over de projecten waarin jij werkzaam bent geweest. Richt je op de projecten die je in vraag één hebt aangegeven.

Bij het opstellen van software requirements kun je gebruik maken van verschillende notatietechnieken om requirements te definiëren. Er is een aantal van deze technieken in omloop, variërend van informeel (natuurlijke taal, use cases en plaatjes) tot zeer formeel (wiskundig).

25. Heb jij in de projecten gebruik gemaakt van een notatietechniek bij het opstellen van de software requirements? *

☐ Ja ☐ Nee

Indien ja:

26. Welke notatietechniek heb je in de projecten gebruikt voor het opstellen van de software requirements? *

_____

27. Kun je de reden aangeven waarom jij voor deze notatietechniek koos? *

☐ Ik heb voor deze notatietechniek gekozen omdat ik hier ervaring mee heb.

☐ Ik heb voor deze notatietechniek gekozen omdat deze standaard binnen ons project wordt gebruikt.

☐ Ik heb voor deze notatietechniek gekozen omdat de opdrachtgever hier de voorkeur aan gaf.

☐ Ik heb voor deze notatietechniek gekozen omdat de applicatieontwikkelaar hier de voorkeur aan gaf.

☐ Anders:

GA VERDER NAAR PAGINA 17

8% ingevuld van de enquête
De volgende vragen gaan over de projecten waarin jij werkzaam bent geweest. Richt je op de projecten die je in vraag één hebt aangegeven.

In de vragen gaat het over een requirements engineer. Dit is een persoon die software requirements opstelt of beheert. In het geval van VitalHealth is dit meestal een consultant.

28. Hoe vaak communiceer jij over de software requirements met de requirements engineer op de volgende manier? *

	nooit	1x per maand	1x per twee weken	1x per week	meerdere malen per week
Mondeling contact in gesprekken met de requirements engineer	☐	☐	☐	☐	☐
Mail contact met de requirements engineer	☐	☐	☐	☐	☐
Communicatie via het VitalHealth Support systeem	☐	☐	☐	☐	☐

29. Zijn er nog andere manieren waarop jij communiceert met de requirements engineer?

_____

15% ingevuld van de enquête
De volgende vragen gaan over de projecten waarin jij werkzaam bent geweest. Richt je op de projecten die je in vraag één hebt aangegeven.

30. Was je in het bezit van de laatste versie van de requirements? *

☐ Nooit ☐ Soms ☐ Vaak ☐ Altijd

31. Is het voorgekomen dat je de software requirements niet had doorgelezen? *

☐ Ja ☐ Nee

32. Kun je op een schaal van één tot zeven aangeven hoe nauwkeurig je de software requirements las die waren opgesteld? *

	1	2	3	4	5	6	7	
zeer onnauwkeurig	☐	☐	☐	☐	☐	☐	☐	zeer nauwkeurig

22% ingevuld van de enquête
De volgende vragen gaan over de projecten waarin jij werkzaam bent geweest. Richt je op de projecten die je in vraag één hebt aangegeven.

33. Hoe vaak komt het voor dat je een woord in de software requirements niet begrijpt? *

☐ Nooit ☐ Soms ☐ Vaak ☐ Heel vaak

34. Hoe vaak komt het voor dat je een software requirement anders interpreteert dan bedoeld? *

☐ Nooit ☐ Soms ☐ Vaak ☐ Heel vaak

30% ingevuld van de enquête
De volgende vragen gaan over de projecten waarin jij werkzaam bent geweest. Richt je op de projecten die je in vraag één hebt aangegeven.

Door gesprekken met de opdrachtgevers en andere belanghebbenden krijgt de requirements engineer kennis van het domein. Om de requirements in de applicatie te maken heb jij soms ook kennis nodig van het domein.

35. Hoe vaak heb jij op de volgende manieren domeinkennis overgedragen gekregen van de requirements engineer?*

	Nooit	Soms	Vaak	Heel vaak
Door mondelinge toelichting van de requirements engineer	☐	☐	☐	☐
Doordat de requirements engineer een toelichting had geschreven bij de requirements	☐	☐	☐	☐
Doordat de requirements engineer een woordenlijst bij de requirements had opgenomen waarin domein-specifieke woorden zijn uitgelegd.	☐	☐	☐	☐

36. Zijn er nog andere belangrijke activiteiten waarmee de requirements engineer domeinkennis aan jou heeft overgedragen?

_____

35% ingevuld van de enquête
De volgende vragen gaan over de projecten waarin jij werkzaam bent geweest. Richt je op de projecten die je in vraag één hebt aangegeven.

Bij het opstellen van software requirements kun je gebruik maken van verschillende notatietechnieken om requirements te definiëren. Er is een aantal van deze technieken in omloop, variërend van informeel (natuurlijke taal, use cases en plaatjes) tot zeer formeel (wiskundig).

37. Heb jij in de projecten te maken gehad met een notatietechniek voor software requirements? *

☐ Ja ☐ Nee

INDIEN JA: -> GA VERDER
INDIEN NEE: -> GA NAAR DE VOLGENDE PAGINA

Welke notatietechniek werd gebruikt voor het opstellen van de software requirements? *

_____

38. Kun je de reden aangeven waarom de requirements engineer voor deze notatietechniek koos?*

☐ Ik heb geen idee

☐ De requirements engineer koos deze notatietechniek omdat hij hier ervaring mee had.

☐ De requirements engineer koos deze notatietechniek omdat deze standaard binnen het project werd gebruikt.

☐ De requirements engineer koos deze notatietechniek omdat de opdrachtgever hier de voorkeur aan gaf.

☐ De requirements engineer koos deze notatietechniek omdat de applicatieontwikkelaar hier de voorkeur aan gaf.

☐ Anders:

55% ingevuld van de enquête
In de volgende vragen wordt jouw persoonlijke mening gevraagd over software requirements.

Het kan voorkomen dat een requirement door een opdrachtgever of applicatieontwikkelaar niet goed wordt geïnterpreteerd. Dit kan meerdere oorzaken hebben.

39. Hoe vaak denk jij dat de volgende factoren maken dat requirements verkeerd worden geïnterpreteerd? *

	Nooit	Soms	Vaak	Heel vaak
Een woord in de requirement is voor meerdere uitleg vatbaar.	☐	☐	☐	☐
De zinsstructuur in de requirement is verwarrend waardoor de requirement voor meerdere uitleg vatbaar is.	☐	☐	☐	☐
De requirement is verwarrend met een andere requirement in hetzelfde requirementsdocument	☐	☐	☐	☐
De requirement is verwarrend met een andere requirement in het domein waarin de applicatie wordt gemaakt.	☐	☐	☐	☐

60% ingevuld van de enquête
In de volgende vragen wordt jouw persoonlijke mening gevraagd over software requirements.

40. Welke van de onderstaande twee stellingen is van toepassing op jou? *

☐ Het heeft mijn voorkeur om de requirements declaratief te definiëren. Bij het opstellen van de requirements beschrijven wat het systeem moet doen.

☐ Het heeft mijn voorkeur om de requirements procedureel te definiëren. Bij het opstellen van de requirements beschrijven hoe het systeem moet gaan werken.

41. Welke vorm heeft bij jou de voorkeur bij het opstellen van de requirements? *

☐ In een stukje tekst een aantal requirements beschrijven.

☐ Requirements opstellen in een lijst, waarbij iedere requirement een item in de lijst is.

42. Wil je aangeven waarom dit jouw voorkeur heeft? *

_____

65% ingevuld van de enquête
In de volgende vragen wordt jouw persoonlijke mening gevraagd over software requirements.

43. Hoe belangrijk vind jij het dat de software requirements "consistent" zijn? *
Dat wil zeggen dat een set requirement consistent is wanneer er geen requirements in de requirements set met elkaar conflicteren.

☐ Zeer onbelangrijk ☐ Onbelangrijk ☐ Neutraal ☐ Belangrijk ☐ Zeer belangrijk

44. Hoe belangrijk vind jij het dat de software requirements "eenduidig" zijn? *
Dat wil zeggen een requirement kan maar op één manier worden uitgelegd, de requirement heeft maar één betekenis in het domein.

☐ Zeer onbelangrijk ☐ Onbelangrijk ☐ Neutraal ☐ Belangrijk ☐ Zeer belangrijk

45. Hoe belangrijk vind jij het dat de software requirements "ontwerp onafhankelijk" zijn? *
De requirements zijn onafhankelijk van de implementatie wijze in de applicatie.

☐ Zeer onbelangrijk ☐ Onbelangrijk ☐ Neutraal ☐ Belangrijk ☐ Zeer belangrijk

70% ingevuld van de enquête
In de volgende vragen wordt jouw persoonlijke mening gevraagd over software requirements.

46. Hoe belangrijk vind jij het dat de software requirements "traceerbaar" zijn? *
Dat wil zeggen de requirements zijn goed traceerbaar wanneer de relatie tussen de verschillende requirements is vastgelegd.

☐ Zeer onbelangrijk ☐ Onbelangrijk ☐ Neutraal ☐ Belangrijk ☐ Zeer belangrijk

47. **Hoe belangrijk vind jij het dat de software requirements "verifieerbaar" zijn?** *
Dat wil zeggen de requirements zijn goed verifieerbaar wanneer via de requirements kan worden getest of de applicatie aan de wensen van de opdrachtgever voldoet. Zinsneden als 'het systeem moet gebruikersvriendelijk zijn' zijn niet te testen.

☐ Zeer onbelangrijk ☐ Onbelangrijk ☐ Neutraal ☐ Belangrijk ☐ Zeer belangrijk

75% ingevuld van de enquête
In de vorige vragen heb je aangegeven welke eigenschappen je belangrijk vindt voor de kwaliteit van de requirements. Met deze eigenschappen kan de kwaliteit van de requirements worden verbeterd. Er is een aantal manieren die ervoor zorgen de requirements aan die eigenschappen voldoen. Eén manier is om bij het opstellen van de requirements gebruik te maken van een gecontroleerde taal, een CNL (Controlled Natural Language). Bij een gecontroleerde taal worden er restricties op de grammatica en woordenschat van een gewonen taal gelegd zodat de taal consistenter en eenduidiger wordt. Veel gecontroleerde talen kunnen eenvoudig worden omgezet naar een formele taal. Een formele taal is een kunstmatige taal waarvan de vorm en de betekenis exact is vastgelegd, vaak door middel van wiskundige definities.

Een voorbeeld is de volgende requirement: *"Members are assigned to one practice, but can also work in other practices with the same network."* Deze requirement is niet eenduidig en kan op meerdere manieren worden uitgelegd. Het laatste deel van de requirement kan worden uitgelegd als *"praktijken met hetzelfde netwerk"* of als *"de member kan werken met hetzelfde netwerk als in de praktijk waarbij hij hoort in een andere praktijk"*. Deze requirement is geanalyseerd en opnieuw opgesteld via de gecontroleerde taal ACE met het volgende resultaat: *"The members are assigned to one practice. The member can work in a other practice that has the same network."*

48. Heb jij weleens met een gecontroleerde taal gewerkt? *

☐ Ja ☐ Nee

49. Heb jij weleens met een formele taal gewerkt? *

☐ Ja ☐ Nee

80% ingevuld van de enquête
Met een gecontroleerde taal kan de kwaliteit van de requirements verbeterd worden. De eigenschappen die jij als belangrijk voor de requirements hebt opgegeven, daar kan een requirement aan voldoen die is opgesteld met een gecontroleerde taal.

50. Hoeveel tijd wil jij ongeveer investeren om een gecontroleerde taal te leren waarmee je software requirements maakt? *

☐ 1 dag
☐ 3 dagen
☐ 1 werkweek
☐ 2 werkweken
☐ langer dan 2 werkweken

85% ingevuld van de enquête
In deze vragen wordt jouw persoonlijke mening gevraagd over het opstellen van software requirements met een gecontroleerde taal. De volgende vragen hebben betrekking op taal eigenschappen die je graag wilt blijven gebruiken wanneer je requirements opstelt met een gecontroleerde taal.

51. Hoe belangrijk vind jij het dat "beeldspraak" mag gebruikt worden in de software requirements? *
Dat wil zeggen dat je in de requirements gebruik maakt van zinsdelen zoals "als een kapstok gebruiken" of "in de database kijken".

☐ Zeer onbelangrijk ☐ Onbelangrijk ☐ Neutraal ☐ Belangrijk ☐ Zeer belangrijk

52. Hoe belangrijk vind jij het dat een "gezegde" mag gebruikt worden in de software requirements? *
Dat wil zeggen dat je in de requirements gebruik maakt van zinsdelen zoals "Aan het roer zitten".

☐ Zeer onbelangrijk ☐ Onbelangrijk ☐ Neutraal ☐ Belangrijk ☐ Zeer belangrijk

53. Hoe belangrijk vind jij het dat een "spreekwoord" mag gebruikt worden in de software requirements? *
Dat wil zeggen dat je in de requirements gebruik maakt van zinsdelen zoals "De beste stuurlui staan aan wal".

☐ Zeer onbelangrijk ☐ Onbelangrijk ☐ Neutraal ☐ Belangrijk ☐ Zeer belangrijk

90% ingevuld van de enquête
In deze vragen wordt jouw persoonlijke mening gevraagd over het opstellen van software requirements met een gecontroleerde taal. De volgende vragen hebben betrekking op taal eigenschappen die je graag wilt blijven gebruiken wanneer je requirements opstelt met een gecontroleerde taal.

54. Hoe belangrijk vind jij het dat "afwisseling in structuur van zinnen" mag gebruikt worden in de software requirements? *
Dat wil zeggen dat je in de requirements gebruik maakt van zinsdelen zoals "Patiënt heeft ziekte x en is gebruiker" en later voor een zin met dezelfde betekenis deze structuur "De patiënt is gebruiker en heeft ziekte x" hanteert.

☐ Zeer onbelangrijk ☐ Onbelangrijk ☐ Neutraal ☐ Belangrijk ☐ Zeer belangrijk

55. Hoe belangrijk vind jij het dat "tussenzinnen" mogen gebruikt worden in de software requirements? *
Dat wil zeggen dat je in de requirements gebruik maakt van zinsdelen zoals "De gebruiker, dit is een persoon in rol x, moet inloggen op het systeem".

☐ Zeer onbelangrijk ☐ Onbelangrijk ☐ Neutraal ☐ Belangrijk ☐ Zeer belangrijk

95% ingevuld van de enquête
In deze vragen wordt jouw persoonlijke mening gevraagd over het opstellen van software requirements met een gecontroleerde taal. De volgende vragen hebben betrekking op taal eigenschappen die je graag wilt blijven gebruiken wanneer je requirements opstelt met een gecontroleerde taal.

***56.* Hoe belangrijk vind jij het dat "opsommingen" mogen gebruikt worden in de software requirements?** *

☐ Zeer onbelangrijk ☐ Onbelangrijk ☐ Neutraal ☐ Belangrijk ☐ Zeer belangrijk

57. Hoe belangrijk vind jij het dat vanuit een requirement kan verwezen worden naar een andere requirement in het document? *

☐ Zeer onbelangrijk ☐ Onbelangrijk ☐ Neutraal ☐ Belangrijk ☐ Zeer belangrijk

***58.* Met een gecontroleerde taal die meertalig is kun je zinnen maken in bijvoorbeeld het Nederlands en het Engels. Hoe belangrijk vind jij het dat een gecontroleerde taal meerdere natuurlijke talen ondersteunt?** *

☐ Zeer onbelangrijk ☐ Onbelangrijk ☐ Neutraal ☐ Belangrijk ☐ Zeer belangrijk

99% ingevuld van de enquête

Gecontroleerde talen kunnen ondersteunt worden door een tool. In deze tools kunnen de requirements worden opgesteld. Met deze tools kunnen activiteiten worden uitgevoerd zoals, de consistentie van de requirements controleren, redundantie in requirements voorkomen en helpen bij het opstellen van de requirements door middel van stylechecks.

59. Hoe belangrijk vind jij dat de gecontroleerde taal wordt ondersteunt door een tool? *

☐ Zeer onbelangrijk ☐ Onbelangrijk ☐ Neutraal ☐ Belangrijk ☐ Zeer belangrijk

100% ingevuld van de enquête

Dankjewel dat je mee wilde werken aan dit onderzoek door middel van het invullen van deze enquête.

D Controlled Natural Language onderzoek

D.1 Attempto Controlled English (ACE)

Nauwkeurigheid

Eigenschap	Waarde
Lexicale ambiguïteit	+/- ACE beschikt over twee groepen woorden; voor gedefinieerde functiewoorden en domeinwoorden. De functiewoorden zijn standaard woorden in ACE. De domeinwoorden kunnen door de gebruiker worden opgegeven met daarbij de betekenis. Doordat gebruikers zelf de domeinwoorden kunnen opgeven is lexicale ambiguïteit mogelijk.
Syntactische ambiguïteit	+ Met ACE wordt syntactische ambiguïteit voorkomen. ACE heeft constructie - en interpretatieregels om syntactische ambiguïteit te voorkomen.
Semantische ambiguïteit	+ Met ACE wordt semantische ambiguïteit voorkomen. In ACE zijn regels om scope en referentiële ambiguïteit (zin) te voorkomen.
Pragmatische ambiguïteit	+ Referentiële ambiguïteit komt niet voor in ACE. Anaforen en antecedenten worden door syntactische regels aan elkaar verbonden.

Expressiviteit

Eigenschap	Waarde
Maakt de CNL gebruik van een eigen woordenlijst	+/- ACE heeft voor gedefinieerde functiewoorden. ACE heeft geen corpus waarin de betekenis van de woorden is opgenomen.
Kunnen er woorden worden toegevoegd aan de woordenlijst	+ In ACE kunnen er eigen domein woorden worden gedefinieerd.
Meervoud / enkelvoud	+ In ACE zijn er regels voor meervoud en enkelvoud van woorden. In ACE worden telbare meervoud woorden herkent. Voorbeeld hiervan is de zin: “3 cards”.
Heeft de CNL taal regels voor nominalisatie	- ACE heeft geen ingebouwde kennis hoe een werkwoord gerelateerd is aan een genominaliseerd zelfstandig naamwoord.
Is een andere schrijfwijze van een woord toegestaan	+ ACE geeft de gebruiker de mogelijkheid om synoniemen te definiëren (alarm signal button, signal button) en afkortingen (ABS betekent alarm signal button).
Kunnen er verwijzingen worden gemaakt.	+ ACE ondersteunt verwijzing, in ACE zijn ook voornaam woorden toegestaan. Elke anaforische referentie is een zelfstandig naamwoord zin of is een voornaamwoord, variabelen of een eigen naam
Kunnen er opsommingen worden gemaakt	+/- In ACE kunnen geen opsommingen worden gemaakt door het scheiden van items met een komma. Om een opsomming in ACE te maken moet de volgende structuur worden toegepast: A patient has a name and address and number and id (Fuchs, Kaljurand, Kuhn, & Schneider, 2006).
Zijn tussenzinnen toegestaan.	- In ACE kan geen gebruik gemaakt worden van tussenzinnen. In ACE kan een komma alleen gebruikt worden bij een conjunctie om de standaard orde in koppelwoorden te veranderen.
Is beeldspraak toegestaan.	- In ACE is beeldspraak niet toegestaan.
Zijn gezegde toegestaan.	- In ACE zijn gezegde niet toegestaan.
Zijn spreekwoorden toegestaan.	- In ACE is een spreekwoord niet toegestaan.

Natuurlijk karakter

Eigenschap	Indicatoren
Is de taal makkelijk te leren?	+/- De syntax van ACE wordt beschreven in minder dan 20 constructie regels. De constructie regels beschrijven de syntax van ACE, deze regels beperken de vorm van al de toelaatbare zinnen van ACE.

Is de taal makkelijk te lezen?	+	In ACE wordt gebruik gemaakt van de syntaxis van het Engels in een vorm met restricties op de lexicon en de grammatica. Hierdoor is ACE voor iedereen die Engels kan lezen te eenvoudig te lezen. Niet iedere zinsconstructie is toegestaan in ACE. Een voorbeeld hiervan is het gebruik van de constructie “not every”, in plaats hiervan moet gebruik worden gemaakt van “there is ... not” of “no”. In de zin “Not every card is correct” kan in de plaats van “not every” de volgende zin worden gemaakt “There is a card that is not correct” of om uit te drukken dat elke kaart niet correct is “No card is correct” (Fuchs et al., 1999).
Is de taal makkelijk te schrijven?	+	In ACE is ook syntactic sugar verwerkt, dit houdt in dat er in ACE verschillende manieren zijn om een tekst te formuleren terwijl de tekst op dezelfde manier wordt geïnterpreteerd. In ACE is syntactic sugar op meerdere levels in de taal geïmplementeerd. Op het woord niveau kunnen functiewoorden op verschillende manieren worden opgeschreven, voorbeelden ‘every’ = ‘each’ en ‘nobody’ = ‘no one’ en op het logische niveau waar bijvoorbeeld dubbele ontkenning wordt geïnterpreteerd als de afwezigheid van ontkenning. In verschil met veel formele talen, is in ACE syntactic sugar niet alleen toegestaan maar ook nauwgezet toegevoegd om de taal in gebruik makkelijker te maken. De gebruiker wordt niet verplicht om zinnen te zoals ‘every’ zinnen maar wanner de gebruiker het wil kan ook gebruik worden gemaakt van if-then zinnen (Kaljurand, 2007).
Is de taal makkelijk te begrijpen?	+	In ACE wordt gebruik gemaakt van de syntaxis van het Engels in een vorm met restricties op de lexicon en de grammatica. Hierdoor voelt ACE aan als natuurlijk Engels, ACE biedt de gebruiker dan ook veel ruimte om zinnen te creëren. Om ervoor te zorgen dat de zinnen die gemaakt worden met ACE ook leesbaar en goed te begrijpen blijven zijn bij de handleiding (Fuchs et al., 1999) van ACE aantal style richtlijnen meegegeven waaraan een goed leesbare ACE zin moet voldoen.

Eenvoud

Eigenschap		**Indicatoren**
Worden zinnen omgezet naar een formele taal.	+	ACE is een nauwkeurige gedefinieerde subset van het Engels die door een computer kan worden vertaald naar eerste orde logica. Hierdoor is ACE een eerste orde logica taal met als syntax een subset van het Engels, waardoor ACE leesbaar is voor mens en computer
Wordt de discours omgezet naar een formele taal.	+/-	In ACE wordt er geen specifiek applicatie domein ondersteunt. ACE heeft in zich zelf geen kennis of ontologie van het applicatie domein, of van software engineerings methode of de wereld in het algemeen (Fuchs et al., 1990). Gebruikers moeten zelf de discource definiëren met ACE. Dit kan doormiddel van het aanvullen van de woordenschat of het definiëren van ACE zinnen met daarin kennis van het domein.
Is er een mapping naar een grafische presentatie mogelijk.	-	nee
Ondersteunt de CNL taal meerder natuurlijke talen.	-	ACE is eentalig, met ACE wordt alleen het Engels ondersteunt.
Beschikt de CNL over een style richtlijn	+	In de Attempto Controlled English (ACE) Language Manual (Fuchs, Schwertel & Schwitter, 1999) is een style guide opgenomen voor de taal.
Wordt de CNL taal ondersteunt door een tool?	+/-	Voor ACE is de tool APE beschikbaar. De tool, Attempto Parsing Engine (APE), kan ACE zinnen vetalen naar eerste orde logica.

D.2 Processable ENGlish (PENG)

Nauwkeurigheid

Eigenschap		**Indicatoren**
Lexicale ambiguïteit	+/-	PENG beschikt over twee groepen woorden; voor gedefinieerde functiewoorden en domeinwoorden. De domeinwoorden kunnen door de gebruiker worden opgegeven met daarbij de betekenis.
Syntactische ambiguïteit	+	PENG heeft zeven interpretatie principes, hiermee wordt o.a. syntactische ambiguïteit in PENG zinnen voorkomen.
Semantische ambiguïteit	+	PENG heeft zeven interpretatie principes, hiermee wordt o.a. semantische ambiguïteit in PENG zinnen voorkomen.

Pragmatische ambiguïteit	+	In PENG wordt met het anafora principe referentiële ambiguïteit voorkomen.

Expressiviteit

Eigenschap	Waarde	
Maakt de CNL gebruik van een eigen woordenlijst	+/-	PENG heeft voor gedefinieerde functiewoorden. PENG heeft geen corpus waarin de betekenis van de woorden is opgenomen.
Kunnen er woorden worden toegevoegd aan de woordenlijst	+	In PENG kunnen er eigen domein woorden worden gedefinieerd.
Meervoud / enkelvoud	+	In PENG mag gebruik worden gemaakt vaan meervoud.
Heeft de CNL taal regels voor nominalisatie	-	PENG heeft geen kennis hoe een werkwoord gerelateerd is aan een genominaliseerd zelfstandig naamwoord.
Is een andere schrijfwijze van een woord toegestaan	+	In PENG mag de auteur gebruik maak van synoniemen voor content woorden, acroniemen of afkortingen voor zelfstandig naamwoorden.
Kunnen er verwijzingen worden gemaakt.	+	Met het anafora principe ondersteunt PENG het gebruik van verwijzingen en voornaamwoorden.
Kunnen er opsommingen worden gemaakt	-	In PENG kunnen geen opsommingen worden gemaakt. De zin *"Wolves, foxes, birds, caterpillars, and snails are ani- mals, and there are some of each of them."* wordt op de volgende manier omgezet naar PENG: *"Wolves are animals. Foxes are animals. Birds are an- imals. Caterpillars are animals. Snails are animals. There are some wolves and some foxes. There are some caterpillars and some snails."* (Schwitter, 2002).
Zijn tussenzinnen toegestaan.	-	Tussenzinnen zijn niet toegestaan in PENG.
Is beeldspraak toegestaan.	-	Beeldspraak zijn niet toegestaan in PENG.
Zijn gezegde toegestaan.	-	Gezegde zijn niet toegestaan in PENG.
Zijn spreekwoorden toegestaan.	-	Spreekwoorden zijn niet toegestaan in PENG.

Natuurlijk karakter

Eigenschap	Indicatoren	
Is de taal makkelijk te leren?	+	Voor het schrijven van zinnen in PENG met behulp van de tool heeft de auteur geen kennis nodig van de restrictie regels waaraan een zin moet voldoen. Tijdens het schrijven van een zin geeft de tool aan welke mogelijkheden er kunnen volgen op de ingevoerde tekst
Is de taal makkelijk te lezen?	+	In PENG wordt gebruik gemaakt van de syntaxis en lexicon van de natuurlijke taal Engels. De zinnen zijn een natuurlijke representatie van formele zinnen.
Is de taal makkelijk te schrijven?	+	
Is de taal makkelijk te begrijpen?	+	

Eenvoud

Eigenschap		Indicatoren
Worden zinnen omgezet naar een formele taal.	+	Specificatie teksten die geschreven zijn in PENG lijken informeel, maar de taal ontworpen om dezelfde nauwkeurigheid van een formele taal te hebben. Zinnen die gemaakt zijn in PENG kunnen één op één worden omgezet naar FOL.
Wordt de discours omgezet naar een formele taal.	+/-	In PENG wordt er geen specifiek applicatiedomein ondersteunt. Gebruikers moeten zelf de discource definiëren met PENG. Dit kan doormiddel van het aanvullen van de woordenschat of het definiëren van PENG zinnen met daarin kennis van het domein.
Is er een mapping naar een grafische presentatie mogelijk.	-	nee

Ondersteunt de CNL taal meerder natuurlijke talen.	-	PENG is niet meertalig, met PENG wordt alleen het Engels ondersteunt.
Beschikt de CNL over een style richtlijn	+	In PENG wordt de gebruiker geleid door de tool. In de tool zijn richtlijnen voor de taal verwerkt. De tool geeft de gebruiker aanbevelingen voor de style.
Wordt de CNL taal ondersteunt door een tool?	+	Ja, de PENG editor. De tool die gebruikt wordt met PENG herkent de restrictie terwijl de zinnen worden opgebouwd, hierdoor kan de tool voorstellen aan de gebruiker doen hoe een zin opgesteld moet worden.

D.3 Common Logic Controlled English (CLCE)

Nauwkeurigheid

Eigenschap	**Waarde**	
Lexicale ambiguïteit	+/-	CLCE beschikt over twee groepen woorden; voor gedefinieerde functiewoorden en domeinwoorden. De domeinwoorden kunnen door de gebruiker worden opgegeven met daarbij de betekenis.
Syntactische ambiguïteit	+	Elke zin in CLCE kan één op één worden omgezet naar een formele zin. Hierdoor heeft elke zin maar één betekenis en wordt syntactische ambiguïteit voorkomen.
Semantische ambiguïteit	+	Elke zin in CLCE kan één op één worden omgezet naar een formele zin. Hierdoor heeft elke zin maar één betekenis en wordt semantische ambiguïteit voorkomen.
Pragmatische ambiguïteit	+	Om referentiële ambiguïteit te voorkomen worden in CLCE zelfstandig voornaamwoorden omgezet naar variabelen.

Expressiviteit

Eigenschap	**Waarde**	
Maakt de CNL gebruik van een eigen woordenlijst	+/-	De woorden die gebruikt worden in CLCE kunnen worden onderverdeeld in twee groepen, er is een kleine groep met daarin een aantal gereserveerde woorden. De gereserveerde woorden worden op dezelfde manier gespeld als de Engelse woorden, maar de betekenis van de woorden is beperkt tot één betekenis.
Kunnen er woorden worden toegevoegd aan de woordenlijst	+	In CLCE kunnen er eigen domein woorden worden gedefinieerd.
Meervoud / enkelvoud	-	Om taalfout ambiguïteit in CLCE te voorkomen zijn er aan meervoud regels verbonden.
Heeft de CNL taal regels voor nominalisatie	-	CLCE heeft geen regels voor nominalisatie.
Is een andere schrijfwijze van een woord toegestaan	-	In CLCE is er geen mogelijkheid om afkortingen of synoniemen op te geven voor een woord.
Kunnen er verwijzingen worden gemaakt.	+	In CLCE worden (zelfstandig) voornaamwoorden omgezet naar variabelen. Hier kunnen anaforen naar verwijzen.
Kunnen er opsommingen worden gemaakt	+	In CLCE kunnen er opsommingen worden gemaakt waarbij de items zijn gescheiden door een komma.
Zijn tussenzinnen toegestaan.	-	In CLCE zijn tussenzinnen niet toegestaan.
Is beeldspraak toegestaan.	-	In CLCE is beeldspraak niet toegestaan.
Zijn gezegde toegestaan.	-	In CLCE zijn gezegde niet toegestaan.
Zijn spreekwoorden toegestaan.	-	In CLCE is een spreekwoord niet toegestaan.

Natuurlijk karakter

Eigenschap	**Indicatoren**	
Is de taal makkelijk te leren?	-	In CLCE wordt gebruik gemaakt van de grammatica en semantiek van formele talen. De regels voor de grammatica en semantiek zijn niet expliciet in CLCE gedefinieerd.
Is de taal makkelijk te lezen?	+	

Is de taal makkelijk te schrijven?	-	In CLCE wordt gebruik gemaakt van de grammatica van formele talen. In CLCE is geen extra syntactic sugar toegepast om zinnen makkelijker te laten opschrijven.
Is de taal makkelijk te begrijpen?	+	De zinnen die gemaakt zijn met CLCE voelen natuurlijk aan. De betekenis die een zin heeft komt overeen met de betekenis in een natuurlijke taal.

Eenvoud

Eigenschap		**Indicatoren**
Worden zinnen omgezet naar een formele taal.	+	CLCE zinnen kunnen één op één worden overgezet op eerste orde logica. Zinnen die gemaakt zijn in een FOL kunnen ook worden vertaald naar CLCE.
Wordt de discours omgezet naar een formele taal.	+/-	Gebruikers moeten zelf de discource definiëren met CLCE. Dit kan door het aanvullen van de woordenschat of het definiëren van CLCE zinnen met daarin kennis van het domein.
Is er een mapping naar een grafische presentatie mogelijk.	+	
Ondersteunt de CNL taal meerder natuurlijke talen.	-	CLCE is eentalig, met ACE wordt alleen het Engels ondersteunt.
Beschikt de CNL over een style richtlijn	-	Nee
Wordt de CNL taal ondersteunt door een tool?	-	Er is geen tool beschikbaar voor CLCE.

D.4 Computer Processable Language (CPL)

Nauwkeurigheid

Eigenschap	**Waarde**	
Lexicale ambiguïteit	+	Om lexicale ambiguïteit in woorden tegen te gaan gebruikt CPL WordNet als ontologie. De betekenis de woorden wordt gepresenteerd aan de gebruiker voor verificatie of voor correctie.
Syntactische ambiguïteit	+/-	In CPL zijn er regels in de grammatica om ambiguïteit te voorkomen. Syntactische ambiguïteit wordt niet expliciet voorkomen in CPL.
Semantische ambiguïteit	+/-	In CPL zijn er regels in de grammatica om ambiguïteit te voorkomen. Semantische ambiguïteit wordt niet expliciet voorkomen in CPL.
Pragmatische ambiguïteit	+	In CPL zijn geen voornaamwoorden toegestaan. Het verwijzen in een zin naar een andere zin met een voornaamwoord is niet mogelijk in CPL.

Expressiviteit

Eigenschap	**Waarde**	
Maakt de CNL gebruik van een eigen woordenlijst	+	CPL gebruikt WordNet als ontologie.
Kunnen er woorden worden toegevoegd aan de woordenlijst	-	In CPL wordt de betekenis van een woord uit de WordNet database gehaald. Wanneer een woord niet voorkomt in de database kan deze niet worden toegevoegd. Het woord heeft dan geen betekenis in de CPL zin.
Meervoud / enkelvoud	+	De CLP interpretator voert de meervoud regels uit
Heeft de CNL taal regels voor nominalisatie	+	De CLP interpretator voert de nominalistie regels uit. Door WordNet8 lexicon legt CPL de relatie tussen een werkwoord en een genominaliseerd zelfstandig naamwoord.
Is een andere schrijfwijze van een woord toegestaan	+	Synoniemen zijn toegestaan in CPL. CPL gebruikt WordNet als ontologie om de betekenis van elk woord vast te stellen, zo ook van de synoniemen.
Kunnen er verwijzingen worden gemaakt.	+/-	In CPL zijn geen voornaamwoorden toegestaan. In de taal worden wel relaties gelegd tussen object en instanties van het object.
Kunnen er opsommingen worden gemaakt	+/-	
Zijn tussenzinnen toegestaan.	-	In de richtlijnen van CPL staat dat er per zin maar één clausule gebruikt mag worden.

Is beeldspraak toegestaan.	-	In CPL is beeldspraak niet toegestaan.
Zijn gezegde toegestaan.	-	In CPL zijn gezegde niet toegestaan.
Zijn spreekwoorden toegestaan.	-	In CPL is een spreekwoord niet toegestaan.

Natuurlijk karakter

Eigenschap	**Indicatoren**	
Is de taal makkelijk te leren?	-	Clark (2005) noemt gebruiksvriendelijkheid één van de uitdagingen die nog verbeterd moeten worden aan CPL. Een gebruiker moet ervaren zijn om goed met CPL te kunnen werken. Bij het opstellen van een CPL zin in de tool moet de gebruiker nog rekening houden met de semantiek van de zin.
Is de taal makkelijk te lezen?	+	
Is de taal makkelijk te schrijven?	+/-	De regels van CPL zijn niet zo strikt als die van ACE of PENG. Hierdoor is het mogelijk dat er ambiguïteit in de zinnen voorkomt. De gebruiker moet bij het opstellen van de CPL zinnen hier rekening mee houden.
Is de taal makkelijk te begrijpen?	+	In CPL wordt gebruik gemaakt van de syntaxis van het Engels in een vorm met restricties op de lexicon en de grammatica. CPL voelt aan als natuurlijk Engels. Om te zorgen dat de zinnen goed leesbaar zijn heeft CPL een style handleiding waarin beschreven is hoe leesbare zinnen moeten worden opgesteld.

Eenvoud

Eigenschap		**Indicatoren**
Worden zinnen omgezet naar een formele taal.	+/-	CPL zinnen worden door de CPL interpretator omgezet naar formele zinnen met een één op veel relatie, er komt nog ambiguïteit in voor. Het is aan de gebruiker om de juiste interpretatie uit te kiezen.
Wordt de discours omgezet naar een formele taal.	+/-	Gebruikers moeten zelf de discource definiëren met CPL. Dit kan door het aanvullen van de woordenschat of het definiëren van CPL zinnen met daarin kennis van het domein.
Is er een mapping naar een grafische presentatie mogelijk.	-	nee
Ondersteunt de CNL taal meerder natuurlijke talen.	-	CPL is eentalig, met ACE wordt alleen het Engels ondersteunt.
Beschikt de CNL over een style richtlijn	+	Ja. Om zinnen met CPL op te stellen is er een set van richtlijnen beschikbaar. Sommige van deze richtlijnen zijn aanbevelingen om ambiguïteit en misinterpretatie te voorkomen (Clark et al., 2007).
Wordt de CNL taal ondersteunt door een tool?	+	Ja, de CPL interpretator die zinnen vertaald naar FOL met de Knowledge Machine (KM). De taal geeft aanwijzingen welke betekenissen een zin kan hebben.

E Requirements in een gecontroleerde natuurlijke taal

Requirement 1 (project 1)

Origineel	The authorization is done via the login with username and password.
CNL	The authorization is done via the login that has a username and password.
Ambiguïteit	*Attachment ambiguïteit (opgelost)*

Requirement 2 (project 1)

Origineel	The different tests and measurements done by the technician can be different in multiple practices.
CNL	A technician can measure on more than 1 practice. The technician can test on more than 1 practice.
Ambiguïteit	*Coördinatie ambiguïteit (opgelost)*

Requirement 3 (project 1)

Origineel	The summary overview will display the latest medications and information in a single view.
CNL	A summary-overview displays the latest information in a single view and displays the relevant medication in a single view.
Ambiguïteit	*Coördinatie ambiguïteit (opgelost)*

Requirement 4 (project 3)

Origineel	Members are external users of type Referring Physician or Support Staff.
CNL	A member is an external user and is a type Referring Physician or is a type Support Staff.
Ambiguïteit	*Coördinatie ambiguïteit (opgelost)*

Requirement 5 (project 3)

Origineel	Members are assigned to one practice, but can also work in other practices with the same network.
CNL	The members are assigned to one practice. The member can work in a other practice with the same network. Network: A network is a number of connected computers.
Ambiguïteit	*Lexicale en attachment ambiguïteit (opgelost)*

Requirement 6 (project 3)

Origineel	Please note that the Self-sign on form is filled in by an anonymous user. This is viewed as a separate portal, in which the user will only be able to access the self-sign on form.
CNL	A 'Self-sign on' form is filled by an anonymous user. The 'Self-sign on' form is viewed as a separate portal. A user can only access the 'Self-sign on' form in the separate portal.
Ambiguïteit	*Referentiële ambiguïteit (zin) en Referentiële ambiguïteit (context) (opgelost)*